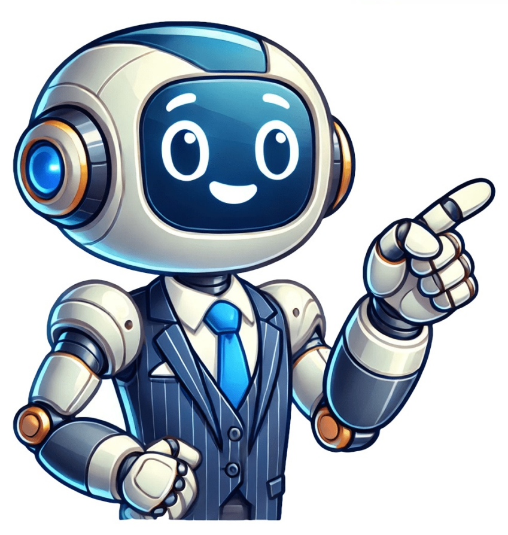
I'm not a robot



Seems like cookies are disabled on this browser, please enable them to open this website It includes AI that generates DSA solutions in four languages: C, C++, Java, and Python. Seems like cookies are disabled on this browser, please enable them to open this website Source: Update 18 October 2019: I have created a curation of Leetcode problems which I personally use to prepare for technical interviews. Stars are welcome, and feel free to fork it for your own modification and use! Ill be adding more questions in time! :)If youre looking for a new job, use Triplebyte to interview once and apply to multiple top tech companies!This post draws on my personal experiences and challenges over the past term at school, which I entered with hardly any knowledge of DSA (data structures and algorithms) and problem-solving strategies. As a self-taught programmer, I was a lot more familiar and comfortable with general programming, such as object-oriented programming, than with the problem-solving skills required in DSA questions.This post reflects my journey throughout the term and the resources I turned to in order to quickly improve my data structures, algorithms, and problem-solving skills.Problem: You know the theory, but you get stuck on practical applicationsI faced this issue early in the term when I didnt know what I didnt know, which is a particularly pernicious problem. I understood the theory well enough for instance, what a linked list was, how it worked, its various operations and their time complexities, the ADTs (abstract data types) it supported, and how the ADT operations were implemented.But because I didnt know what I didnt know, I couldnt identify gaps in my understanding of its practical applications in problem-solving.The different types of questionsAn example of a data structures question: describe how you would insert a node in a linked list and state the time complexity.And heres an algorithms question: search for an element in a rotated sorted array and state the time complexity.Finally, a problem-solving question, which I consider to be at a higher level than the previous two, might briefly describe a scenario, and list the requirements of the problem. In an exam it might ask for a description of the solution. In competitive programming it might require you to submit working code without explicitly providing any data structures or algorithms. In other words, you are expected to apply the most applicable data structures and algorithms to solve the problem as efficiently as possible.How can you improve your data structures, algorithms, and problem-solving skills?I primarily use three websites for practice: HackerRank, LeetCode, and Kattis. They are largely similar, especially the first two, but not identical. I find that each site has a slightly different focus, each of which is immensely helpful in its own way.I would loosely categorize the skills required for problem-solving into:knowledge of data structuresknowledge of algorithmsknowledge of the application of data structures and algorithmsThe first two could be considered the primitives, or building blocks, that go into the third, which is about knowing what to apply for a particular scenario.Knowledge of data structuresIn this respect, I found HackerRank to be a valuable resource. It has a section dedicated to data structures, which you can filter by type, such as arrays, linked lists, (balanced) trees, heaps, and so forth.The questions are not so much about problem-solving as they are about working with data structures. For instance:You get the idea. Some of the questions might not ever be directly applicable in problem-solving. But they are great for conceptual understanding, which is extremely important in any case.HackerRank does not have freely accessible model solutions, although the discussions section is usually full of hints, clues, and even working code snippets. I have found those to be adequate so far, although you might have to step through the code a line at a time in an IDE to really understand something.Knowledge of algorithmsHackerRank also has an algorithms section, although I prefer LeetCode for this. I found LeetCodes variety of problems to be a lot wider, and I really like that a lot of problems have solutions with explanations and even time complexities.A great starting point would be LeetCodes top 100 liked questions. Some questions which I thought were great:Unlike data structures questions, the focus here isnt so much about working with or manipulating data structures, but rather, how to do something. For instance, the accounts merge problem is primarily on the application of standard UFDS algorithms. The searching in a rotated sorted array problem presents a twist on binary search. And sometimes you learn an entirely new problem-solving technique. For example, the sliding window solution for the longest continuous increasing subsequence problem.Knowledge of the application of data structures and algorithmsFinally, I use Kattis to improve my general problem-solving skills. The Kattis Problem Archive has a bunch of programming problems from various sources, such as competitive programming competitions, around the world.Kattis can be incredibly frustrating because there are no official solutions or a discussion forum, (unlike HackerRank and LeetCode). Also, test cases are private. I have a handful of pending Kattis problems which I cant solve not because I dont know the solution, but because I cant figure out the bug.Its my least favorite site among the three for practicing and learning, and I didnt spend a lot of time on it.Other resourcesGeeksforgeeks is another very valuable resource for learning about data structures and algorithms. I like how it provides code snippets in various languages, usuallyC++, Java, and Python, which you can copy and paste into your IDE to step through line-by-line.Finally, there is trusty old Google, which would lead you to GeeksForGeeks most of the time, and Youtube, for visual explanations.ConclusionAt the end of the day, however, there are no shortcuts. You just have to dive into it head-first start writing code, debugging code, and reading other peoples correct code to figure out where, how, and why you went wrong. Its tough, but you get better with each attempt, and it gets easier as you get better.Im nowhere near the level of competency I want to be, but Ive definitely come a long way since I started. :) CategoryArrayBacktrackingBinary TreeBit ManipulationBSTDivide & ConquerDynamic ProgrammingGraphHeapLinked ListMatrixProgramming PuzzlesQueueSortingStackStringTrie TagAlgorithmBinary SearchBottom-upBreadth-First SearchDepth-First SearchGreedyHashingMust KnowPriority QueueRecursiveSliding WindowTop-down DifficultyEasyMediumHardBeginner Curated ListsTop 100 Most LikedTop 50 ClassicTop 25 AlgorithmsTop 10 DP1. Find a pair with the given sum in an arrayArray, SortingAmazon, HashingTopClassic, TopLikedEasy2. Check if a subarray with 0 sum exists or notArrayHashingTopLikedMedium3. Print all subarrays with 0 sumArrayAmazon, HashingTopLikedMedium4. Sort binary array in linear timeArray, SortingTopLikedEasy5. Find maximum length subarray having a given sumArrayHashingTopLikedMedium6. Find the largest subarray having an equal number of 0s and 1sArrayHashingTopLikedMedium7. Find the maximum product of two integers in an arrayArray, SortingTopLikedEasy8. Sort an array of 0s, 1s, and 2s (Dutch National Flag Problem)Array, SortingAlgorithm, Amazon, MicrosoftTopClassic, TopLikedMedium9. In-place merge two sorted arraysArray, SortingTopLikedMedium10. Merge two arrays by satisfying given constraintsArray, SortingMedium11. Find index of 0 to be replaced to get the maximum length sequence of continuous onesArrayAmazonHard12. Shuffle an array using FisherYates shuffle algorithmArrayAlgorithmMedium13. Rearrange an array with alternate high and low elementsArrayAmazonTopLikedMedium14. Find equilibrium index of an arrayArrayAmazonEasy15. Find the largest subarray formed by consecutive integersArrayAmazon, HashingTopClassic, TopLikedMedium16. BoyerMoore Majority Vote AlgorithmArrayAlgorithm, Amazon, Hashing, MicrosoftTopAlgoEasy17. Move all zeros present in an array to the endArray, SortingAmazonTopLikedEasy18. Replace every array element with the product of every other elementArrayRecursiveMedium19. Longest Bitonic Subarray ProblemArrayAlgorithmTopClassicMedium20. Find maximum difference between two array elements that satisfies given constraintsArrayAmazonTopLikedMedium21. Maximum Sum Subarray Problem (Kadanes Algorithm)Array, DPAlgorithm, Amazon, Microsoft, Must KnowTopAlgo, TopClassic, TopAlgoEasy22. Print continuous subarray with maximum sumArrayMedium23. Maximum Sum Circular SubarrayArrayAlgorithmTopLikedHard24. Find all distinct combinations of a given length IArray, SortingRecursiveTopLikedMedium25. Find all distinct combinations of a given length IIArray, SortingRecursiveMedium26. Find maximum sequence of continuous 1s formed by replacing at-most k 0s by 1sArraySliding WindowMedium27. Find minimum sum subarray of size kArrayAmazon, Sliding WindowMedium28. Maximum Product Subarray ProblemArrayAlgorithm, AmazonTopClassic, TopLikedHard29. Find a subarray having the given sum in an integer arrayArrayAmazon, Hashing, Sliding WindowMedium30. Find the smallest subarray length whose sum of elements is greater than kArraySliding WindowMedium31. Find the smallest window in an array sorting which will make the entire array sortedArray, SortingMedium32. Find maximum sum path involving elements of given arraysArrayAmazon, MicrosoftMedium33. Find maximum profit earned by buying and selling shares any number of timesArrayAmazonTopLikedMedium34. Trapping Rain Water ProblemArrayAlgorithm, Amazon, MicrosoftTopClassicHard35. Find minimum platforms needed to avoid delay in the train arrivalArray, SortingGreedyMedium36. Decode an array constructed from another arrayArrayMedium37. Sort an array in one swap whose two elements are swappedArray, SortingEasy38. Find a triplet with the given sum in an arrayArray, SortingHashing, RecursiveTopClassic, TopLikedMedium39. Find the longest continuous sequence length with the same sum in given binary arraysArrayHashingHard40. Reverse every consecutive m-elements of a subarrayArrayRecursiveMedium41. Maximum Product Subset ProblemArrayAlgorithm, RecursiveTopClassicEasy42. Find pairs with difference k in an arrayArray, SortingHashingEasy43. 4Sum Problem | Quadruplets with a given sumArray, SortingAlgorithm, Hashing, RecursiveTopClassic, TopLikedMedium44. Print all quadruplets with a given sum | 4 sum problem extendedArray, SortingMedium45. Count quadruplets with a zero sumArrayHashingMedium46. Quickselect AlgorithmArrayAlgorithm, RecursiveTopAlgoMedium47. Rearrange array such that A[A[i]] is set to i for every element A[i]ArrayHard48. Print all triplets that form an arithmetic progressionArrayMedium49. Print all triplets that form a geometric progressionArrayMedium50. Group elements of an array based on their first occurrenceArrayHashingMedium51. Find minimum difference between the index of two given elements present in an arrayArrayEasy52. Find maximum absolute difference between the sum of two non-overlapping subarraysArrayHard53. Find all symmetric pairs in an array of pairsArrayHashingMedium54. Find the closest pair to a given sum in two sorted arraysArrayMedium55. Partition an array into two subarrays with the same sumArrayEasy56. Find the count of distinct elements in every subarray of size kArrayHashing, Microsoft, Sliding WindowMedium57. Find two numbers with maximum sum formed by array digitsArray, SortingEasy58. Print all subarrays of an array having distinct elementsArrayHashing, Sliding WindowMedium59. Find a triplet having the maximum product in an arrayArray, SortingMedium60. Find the minimum index of a repeating element in an arrayArrayHashingEasy61. Find a pair with a minimum absolute sum in an arrayArray, SortingEasy62. Find an index of the maximum occurring element with equal probabilityArrayHashingEasy63. Check if an array is formed by consecutive integersArrayHashingMedium64. Find two non-overlapping pairs having the same sum in an arrayArrayHashingMedium65. Add elements of two arrays into a new arrayArrayRecursiveEasy66. Find minimum product among all combinations of triplets in an arrayArray, SortingMedium67. Count distinct absolute values in a sorted arrayArrayHashing, Sliding WindowMedium68. Print all combinations of positive integers in increasing order that sums to a numberArrayRecursiveHard69. Find subarrays with a given sum in an arrayArrayHashingTopLikedMedium70. Find maximum length sequence of continuous ones (Using Sliding Window)ArrayAmazon, Sliding WindowMedium71. Find maximum length sequence of continuous onesArrayAmazonMedium72. Find the index that divides an array into two non-empty subarrays with equal sumArrayEasy73. Efficiently calculate the frequency of all elements present in a limited range arrayArrayHashingMedium74. Rearrange an array such that it contains alternate positive and negative numbersArray, SortingMedium75. Find the sorted triplet in an arrayArrayMedium76. Shuffle an array according to the given order of elementsArrayHashingMedium77. Count the number of strictly increasing subarrays in an arrayArrayMedium78. Find duplicates within a range k in an arrayArrayHashing, Sliding WindowEasy79. Find a minimum range with at least one element from each of the given arraysArrayMicrosoftHard80. Find the longest subsequence formed by consecutive integersArrayHashingTopClassicMedium81. Determine the index of an element that satisfies given constraints in an arrayArrayEasy82. Find minimum moves required for converting a given array to an array of zeroesArrayMedium83. Left rotate an arrayArrayEasy84. Right rotate an array k timesArrayTopLikedEasy85. Activity Selection ProblemArray, SortingAlgorithm, Amazon, GreedyTopClassicEasy86. Job Sequencing Problem with DeadlinesArray, SortingAlgorithm, GreedyTopClassic, TopLikedMedium87. 3partition problem extended | Printing all partitionsArrayRecursiveHard88. Count triplets which form an inversion in an arrayArrayEasy89. Determine whether an array can be divided into pairs with a sum divisible by kArrayHashingMedium90. Find minimum removals required in an array to satisfy given constraintsArrayMedium91. Find a pair with the given sum in a circularly sorted arrayArrayMedium92. Segregate positive and negative integers in linear timeArray, SortingEasy93. Find all distinct combinations of a given length that sum to a targetArrayRecursionMedium94. Find all duplicate elements in a limited range arrayHashingEasy95. Find the minimum and maximum element in an array using minimum comparisonsArrayMedium96. Insertion Sort AlgorithmArray, SortingAlgorithm, Must Know, RecursiveTopAlgoEasy97. Selection Sort AlgorithmArray, SortingAlgorithm, Must Know, RecursiveTopAlgoEasy98. Bubble Sort AlgorithmArray, SortingAlgorithm, RecursiveEasy99. Merge Sort AlgorithmArray, Divide & Conquer, SortingAlgorithm, Microsoft, Must Know, RecursiveTopAlgoEasy100. Iterative Merge Sort Algorithm (Bottom-up Merge Sort)Array, Divide & Conquer, SortingAlgorithmMedium101. Quicksort AlgorithmArray, Divide & Conquer, SortingAlgorithm, Must Know, RecursiveTopAlgoMedium102. Hybrid QuickSort AlgorithmArray, Divide & Conquer, SortingAlgorithm, RecursiveMedium103. Quicksort using Dutch National Flag AlgorithmArray, Divide & Conquer, SortingAlgorithm, RecursiveMedium104. Quicksort algorithm using Hoares partitioning schemeArray, Divide & Conquer, SortingAlgorithm, RecursiveMedium105. Counting Sort AlgorithmArray, SortingAlgorithm, Must KnowTopAlgoEasy106. In-place vs out-of-place algorithmsSortingAlgorithm, Must KnowBeginner107. Inversion count of an arrayArray, Divide & Conquer, SortingAlgorithm, Amazon, Microsoft, RecursiveTopLikedHard108. Problems solved using partitioning logic of QuicksortArray, SortingEasy109. Sort elements by their frequency and indexArray, SortingAmazon, Hashing, MicrosoftMedium110. Sort an array based on order defined by another arrayArray, SortingAmazon, HashingMedium111. Efficiently sort an array with many duplicated valuesArray, SortingHashingMedium112. Find the largest number possible from a given set of numbersArray, SortingTopLikedMedium113. Find surpasser count for each array elementArray, SortingHashing, RecursiveHard114. Segregate positive and negative integers using merge sortArray, Divide & Conquer, SortingMedium115. How to Boost QuickSort Performance?SortingRecursiveEasy116. Water Jugs ProblemArray, SortingAlgorithm, RecursiveTopClassicHard117. Print matrix in spiral orderMatrixAmazon, RecursiveTopLikedMedium118. Create a spiral matrix from a given arrayMatrixMedium119. Shift all matrix elements by 1 in spiral orderMatrixMedium120. Change all elements of row i and column j in a matrix to 0 if cell (i, j) is 0MatrixAmazonTopLikedMedium121. Print diagonal elements of a matrix having a positive slopeMatrixMedium122. Replace all occurrences of 0 that are not surrounded by 1 in a binary matrixMatrixDepth-First Search, RecursiveMedium123. In-place rotate matrix by 90 degrees in a clockwise directionMatrixAmazonEasy124. Count negative elements present in the sorted matrix in linear timeMatrixEasy125. Report all occurrences of an element in a row-wise and column-wise sorted matrixMatrixMedium126. Check if a matrix is a Toeplitz or notMatrixEasy127. In-place rotate matrix by 180 degreesMatrixMedium128. Fill binary matrix with alternating rectangles of 0 and 1MatrixMedium129. Find all common elements present in each row of a matrixMatrixHashingMedium130. Find common elements present in all rows of a matrixMatrixHashingMedium131. Find the index of a row containing the maximum number of 1s in a binary matrixMatrixHard132. Find the largest square submatrix which is surrounded by all 1sMatrixMedium133. Print a spiral square matrix without using any extra spaceMatrixHard134. Young Tableau | Insert, Search, Extract-Min, Delete, ReplaceMatrixAlgorithm, RecursiveHard135. Replace all occurrences of 0 that are surrounded by 1 in a binary matrixMatrixDepth-First Search, RecursiveMedium136. Find the area of the largest rectangle of 1s in a binary matrixMatrixHard137. Find maximum value of M[c][d] M[a][b] over all choices of indexesMatrixMedium138. Generate pascal triangle of the given sizeMatrixEasy139. Find perimeter of an IslandMatrixEasy140. Find kth smallest value in a sorted matrixMatrix, Divide & ConquerBinary SearchMedium141. Sort an array using Young tableauArray, Matrix, SortingRecursiveHard142. Print all possible solutions to NQueens problemBacktracking, MatrixAlgorithm, RecursiveTopClassic, TopLikedHard143. Print all possible Knights tours on a chessboardBacktracking, MatrixRecursiveTopClassic, TopLikedHard144. Find the shortest path in a mazeBacktracking, MatrixMaze, RecursiveTopLikedMedium145. Find the longest possible route in a matrixBacktracking, MatrixMaze, RecursiveMedium146. Find the path from source to destination in a matrix that satisfies given constraintsBacktracking, MatrixDepth-First Search, Maze, RecursiveMedium147. Find the total number of unique paths in a maze from source to destinationBacktracking, MatrixMaze, RecursiveTopLikedMedium148. Find all combinations of elements satisfying given constraintsArray, BacktrackingAmazon, RecursiveMedium149. KPartition Problem | Printing all partitionsArray, BacktrackingAlgorithm, RecursiveTopClassic, TopLikedHard150. Magnet PuzzleBacktracking, MatrixRecursiveHard151. Find all paths from the first cell to the last cell of a matrixBacktracking, MatrixAmazon, RecursiveMedium152. Print all shortest routes in a rectangular gridBacktracking, MatrixRecursiveMedium153. Find all distinct combinations of a given length with repetition allowedArray, Backtracking, SortingRecursiveMedium154. Print all combinations of numbers from 1 to n having sum nArray, BacktrackingRecursiveTopLikedMedium155. Print all triplets in an array with a sum less than or equal to a given numberArray, Backtracking, SortingMedium156. Check if a string is a rotated palindrome or notStringRecursiveMedium157. Longest Palindromic Substring ProblemStringAlgorithm, MicrosoftTopClassic, TopLikedMedium158. Check if a repeated subsequence is present in a string or notStringHashing, RecursiveHard159. Check if strings can be derived from each other by circularly rotating themStringEasy160. Check if a set of moves is circular or notStringAmazonMedium161. Convert a number into a corresponding excel column nameStringAmazon, MicrosoftMedium162. Convert column name in Excel to the corresponding numberStringEasy163. Find all interleaving of given stringsStringRecursiveEasy164. Isomorphic StringsStringHashingMedium165. Remove all extra spaces from a stringStringMedium166. Find all possible palindromic substrings of a stringStringTopLikedHard167. Find all possible combinations of words formed from the mobile keypadStringAmazon, RecursiveTopLikedHard168. Find all combinations by replacing given digits with corresponding list charactersBacktracking, StringHashing, RecursiveHard169. Find all words that follow the same order of characters as given patternStringAmazon, HashingMedium170. Group anagrams together from a list of wordsSorting, StringHashing, MicrosoftMedium171. Find minimum operations required to transform a string into another stringSorting, StringHashingHard172. Determine whether a string can be transformed into another string in a single editStringMedium173. Remove all occurrences of AB and C from a stringStringEasy174. Find the longest even-length palindromic sum substring of a stringStringMedium175. Print string in the zigzag form in k rowsStringMedium176. Run Length Encoding (RLE) Data Compression AlgorithmStringAlgorithm, Amazon, Microsoft, Must KnowEasy177. Find the longest substring of a string containing k distinct charactersStringHashing, Sliding WindowHard178. Find all palindromic permutations of a stringSorting, StringHashingMedium179. Find all substrings of a string that are a permutation of another stringStringHashing, Microsoft, Sliding WindowMedium180. Find the longest substring of a string containing distinct charactersStringAmazon, Microsoft, Sliding WindowMedium181. Find all permutations of a string C++, Java, PythonBacktracking, StringAmazon, Must Know, RecursiveHard182. Iterative approach to finding permutations of a stringJava, Sorting, StringHard183. Find all lexicographically next permutations of a stringSorting, StringAmazon, MicrosoftHard184. Lexicographically Minimal String RotationStringAlgorithmMedium185. Find all strings of a given length containing balanced parenthesesStringRecursiveMedium186. Find all combinations of non-overlapping substrings of a stringBacktracking, StringAmazon, RecursiveMedium187. Determine whether a string is a palindrome or notBasic, StringRecursiveEasy188. Find the minimum number of inversions needed to make an expression balancedStringMedium189. Construct the longest palindrome by shuffling or deleting characters from a stringStringHashingMedium190. Print all combinations of phrases formed by picking words from each of the given listsStringRecursiveMedium191. Break a string into all possible combinations of non-overlapping substringsStringRecursiveMedium192. Convert a Roman numeral to an IntegerStringEasy193. Remove adjacent duplicate characters from a stringStringRecursiveEasy194. Find the first non-repeating character in a string by doing only one traversal of itStringHashingMedium195. Find all n-digit strictly increasing numbers (Bottom-up and Top-down approach)StringRecursiveMedium196. Find all n-digit binary numbers having more 1s than 0s for any prefixStringRecursiveMedium197. Find all n-digit numbers with a given sum of digitsStringRecursiveHard198. Find all n-digit binary numbers with k-bits set where k ranges from 1 to nStringHard199. Find all n-digit binary numbers with an equal sum of bits in their two halvesStringRecursiveHard200. Find all n-digit numbers with equal sum of digits at even and odd indicesBacktracking, StringRecursiveHard201. Find all lexicographic permutations of a stringSorting, StringRecursiveHard202. Determine if a string is a subsequence of another stringStringEasy203. Find all lexicographically previous permutations of a stringStringHard204. Replace all non-overlapping occurrences of a patternStringMedium205. Find all substrings containing exactly k distinct charactersStringHashingMedium206. Introduction to Pattern MatchingStringMust KnowBeginner207. KMP AlgorithmC, C++, Java, StringAlgorithm, Must KnowTopAlgoHard208. Reverse a string using recursionBasic, StringRecursiveEasy209. Determine whether the characters of a string follow a specified order or notStringMedium210. Check if a sentence is syntactically correct or notStringMedium211. Check a string for repeated substringsStringEasy212. Find difference between two stringsStringEasy213. Construct smallest number after removing k digits from a stringStringMedium214. Number to word conversionC++, Java, Python, StringMicrosoft, RecursiveHard215. Find all occurrences of the given string in a character matrixBacktracking, Matrix, StringDepth-First Search, RecursiveHard216. Shortest Superstring ProblemStringGreedyHard217. Find the shortest route in a device to construct a given stringMatrix, StringMedium218. Find the minimum number possible by doing at-most k swapsBacktracking, StringRecursiveMedium219. Determine whether a string matches with a given patternBacktracking, StringHashing, RecursiveHard220. Difference between Subarray, Subsequence, and SubsetArray, Basic, StringMust KnowBeginner221. Determine whether two strings are anagram or notStringHashingEasy222. Bit Hacks Part 1 (Basic)Bit ManipulationEasy223. Bit Hacks Part 2 (Playing with kth bit)Bit ManipulationEasy224. Bit Hacks Part 3 (Playing with the rightmost set bit of a number)Bit ManipulationEasy225. Bit Hacks Part 4 (Playing with letters of the English alphabet)Bit ManipulationEasy226. Bit Hacks Part 5 (Find the absolute value of an integer without branching)Bit ManipulationEasy227. Find the total number of bits needed to be flippedBit ManipulationEasy228. Brian Kernighans Algorithm to count set bits in an integerBit ManipulationAlgorithm, AmazonEasy229. Round up to the next highest power of 2Bit ManipulationMedium230. Round up to the previous power of 2Bit ManipulationMedium231. Compute the parity of a number using a lookup tableBit ManipulationHard232. Count set bits using a lookup tableBit ManipulationAmazonHard233. Multiply 16-bit integers using an 8-bit multiplierBit ManipulationMedium234. Swap two bits at a given position in an integerBit ManipulationMedium235. Swap individual bits at a given position in an integerBit ManipulationHard236. Check if a number is a power of 4 or notBit ManipulationMedium237. Calculate hamming distance between two integersBit ManipulationEasy238. Generate an array with the set bit count of each indexBit ManipulationEasy239. Reverse bits of an integerBit ManipulationMedium240. Print binary representation of a numberBasic, Bit Manipulation, C, C++, Java, PythonRecursiveEasy241. Add binary representation of two integersBit ManipulationEasy242. Swap adjacent bits of a numberBit ManipulationMedium243. Check if adjacent bits are set in the binary representation of a numberBit ManipulationEasy244. Reverse bits of an integer using a lookup tableBit ManipulationHard245. Circular shift on the binary representation of an integer by k positionsBit ManipulationMedium246. Find XOR of two numbers without using the XOR operatorBit ManipulationMedium247. Print all distinct subsets of a given setArray, Backtracking, Bit Manipulation, SortingRecursiveTopLikedHard248. Find the missing number in an arrayArray, Bit ManipulationEasy249. Find the missing number in an array without using any extra spaceArray, Bit ManipulationTopLikedEasy250. Find the odd occurring element in an array in a single traversalArray, Bit ManipulationHashingEasy251. Find two odd occurring elements in an array without using any extra spaceArray, Bit ManipulationHashingMedium252. Find all odd occurring elements in an array having a limited range of elementsArray, Bit ManipulationMedium253. Find the duplicate element in a limited range arrayArray, Bit ManipulationAmazon, Hashing, MicrosoftTopLikedMedium254. Find two duplicate elements in a limited range array (using XOR)Array, Bit ManipulationHashingMedium255. Find the missing number and duplicate elements in an arrayArray, Bit ManipulationMedium256. Stack implementation using an array C, C++, C++ (Using Templates), Java, PythonStackMust KnowBeginner257. Check if an expression is balanced or notStack, StringEasy258. Find duplicate parenthesis in an expressionStack, StringAmazonMedium259. Evaluate a postfix expressionStack, StringTopLikedEasy260. Decode a given sequence to construct a minimum number without repeated digitsStack, StringAmazonHard261. Design a stack that returns the minimum element in constant timeStackHard262. Design a stack that returns a minimum element without using an auxiliary stackStackHard263. Merging Overlapping IntervalsArray, Sorting, StackAlgorithm, AmazonTopClassicMedium264. Maximum Overlapping Intervals ProblemArray, SortingAlgorithmTopClassicMedium265. Insert an interval by merging overlapping intervalsArrayMedium266. Convert an infix expression into a postfix expressionStack, StringMedium267. Implement two stacks in a single arrayStackEasy268. Recursive solution to sort a stackStackRecursiveHard269. Reverse a stack using recursionStackRecursiveHard270. Reverse a string using a stack data structureStack, StringEasy271. Find the next greater element for every array elementArray, StackMedium272. Find the next greater element for every element in a circular arrayArray, StackHard273. Find the previous smaller element for each array elementArray, StackMedium274. Reverse an array in C++Array, Basic, C++, StackRecursiveEasy275. Longest Increasing Subsequence ProblemArray, StackAlgorithm, AmazonTopClassicHard276. Find all increasing subsequences of an arrayArray, BacktrackingRecursiveMedium277. Find all elements in an array that are greater than all elements to their rightArray, StackEasy278. Iterative Implementation of QuicksortArray, Divide & Conquer, Sorting, StackMedium279. Find all binary strings that can be formed from a wildcard patternBacktracking, Stack, StringRecursiveMedium280. Find the length of the longest balanced parenthesis in a stringStack, StringHard281. Reverse text without reversing individual wordsStack, StringMedium282. Evaluate a given expressionString, StackHard283. Reverse a string without using recursionBasic, C++, Java, Stack, StringEasy284. Construct a string from an encoded sequenceString, StackHard285. Inorder Tree TraversalBinary Tree, StackAlgorithm, Depth-First Search, Must Know, RecursiveTopLikedMedium286. Preorder Tree TraversalBinary Tree, StackAlgorithm, Depth-First Search, Must Know, RecursiveTopLikedMedium287. Postorder Tree TraversalBinary Tree, StackAlgorithm, Depth-First Search, Must Know, RecursiveTopLikedMedium288. Level order traversal of a binary treeBinary Tree, QueueAlgorithm, Amazon, Breadth-First Search, Depth-First Search, Hashing, Microsoft, Must Know, RecursiveTopLikedEasy289. Check if two binary trees are identical or notBinary Tree, StackAmazon, Microsoft, RecursiveTopLikedEasy290. Print bottom view of a binary treeBinary TreeAmazon, Depth-First Search, Hashing, RecursiveTopLikedMedium291. Print top view of a binary treeBinary TreeDepth-First Search, Hashing, RecursiveTopLikedMedium292. Calculate the height of a binary treeBinary Tree, QueueAmazon, Breadth-First Search, Depth-First Search, RecursiveEasy293. Delete a binary treeBinary Tree, QueueBreadth-First Search, Depth-First Search, RecursiveEasy294. Spiral order traversal of a binary treeBinary Tree, QueueAlgorithm, Amazon, Breadth-First Search, Depth-First Search, Hashing, Microsoft, RecursiveTopLikedMedium295. Reverse level order traversal of a binary treeBinary Tree, Queue, StackAlgorithm, Amazon, Breadth-First Search, Depth-First Search, Hashing, Microsoft, RecursiveEasy296. In-place convert a binary tree to its sum treeBinary TreeAmazon, Depth-First Search, Microsoft, RecursiveEasy297. Determine whether the given binary tree nodes are cousins of each otherBinary TreeDepth-First Search, RecursiveMedium298. Print cousins of a given node in a binary treeBinary TreeDepth-First Search, RecursiveMedium299. Check if a binary tree is a sum tree or notBinary TreeAmazon, Depth-First Search, RecursiveMedium300. Combinations of words formed by replacing given numbers with corresponding alphabetsArray, Binary Tree, StringAmazon, RecursiveHard301. Determine whether a binary tree is a subtree of another binary treeBinary TreeDepth-First Search, RecursiveMedium302. Find the diameter of a binary treeBinary TreeAmazon, Depth-First Search, Microsoft, RecursiveTopLikedMedium303. Check if a binary tree is symmetric or notBinary TreeAmazon, Microsoft, RecursiveEasy304. Convert a binary tree to its mirrorBinary TreeDepth-First Search, RecursiveEasy305. Determine if a binary tree can be converted to another by swapping childrenBinary TreeRecursiveEasy306. Find the Lowest Common Ancestor (LCA) of two nodes in a binary treeBinary TreeAmazon, Microsoft, RecursiveTopLikedMedium307. Print all paths from the root to leaf nodes of a binary treeBinary Tree, BacktrackingAmazon, Depth-First Search, Microsoft, RecursiveEasy308. Find ancestors of a given node in a binary treeBinary Tree, StackDepth-First Search, Hashing, RecursiveTopLikedMedium309. Find distance between given pairs of nodes in a binary treeBinary TreeAmazon, RecursiveHard310. Find the diagonal sum of a binary treeBinary TreeDepth-First Search, Hashing, RecursiveMedium311. Sink nodes containing zero to the bottom of a binary treeBinary TreeDepth-First Search, RecursiveHard312. Convert a binary tree to a full tree by removing half nodesBinary TreeDepth-First Search, RecursiveMedium313. Truncate a binary tree to remove nodes that lie on a path having a sum less than kBinary TreeAmazon, Depth-First Search, RecursiveMedium314. Find maximum sum root to leaf path in a binary treeBinary TreeAmazon, Depth-First Search, RecursiveMedium315. Check if a binary tree is height-balanced or notBinary TreeDepth-First Search, RecursiveMedium316. Convert binary tree to Left-child right-sibling binary treeBinary TreeDepth-First Search, RecursiveMedium317. Print all paths from leaf to root node of a binary treeBinary Tree, BacktrackingDepth-First Search, RecursiveMedium318. Iteratively print the leaf to root path for every leaf node in a binary treeBinary Tree, StackDepth-First Search, HashingMedium319. Build a binary tree from a parent arrayBinary TreeAmazon, Hashing, MicrosoftTopLikedHard320. Find all nodes at a given distance from leaf nodes in a binary treeBinary TreeDepth-First Search, RecursiveHard321. Count all subtrees having the same value of nodes in a binary treeBinary TreeDepth-First Search, RecursiveMedium322. Find the maximum difference between a node and its descendants in a binary treeBinary TreeDepth-First Search, RecursiveMedium323. Find the maximum sum path between two leaves in a binary treeBinary TreeRecursiveHard324. Construct a binary tree from inorder and preorder traversalBinary TreeDepth-First Search, Hashing, RecursiveTopLikedHard325. Construct a binary tree from inorder and postorder traversalsBinary TreeDepth-First Search, Hashing, RecursiveHard326. Construct a binary tree from inorder and level order sequenceBinary TreeDepth-First Search, Hashing, RecursiveHard327. Construct a full binary tree from the preorder sequence with leaf node informationBinary TreeDepth-First Search, RecursiveHard328. Construct a full binary tree from a preorder and postorder sequenceBinary TreeDepth-First Search, Hashing, RecursiveHard329. Find postorder traversal of a binary tree from its inorder and preorder sequenceBinary TreeDepth-First Search, Hashing, RecursiveMedium330. Set next pointer to the inorder successor of all nodes in a binary treeBinary TreeDepth-First Search, RecursiveEasy331. Find preorder traversal of a binary tree from its inorder and postorder sequenceBinary Tree, StackDepth-First Search, Hashing, RecursiveHard332. Find difference between sum of all nodes present at odd and even levels in a binary treeBinary TreeRecursiveEasy333. Clone a binary treeBinary TreeRecursiveEasy334. Clone a binary tree with random pointersBinary TreeDepth-First Search, Hashing, RecursiveHard335. Threaded Binary Tree Overview and ImplementationBinary TreeDepth-First Search, RecursiveMedium336. Determine if a binary tree satisfies the height-balanced property of a redblack treeBinary TreeDepth-First Search, RecursiveMedium337. Construct an ancestor matrix from a binary treeBinary Tree, MatrixDepth-First Search, RecursiveEasy338. Find all possible binary trees having the same inorder traversalBinary TreeDepth-First Search, RecursiveHard339. Perform boundary traversal on a binary treeBinary TreeDepth-First Search, RecursiveMedium340. Check if binary representation of a number is palindrome or notBit ManipulationEasy341. Check if each node of a binary tree has exactly one childBinary TreeDepth-First Search, RecursiveEasy342. Evaluate a Binary Expression TreeBinary TreeDepth-First Search, RecursiveEasy343. Construction of an expression treeBinary Tree, StackDepth-First Search, RecursiveEasy344. Fix children-sum property in a binary treeBinary TreeDepth-First Search, RecursiveMedium345. Maximum path sum in a binary treeBinary TreeAlgorithm, RecursiveHard346. Create a mirror of an mary treeBinary TreeDepth-First Search, RecursiveEasy347. Print a two-dimensional view of a binary treeBinary TreeDepth-First Search, RecursiveEasy348. Construct a binary tree from an ancestor matrixBinary Tree, MatrixHashingHard349. Insertion in a BSTBSTAlgorithm, Amazon, Microsoft, Must Know, RecursiveTopLikedEasy350. Search a given key in BSTBSTAlgorithm, Must Know, RecursiveTopLikedEasy351. Deletion from BST (Binary Search Tree)BSTAlgorithm, Amazon, Must Know, RecursiveTopLikedMedium352. Construct a balanced BST from the given keysBST, SortingAmazon, RecursiveEasy353. Determine whether a given binary tree is a BST or notBinary Tree, BSTAmazon, Depth-First Search, Microsoft, RecursiveTopLikedMedium354. Check if the given keys represent the same BSTs or not without building BSTBSTRecursiveHard355. Find inorder predecessor for the given key in a BSTBSTRecursiveMedium356. Find the Lowest Common Ancestor (LCA) of two nodes in a BSTBSTAmazon, RecursiveEasy357. Find kth smallest node in a BSTBSTDepth-First Search, RecursiveEasy358. Find kth largest node in a BSTBSTDepth-First Search, RecursiveTopLikedEasy359. Find floor and ceil in a Binary Search TreeBSTRecursiveMedium360. Convert a binary tree to BST by maintaining its original structureBSTDepth-First Search, RecursiveMedium361. Remove nodes from a BST that have keys outside a valid rangeBSTDepth-First Search, RecursiveMedium362. Find a pair with the given sum in a BSTBSTDepth-First Search, Hashing, RecursiveEasy363. Find inorder successor for the given key in a BSTBinary Tree, BSTRecursiveTopLikedMedium364. Replace every array element with the least greater element on its rightArray, BSTMedium365. Fix a binary tree that is only one swap away from becoming a BSTBinary Tree, BSTDepth-First Search, RecursiveHard366. Update every key in a BST to contain the sum of all greater keysBSTDepth-First Search, RecursiveMedium367. Check if a given sequence represents the preorder traversal of a BSTBSTDepth-First Search, RecursiveHard368. Build a Binary Search Tree from a postorder sequenceBSTDepth-First Search, RecursiveHard369. Build a Binary Search Tree from a preorder sequenceBSTDepth-First Search, RecursiveTopLikedHard370. Count subtrees in a BST whose nodes lie within a given rangeBSTDepth-First Search, RecursiveMedium371. Find the size of the largest BST in a binary treeBinary Tree, BSTDepth-First Search, RecursiveHard372. Calculate sum of root to leaf digits in a binary treeBinary Tree, QueueRecursiveMedium373. Count paths with the given sum in a binary treeBinary TreeRecursive, HashingHard374. Print complete Binary Search Tree (BST) in increasing orderArray, BST, StackDepth-First Search, RecursiveEasy375. Print binary tree structure with its contentsBinary Tree, BSTRecursiveMedium376. Binary Search AlgorithmArray, Divide & ConquerAlgorithm, Binary Search, Must Know, RecursiveTopAlgoEasy377. Find the number of rotations in a circularly sorted arrayArray, Divide & ConquerAmazon, Binary Search, RecursiveTopLikedEasy378. Search an element in a circularly sorted arrayArray, Divide & ConquerAmazon, Binary Search, MicrosoftMedium379. Find the first or last occurrence of a given number in a sorted arrayArray, Divide & ConquerBinary SearchTopLikedEasy380. Count occurrences of a number in a sorted array with duplicatesArray, Divide & ConquerBinary SearchTopLikedMedium381. Find the smallest missing element from a sorted arrayArray, Divide & ConquerBinary Search, RecursiveMedium382. Find floor and ceil of a number in a sorted integer arrayArray, Divide & ConquerAmazon, Binary SearchEasy383. Search in a nearly sorted array in logarithmic timeArray, Divide & ConquerBinary SearchMedium384. Find the number of 1s in a sorted binary arrayArray, Divide & ConquerBinary Search, RecursiveEasy385. Find the peak element in an arrayArray, Divide & ConquerAmazon, Binary Search, RecursiveTopLikedMedium386. Maximum Subarray Sum using Divide and ConquerArray, Divide & ConquerAlgorithm, RecursiveTopLikedMedium387. Efficiently implement power functionBit Manipulation, Divide & ConquerRecursiveEasy388. Find the missing term in a sequence in logarithmic timeArray, Divide & ConquerBinary SearchMedium389. Find floor and ceil of a number in a sorted array (Recursive solution)Array, Divide & ConquerAmazon, Binary Search, RecursiveEasy390. Find the frequency of each element in a sorted array containing duplicatesArray, Divide & ConquerBinary Search, Hashing, RecursiveEasy391. Find the square root of a number using a binary searchDivide & ConquerBinary SearchEasy392. Division of two numbers using binary search algorithmDivide & ConquerAmazon, Binary SearchMedium393. Find the odd occurring element in an array in logarithmic timeArray, Bit Manipulation, Divide & Conquer, SortingBinary Search, RecursiveMedium394. Find pairs with difference k in an array | Constant Space SolutionArray, Divide & Conquer, SortingBinary Search, HashingMedium395. Find k closest elements to a given value in an arrayArray, Divide & ConquerBinary SearchMedium396. Find the maximum value of j i such that A[j] > A[i] in an arrayArrayMedium397. Longest Common Prefix (LCP) ProblemDivide & Conquer, StringAlgorithm, RecursiveTopClassicEasy398. Binary Search in C++ STL and Java CollectionsArray, C++, Divide & Conquer, JavaAlgorithm, Binary SearchBeginner399. Ternary Search vs Binary searchArray, Divide & ConquerAlgorithm, Binary SearchBeginner400. Exponential searchArray, Divide & ConquerAlgorithm, Binary Search, RecursiveTopAlgoEasy401. Unbounded Binary SearchDivide & ConquerAlgorithm, Binary Search, RecursiveEasy402. Interpolation searchArray, Divide & ConquerAlgorithmEasy403. Introduction to Dynamic ProgrammingDynamic ProgrammingBottom-up, Recursive, Top-down, Must KnowBeginner404. Longest Common Subsequence ProblemDynamic Programming, StringAlgorithm, Amazon, Bottom-up, Recursive, Top-downTopClassic, TopLiked, TopDPMedium405. Longest Common Subsequence (LCS) | Space optimized versionDynamic Programming, StringAmazon, Bottom-upMedium406. Longest Common Subsequence of ksequencesDynamic Programming, StringAlgorithm, Bottom-up, Recursive, Top-downMedium407. Longest Common Subsequence | Finding all LCSDynamic Programming, StringAmazon, Bottom-up, Recursive, Top-downHard408. Longest Common Substring ProblemDynamic Programming, StringAlgorithm, Bottom-upTopClassic, TopLikedMedium409. Longest Palindromic Subsequence using Dynamic ProgrammingDynamic Programming, StringAlgorithm, Bottom-up, Microsoft, Recursive, Top-downTopClassic, TopLikedMedium410. Longest Repeated Subsequence ProblemDynamic Programming, StringAlgorithm, Bottom-up, Recursive, Top-downTopClassic, TopLikedMedium411. Implement Diff UtilityDynamic Programming, StringAlgorithm, Recursive, Bottom-up, Top-downMedium412. Shortest Common Supersequence ProblemDynamic Programming, StringAlgorithm, Bottom-up, Recursive, Top-downTopClassic, TopLiked, TopDPMedium413. Shortest Common Supersequence | Finding all SCSDynamic Programming, StringBottom-up, Recursive, Top-downHard414. Shortest Common Supersequence Problem using LCSDynamic Programming, StringRecursive, Bottom-up, Top-downHard415. Longest Increasing Subsequence using Dynamic ProgrammingArray, Dynamic ProgrammingAlgorithm, Amazon, Bottom-up, Recursive, Top-downTopLiked, TopDPHard416. Longest Decreasing Subsequence ProblemArray, Dynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downHard417. Longest Bitonic SubsequenceArray, Dynamic ProgrammingAlgorithm, Bottom-upMedium418. Maximum Sum Increasing Subsequence ProblemArray, Dynamic ProgrammingAlgorithm, Amazon, Bottom-up, Recursive, Top-downMedium419. The Levenshtein distance (Edit distance) ProblemDynamic Programming, StringAlgorithm, Amazon, Bottom-up, Recursive, Top-downTopClassic, TopLiked, TopDPMedium420. Find the size of the largest square submatrix of 1s present in a binary matrixDynamic Programming, MatrixAmazon, Bottom-up, Recursive, Top-downTopLikedMedium421. Matrix Chain Multiplication using Dynamic ProgrammingArray, Dynamic Programming, MatrixAlgorithm, Bottom-up, Recursive, Top-downTopClassic, TopLiked, TopDPHard422. Find minimum cost to reach the last cell of a matrix from its first cellDynamic Programming, MatrixBottom-up, Recursive, Top-downTopLikedMedium423. Find the longest sequence formed by adjacent numbers in the matrixDynamic Programming, MatrixRecursive, Top-downMedium424. Count the number of paths in a matrix with a given cost to reach the destination cellDynamic Programming, MatrixMicrosoft, Recursive, Top-downMedium425. 01 Knapsack ProblemArray, Dynamic ProgrammingAlgorithm, Amazon, Bottom-up, Recursive, Top-downTopClassic, TopLiked, TopDPMedium426. Maximize the value of an expressionArray, Dynamic ProgrammingBottom-upHard427. Partition Problem using Dynamic ProgrammingArray, Dynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downTopClassic, TopLiked, TopDPMedium428. Subset Sum Problem Dynamic Programming SolutionArray, Dynamic ProgrammingAlgorithm, Amazon, Bottom-up, Recursive, Top-downTopClassic, TopLikedMedium429. 3Partition ProblemArray, Dynamic ProgrammingAlgorithm, Hashing, Recursive, Top-downTopClassic, TopLikedMedium430. Minimum Sum Partition ProblemArray, Dynamic ProgrammingAlgorithm, Amazon, Bottom-up, Recursive, Top-downTopClassic, TopLikedHard431. Rod Cutting ProblemArray, Dynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downTopClassic, TopLiked, TopDPMedium432. Maximum Product Rod CuttingDynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downTopClassicMedium433. Coin change-making problemArray, Dynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downTopDPMedium434. Coin Change ProblemArray, Dynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downTopClassicHard435. Total possible solutions to a linear equation of k variablesDynamic ProgrammingBottom-up, Recursive, Top-downHard436. Longest Alternating Subsequence ProblemArray, Dynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downTopClassicMedium437. Longest Alternating Subsequence Problem IIArray, Dynamic ProgrammingMedium438. Count the number of times a pattern appears in a given string as a subsequenceDynamic Programming, StringBottom-up, Recursive, Top-downHard439. Collect maximum points in a matrix by satisfying given constraintsDynamic Programming, MatrixBottom-up, Recursive, Top-downHard440. Find all n-digit binary numbers without any consecutive 1sDynamic ProgrammingBottom-up, Microsoft, Recursive, Top-downEasy441. Count total possible combinations of n-digit numbers in a mobile keypadDynamic ProgrammingAmazon, Bottom-up, Recursive, Top-downMedium442. Word Break Problem Dynamic ProgrammingDynamic Programming, StringAlgorithm, Amazon, Recursive, Top-downTopClassic, TopLiked, TopDPHard443. Determine the minimal adjustment cost of an arrayDynamic ProgrammingBottom-upHard444. Check if a string is kpalindrome or notDynamic Programming, StringBottom-up, Recursive, Top-downHard445. Find total ways to achieve a given sum with n throws of dice having k facesDynamic ProgrammingRecursive, Top-downMedium446. Wildcard Pattern MatchingDynamic Programming, StringAlgorithm, Amazon, Recursive, Top-down, Bottom-upTopClassic, TopLikedHard447. Find the number of ways to fill an N 4 matrix with 1 4 tilesDynamic Programming, MatrixBottom-up, Recursive, Top-downMedium448. Ways to reach the bottom-right corner of a matrix with exactly k turns allowedDynamic Programming, MatrixRecursiveHard449. Weighted Interval Scheduling ProblemArray, Dynamic Programming, SortingAlgorithm, Bottom-up, Recursive, Top-downTopClassicMedium450. Box Stacking ProblemArray, Dynamic Programming, SortingAlgorithm, Bottom-upTopClassicHard451. Find total ways to reach nth stair with at-most m stepsDynamic ProgrammingBottom-up, Recursive, Top-downMedium452. Find total ways to reach the nth stair from the bottomDynamic ProgrammingBottom-up, Recursive, Top-downMedium453. Activity Selection Problem using Dynamic ProgrammingArray, Dynamic Programming, SortingAlgorithm, Bottom-upMedium454. Find the minimum number of deletions required to convert a string into a palindromeDynamic Programming, StringBottom-up, Recursive, Top-downTopLikedMedium455. Calculate the minimum cost to reach the destination city from the source cityDynamic Programming, MatrixBottom-upMedium456. Pots of Gold Game Problem using Dynamic ProgrammingDynamic ProgrammingAlgorithm, Amazon, Bottom-up, Recursive, Top-downTopClassicHard457. Find minimum cuts needed for the palindromic partition of a stringDynamic Programming, StringBottom-up, Recursive, Top-downHard458. Weighted Interval Scheduling Dynamic Programming SolutionArray, Dynamic Programming, SortingAlgorithm, Bottom-upMedium459. Find minimum jumps required to reach the destinationArray, Dynamic ProgrammingBottom-up, Recursive, Top-downMedium460. Find the probability that a person is alive after taking n steps on an islandDynamic Programming, MatrixHashing, Recursive, Top-downMedium461. Maximum Length Snake SequenceDynamic Programming, MatrixAlgorithmMedium462. Calculate the size of the largest plus of 1s in a binary matrixDynamic Programming, MatrixBottom-upHard463. Longest Increasing Subsequence using LCSDynamic Programming, SortingAlgorithm, Bottom-up, Recursive, Top-downMedium464. Find maximum profit earned from at most k stock transactionsArray, Dynamic ProgrammingBottom-upHard465. Count all paths in a matrix from the first cell to the last cellDynamic Programming, MatrixBottom-up, Recursive, Top-downEasy466. Check if a string matches with the given wildcard patternDynamic Programming, StringAmazon, Recursive, Top-down, Bottom-upHard467. Check if a string is interleaving of two other given stringsDynamic Programming, StringBottom-up, Recursive, Top-downMedium468. Find all employees who directly or indirectly reports to a managerDynamic ProgrammingHashing, Microsoft, RecursiveHard469. Find optimal cost to construct a binary search treeBST, Dynamic ProgrammingBottom-up, Recursive, Top-downHard470. Find the maximum sum of a subsequence with no adjacent elementsArray, Dynamic ProgrammingBottom-up, Recursive, Top-downTopLikedMedium471. Minimum-weight triangulation of a convex polygonArray, Dynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downHard472. Find maximum profit that can be earned by conditionally selling stocksArray, Dynamic ProgrammingBottom-up, Recursive, Top-downEasy473. Program to find nth Fibonacci numberBasic, Dynamic ProgrammingRecursive, Top-down, Bottom-upEasy474. Count decodings of a given sequence of digitsDynamic ProgrammingBottom-up, Recursive, Top-downMedium475. Hat Check Problem Counting DerangementsDynamic ProgrammingAlgorithm, Bottom-up, Recursive, Top-downTopClassicMedium476. Maximum Independent Set ProblemBinary Tree, Dynamic ProgrammingAlgorithm, Recursive, Top-downTopClassicMedium477. Find the minimum number of squares that sum to a given numberDynamic ProgrammingBottom-up, Recursive, Top-downMedium478. Truncate an integer array such that 2min becomes more than maxArray, Dynamic ProgrammingBottom-up, Recursive, Top-downHard479. Longest Alternating Subarray ProblemArray, Dynamic ProgrammingAlgorithm, Bottom-upTopClassicEasy480. Find maximum profit earned from at most two stock transactionsArray, Dynamic ProgrammingBottom-upHard481. Find ways to calculate a target from elements of the specified arrayArray, Backtracking, Dynamic ProgrammingRecursive, Top-downMedium482. Calculate the sum of all elements in a submatrix in constant timeDynamic Programming, MatrixBottom-upMedium483. Find maximum sum K K submatrix in a given M N matrixDynamic Programming, MatrixBottom-upHard484. Find maximum sum submatrix present in a matrixDynamic Programming, MatrixBottom-upTopClassic, TopLikedMedium485. Find the length of the longest path in a matrix with consecutive charactersDynamic Programming, MatrixDepth-First Search, RecursiveMedium486. Collect maximum value of coins in a matrixDynamic Programming, MatrixRecursiveHard487. Terminology and Representations of GraphsGraphMust KnowBeginner488. Graph Implementation C, C++, C++ STL, Java Collections, PythonGraphMust KnowBeginner489. Depth First Search (DFS)Graph, StackAlgorithm, Amazon, Depth-First Search, Must Know, RecursiveTopAlgoMedium490. Breadth-First Search (BFS)GraphAlgorithm, Amazon, Breadth-First Search, Microsoft, Must Know, RecursiveTopAlgoMedium491. Arrival and departure time of vertices in DFSGraphDepth-First Search, Must Know, RecursiveEasy492. Types of edges involved in DFS and relation between themGraphDepth-First Search, Must KnowBeginner493. Determine whether a graph is Bipartite using DFSGraphDepth-First Search, RecursiveMedium494. Topological Sort Algorithm for DAGGraphAlgorithm, Amazon, Depth-First Search, Microsoft, Must Know, RecursiveMedium495. Kahns Topological Sort AlgorithmGraphAlgorithmTopAlgoMedium496. Transitive closure of a graphGraph, MatrixAlgorithm, Breadth-First Search, Depth-First Search, RecursiveEasy497. Determine whether an undirected graph is a tree (Acyclic Connected Graph)GraphDepth-First Search, RecursiveMedium498. 2Edge Connectivity in a graphGraphAlgorithm, Depth-First Search, RecursiveHard499. 2Vertex Connectivity in a graphGraphDepth-First SearchHard500. Check if a digraph is a DAG (Directed Acyclic Graph) or notGraphDepth-First Search, RecursiveMedium501. DisjointSet Data Structure (UnionFind Algorithm)GraphAlgorithm, RecursiveTopAlgoMedium502. Check if a graph is strongly connected or notGraphBreadth-First Search, Depth-First Search, RecursiveEasy503. Check if a graph is strongly connected or not using one DFS TraversalGraphDepth-First Search, RecursiveHard504. UnionFind Algorithm for cycle detection in a graphGraphAlgorithm, Amazon, RecursiveMedium505. Single-Source Shortest Paths BellmanFord AlgorithmDynamic Programming, GraphAlgorithm, Bottom-up, Must Know, RecursiveTopAlgoMedium506. All-Pairs Shortest Paths Floyd Warshall AlgorithmDynamic Programming, Graph, MatrixAlgorithm, Bottom-up, Must Know, RecursiveTopAlgoEasy507. Find the cost of the shortest path in DAG using one pass of BellmanFordGraphDepth-First Search, RecursiveMedium508. Determine a negative-weight cycle in a graphDynamic Programming, Graph, MatrixMedium509. Find all Possible Topological Orderings of a DAGBacktracking, GraphRecursiveHard510. Find correct order of alphabets in a given dictionary of ancient originGraph, StringDepth-First Search, Hashing, RecursiveHard511. Find the longest path in a Directed Acyclic Graph (DAG)GraphDepth-First Search, RecursiveHard512. Print all kcolorable configurations of a graph (Vertex coloring of a graph)Backtracking, GraphAlgorithm, RecursiveMedium513. Print all Hamiltonian paths present in a graphBacktracking, GraphRecursiveHard514. Graph Coloring ProblemGraphAlgorithm, Greedy, HashingTopClassicMedium515. Kruskals Algorithm for finding Minimum Spanning TreeGraph, SortingAlgorithm, Amazon, Greedy, Must Know, RecursiveTopAlgoHard516. Eulerian cycle in directed graphsGraphAlgorithm, Depth-First Search, RecursiveHard517. Find root vertex of a graphGraphBreadth-First Search, Depth-First Search, RecursiveMedium518. Check whether an undirected graph is EulerianGraphAlgorithm, Depth-First Search, RecursiveMedium519. Check if a set of words can be rearranged to form a circleGraphDepth-First Search, RecursiveHard520. Find itinerary from the given list of departure and arrival airportsArray, GraphHashing, RecursiveEasy521. Introduction to Priority Queues using Binary HeapsArray, HeapPriority Queue, Must KnowBeginner522. Min Heap and Max Heap Implementation C++, JavaHeapMust KnowBeginner523. Check if an array represents a min-heap or notArray, HeapRecursiveMedium524. Convert max heap to min heap in linear timeArray, HeapRecursiveEasy525. Find kth largest element in an arrayArray, HeapAmazon, Priority QueueTopLikedMedium526. Sort a k-sorted arrayArray, HeapPriority QueueMedium527. Merge M sorted lists of variable lengthArray, Heap, SortingAmazon, Priority QueueHard528. Find kth smallest element in an arrayArray, HeapPriority QueueTopLikedMedium529. Find the smallest range with at least one element from each of the given listsArray, HeapPriority QueueHard530. Merge M sorted lists each containing N elementsArray, Heap, Matrix, SortingAmazon, Priority QueueHard531. Find first k non-repeating characters in a string in a single traversalHeap, StringAmazon, Hashing, Priority QueueMedium532. Connect n ropes with minimal costArray, HeapPriority QueueEasy533. Return kth largest element in a streamHeapPriority QueueMedium534. Huffman Coding Compression AlgorithmBit Manipulation, Binary Tree, HeapAlgorithm, Greedy, Priority QueueTopAlgoHard535. Replace each array element by its corresponding rankArray, HeapHashing, Priority QueueEasy536. Single-Source Shortest Paths Dijkstras AlgorithmGraph, HeapAlgorithm, Greedy, Must Know, Priority QueueTopAlgoMedium537. Construct a Cartesian tree from an inorder traversalBinary Tree, HeapDepth-First Search, RecursiveMedium538. Treap Data StructureBST, HeapRecursiveBeginner539. Implementation of Treap Data Structure (Insert, Search, and Delete)BST, HeapRecursiveHard540. Heap Sort AlgorithmArray, C, C++, Heap, Java, SortingAlgorithm, Must Know, Priority Queue, RecursiveTopAlgoMedium541. Introsort Algorithm Overview and C++ ImplementationArray, Divide & Conquer, Heap, SortingAlgorithm, Priority Queue, RecursiveHard542. External Merge Sort AlgorithmArray, Heap, SortingAlgorithm, Priority QueueHard543. Introduction to Linked ListsLinked ListMust KnowBeginner544. Linked List Implementation C, C++, Java, PythonLinked ListMust KnowBeginner545. Linked List Insertion at TailLinked ListMust KnowBeginner546. Static Linked ListC, Linked ListBeginner547. Clone a Linked ListLinked ListRecursiveEasy548. Delete a linked listLinked ListRecursiveEasy549. Pop operation in a linked listLinked ListEasy550. Insert a node to its correct sorted position in a sorted linked listLinked ListAmazonTopLikedEasy551. Rearrange linked list in increasing order (Sort linked list)Linked ListTopLikedMedium552. Split nodes of a linked list into the front and back halvesLinked ListEasy553. Remove duplicates from a sorted linked listLinked ListMicrosoftEasy554. Move the front node of a linked list in front of another listLinked ListEasy555. Move even nodes to the end of the linked list in reverse orderLinked ListAmazonMedium556. Split a linked list into two lists where each list contains alternating elements from itLinked ListRecursiveMedium557. Construct a linked list by merging alternate nodes of two given listsLinked ListAmazon, RecursiveEasy558. Merge two sorted linked lists into oneLinked ListAmazon, Microsoft, RecursiveTopLikedMedium559. Efficiently merge k sorted linked listsDivide & Conquer, Heap, Linked ListPriority Queue, RecursiveHard560. Intersection of two sorted linked listsLinked ListAmazon, MicrosoftMedium561. Reverse a linked List Iterative SolutionLinked ListMicrosoft, Must KnowMedium562. Reverse a Linked List Recursive SolutionLinked ListMicrosoft, Must Know, RecursiveHard563. Reverse every group of k nodes in a linked listLinked ListAmazon, Microsoft, RecursiveMedium564. Find kth node from the end of a linked listLinked ListAmazon, RecursiveEasy565. Merge alternate nodes of two linked lists into the first listLinked ListRecursiveMedium566. Merge two sorted linked lists from their endLinked List, SortingMicrosoftMedium567. Delete every N nodes in a linked list after skipping M nodesLinked ListAmazon, RecursiveEasy568. Rearrange linked list in a specific manner in linear timeLinked ListAmazon, RecursiveMedium569. Check if a linked list is palindrome or notLinked ListRecursiveTopLikedMedium570. Move the last node to the front of a linked listLinked ListRecursiveEasy571. Rearrange linked list in a specific mannerLinked ListEasy572. Floyds Cycle Detection AlgorithmLinked ListAlgorithm, Amazon, Hashing, Microsoft, Must KnowTopAlgoEasy573. Find start node of the cycle in a linked listLinked ListMedium574. Sort linked list containing 0s, 1s, and 2s in a single traversalLinked ListMicrosoftMedium575. Remove duplicates from a linked list in a single traversalLinked ListHashingEasy576. Rearrange linked list so that it has alternating high and low valuesLinked ListMedium577. Rearrange a linked list by separating odd nodes from even onesLinked ListRecursiveMedium578. Calculate height of a binary tree with leaf nodes forming a circular doubly linked listBinary Tree, Linked ListDepth-First Search, RecursiveMedium579. XOR Linked List Overview and Implementation in C/C++Bit Manipulation, C, C++, Linked ListAlgorithm, Must KnowMedium580. Recursively check if the linked list of characters is palindrome or notLinked ListRecursiveMedium581. Merge two BSTs into a doubly-linked list in sorted orderBST, Linked ListDepth-First Search, RecursiveHard582. Remove redundant nodes from a path formed by a linked listLinked ListMedium583. Add a single-digit number to a linked list representing a numberLinked ListRecursiveMedium584. Reverse every alternate group of k nodes in a linked listLinked ListRecursiveMedium585. Determine whether a linked list is palindrome or notLinked ListRecursiveMedium586. Reverse a doubly linked listLinked ListRecursiveEasy587. Pairwise swap adjacent nodes of a linked listLinked ListRecursiveMedium588. Flatten a Linked ListLinked ListRecursiveHard589. Check if a linked list of strings is palindromicLinked List, StringRecursiveEasy590. Flatten a multilevel linked listLinked ListRecursiveMedium591. Construct a height-balanced BST from an unbalanced BSTBST, Linked ListDepth-First Search, RecursiveHard592. Swap kth node from beginning with kth node from the end in a linked listLinked ListMedium593. Add two linked lists without using any extra spaceLinked ListMedium594. Remove all nodes from a linked list that matches a given keyLinked ListRecursiveEasy595. Clone a linked list with random pointerLinked ListHashing, RecursiveHard596. Update random pointer for each linked list node to point to the maximum nodeLinked ListRecursiveMedium597. Link nodes present in each level of a binary tree in the form of a linked listBinary Tree, Linked ListDepth-First Search, Hashing, RecursiveHard598. Convert a ternary tree to a doubly-linked listBinary Tree, Linked ListRecursiveMedium599. Construct a height-balanced BST from a sorted doubly linked listBST, Linked ListDepth-First Search, RecursiveHard600. In-place merge two sorted linked lists without modifying links of the first listLinked ListMedium601. Reverse specified portion of a linked listLinked ListMedium602. Find the intersection point of two linked listsLinked ListHashingMedium603. Extract leaves of a binary tree into a doubly-linked listBinary Tree, Linked ListDepth-First Search, RecursiveMedium604. Find the vertical sum of a binary treeBinary Tree, Linked ListDepth-First Search, Hashing, RecursiveHard605. In-place convert a binary tree to a doubly-linked listBinary Tree, Linked ListAmazon, Depth-First Search, Microsoft, RecursiveHard606. Find a triplet with the given sum in a BSTBST, Linked ListDepth-First Search, RecursiveHard607. Check whether the leaf traversal of given binary trees is the same or notBinary Tree, Linked List, StackDepth-First Search, RecursiveHard608. Merge sort algorithm for a singly linked listDivide & Conquer, Linked List, SortingAlgorithm, RecursiveHard609. Sort a doubly-linked list using merge sortDivide & Conquer, Linked List, SortingRecursiveMedium610. Stack Implementation using a Linked ListBasic, Linked List, StackBeginner611. Clock Angle ProblemProgramming PuzzlesAlgorithm, AmazonTopAlgoEasy612. Add two numbers without using the addition operator | 5 methodsProgramming PuzzlesEasy613. Generate the power set of a given setArray, Backtracking, Bit ManipulationAmazon, RecursiveMedium614. Implement power function without using multiplication and division operatorsProgramming PuzzlesRecursiveEasy615. Print all numbers between 1 to N without using a semicolonProgramming PuzzlesRecursiveMedium616. Swap two numbers without using a third variable | 5 methodsBit Manipulation, Programming PuzzlesEasy617. Determine the if condition to print the specific outputProgramming PuzzlesRecursiveEasy618. Find maximum and minimum value of a triplet without using a conditional statementProgramming PuzzlesMedium619. Find numbers represented as the sum of two cubes for two different pairsProgramming PuzzlesHashingMedium620. Print Hello World with empty main function | 3 methodsProgramming PuzzlesMedium621. Tower of Hanoi ProblemProgramming PuzzlesAlgorithm, RecursiveMedium622. Print all numbers between 1 to N without using any loop | 4 methodsProgramming PuzzlesRecursiveEasy623. Print a semicolon without using a semicolon anywhere in the programProgramming PuzzlesEasy624. Multiply two numbers without using a multiplication operator or loopsProgramming PuzzlesRecursiveEasy625. Find the square of a number without using the multiplication and division operatorBit Manipulation, Divide & Conquer, Programming PuzzlesEasy626. Check if a number is even or odd without using any conditional statementProgramming PuzzlesEasy627. Set both elements of a binary array to 0 in a single lineArray, Programming PuzzlesEasy628. Find minimum number without using conditional statement or ternary operatorProgramming PuzzlesRecursiveMedium629. Perform division of two numbers without using division operatorBit Manipulation, Programming PuzzlesRecursiveMedium630. Generate 0 and 1 with 75% and 25% probabilityBit Manipulation, C, Programming PuzzlesMedium631. Generate desired random numbers with equal probabilityC, Programming PuzzlesMedium632. Return 0, 1, and 2 with equal probability using a specified functionC, Programming PuzzlesMedium633. Generate numbers from 1 to 7 with equal probability using a specified functionC, Programming PuzzlesHard634. Get 0 and 1 with equal probability using a specified functionC, Programming PuzzlesMedium635. Generate random input from an array according to given probabilitiesArray, Programming PuzzlesMedium636. Generate fair results from a biased coinProgramming PuzzlesHard637. Implement ternary operator without using conditional expressionsC, Programming PuzzlesMedium638. Determine if two integers are equal without using comparison and arithmetic operatorsBit Manipulation, C, Programming PuzzlesHashingEasy639. Compute modulus division without division and modulo operatorBit Manipulation, Programming PuzzlesEasy640. Write a C/C++ program without using the main functionC, C++, Programming PuzzlesEasy641. Single line expressions to swap two integers in JavaBit Manipulation, Java, Programming PuzzlesEasy642. Find maximum number without using conditional statement or ternary operatorProgramming PuzzlesRecursiveEasy643. Find minimum or maximum of two integers without using branchingBit Manipulation, Programming PuzzlesHard644. Solve a given set of problems without using multiplication or division operatorsBit Manipulation, Programming PuzzlesMedium645. Queue implementation using an array C, C++, C++ (Using Templates), Java, PythonQueueMust KnowBeginner646. Queue Implementation using a Linked ListBasic, Linked List, QueueBeginner647. Implement a stack using the queue data structureQueue, StackRecursiveMedium648. Implement a queue using the stack data structureQueue, StackRecursiveMedium649. Efficiently print all nodes between two given levels in a binary treeBinary Tree, QueueBreadth-First Search, Depth-First Search, Hashing, RecursiveEasy650. Chess Knight Problem | Find the shortest path from source to destinationMatrix, QueueAlgorithm, Breadth-First SearchTopClassic, TopLikedHard651. Shortest path in a maze Lee AlgorithmMatrix, QueueAlgorithm, Breadth-First Search, Maze, Must KnowTopAlgoMedium652. Find the shortest safe route in a field with sensors presentMatrix, QueueBreadth-First Search, MazeHard653. Flood Fill AlgorithmMatrix, QueueAlgorithm, Breadth-First Search, Depth-First Search, Must Know, RecursiveTopAlgoMedium654. Count number of islandsMatrix, QueueAmazon, Breadth-First SearchTopLikedMedium655. Find shortest path from source to destination in a matrix that satisfies given constraintsMatrix, QueueBreadth-First Search, Maze, RecursiveTopLikedHard656. Generate binary numbers between 1 to n using a queueBit Manipulation, Queue, StringAmazonEasy657. Print nodes of a binary tree in vertical orderBinary Tree, Linked List, QueueBreadth-First Search, Depth-First Search, RecursiveMedium658. Print all nodes of a perfect binary tree in a specific orderBinary Tree, QueueBreadth-First Search, HashingHard659. Print left view of a binary treeBinary Tree, QueueAmazon, Breadth-First Search, Depth-First Search, Hashing, RecursiveTopLikedEasy660. Find the next node at the same level as the given node in a binary treeBinary Tree, QueueAmazon, Breadth-First Search, Depth-First Search, Microsoft, RecursiveMedium661. Check if a binary tree is a complete binary tree or notBinary Tree, QueueBreadth-First Search, RecursiveTopLikedMedium662. Print diagonal traversal of a binary treeBinary Tree, QueueAmazon, Breadth-First Search, Depth-First Search, Hashing, RecursiveMedium663. Print corner nodes of every level in a binary treeBinary Tree, QueueAmazon, Breadth-First SearchEasy664. Invert Binary TreeBinary Tree, Queue, StackBreadth-First Search, Depth-First Search, RecursiveEasy665. Find minimum passes required to convert all negative values in a matrixMatrix, QueueBreadth-First Search, RecursiveHard666. Convert a binary tree into a doubly-linked list in spiral orderBinary Tree, Linked List, QueueBreadth-First Search, Depth-First Search, Hashing, RecursiveHard667. Check if a binary tree is a min-heap or notBinary Tree, Heap, QueueBreadth-First Search, Depth-First Search, RecursiveMedium668. Invert alternate levels of a perfect binary treeBinary Tree, Queue, StackBreadth-First Search, Depth-First Search, RecursiveHard669. Convert a Binary Search Tree into a Min HeapBST, Heap, Linked List, QueueDepth-First Search, RecursiveHard670. Snake and Ladder ProblemGraph, QueueAlgorithm, Breadth-First SearchTopClassic, TopLikedHard671. Find the shortest distance of every cell from a landmine inside a mazeMatrix, QueueBreadth-First Search, MazeHard672. Convert a multilevel linked list to a singly linked listLinked List, QueueMedium673. Check if an undirected graph contains a cycle or notGraph, QueueAmazon, Breadth-First Search, Depth-First Search, RecursiveTopLikedMedium674. Find maximum cost path in a graph from a given source to a given destinationGraph, QueueBreadth-First SearchTopLikedMedium675. Total paths in a digraph from a given source to a destination having exactly m edgesGraph, QueueBreadth-First SearchMedium676. Least cost path in a digraph from a given source to a destination having m edgesGraph, QueueBreadth-First SearchMedium677. Traverse a given directory using BFS and DFS in JavaJava, QueueBreadth-First Search, Depth-First Search, RecursiveEasy678. Perform vertical traversal of a binary treeBinary Tree, QueueAmazon, Breadth-First Search, Depth-First Search, Hashing, RecursiveMedium679. Compute the maximum number of nodes at any level in a binary treeBinary Tree, QueueBreadth-First Search, Depth-First Search, Hashing, RecursiveEasy680. Print right view of a binary treeBinary Tree, QueueBreadth-First Search, Depth-First Search, Hashing, RecursiveMedium681. Find the minimum depth of a binary treeBinary Tree, QueueBreadth-First Search, Depth-First Search, RecursiveEasy682. Depth-First Search (DFS) vs Breadth-First Search (BFS)Binary Tree, Graph, Queue, StackAlgorithm, Breadth-First Search, Depth-First Search, Must KnowBeginner683. Bipartite GraphGraph, QueueAlgorithm, Breadth-First SearchTopLikedMedium684. Compute the least cost path in a weighted digraph using BFSGraph, QueueBreadth-First Search, RecursiveMedium685. Find the path between given vertices in a directed graphBacktracking, Graph, QueueBreadth-First Search, Depth-First Search, RecursiveEasy686. Construct a directed graph from an undirected graph that satisfies given constraintsGraph, QueueBreadth-First Search, Depth-First SearchMedium687. Trie Implementation C, C++, C++ (Memory Efficient), Java, PythonTrieBeginner688. Longest Common Prefix in a given set of strings (using Trie)String, TrieTrieMedium689. Lexicographic sorting of a given set of keysSorting, String, TrieDepth-First Search, Recursive, TrieMedium690. Lexicographic rank of a stringStringHard691. Find the maximum occurring word in a given set of stringsString, TrieDepth-First Search, Recursive, TrieEasy692. Find first k maximum occurring words in a given set of stringsHeap, String, TrieDepth-First Search, Priority Queue, Recursive, TrieMedium693. Find duplicate rows in a binary matrixMatrix, TrieAmazon, Hashing, TrieMedium694. Word Break Problem Using Trie Data StructureDynamic Programming, String, TrieAmazon, Bottom-up, Recursive, TrieMedium695. Generate a list of possible words from a character matrixBacktracking, Matrix, TrieDepth-First Search, Hashing, Recursive, TrieHard696. Find all words matching a pattern in the given dictionaryString, TrieRecursive, TrieMedium697. Find the shortest unique prefix for every word in an arrayString, TrieDepth-First Search, Recursive, TrieMedium698. Remove loop from a linked listLinked ListHashingMedium699. Find number of customers who could not get any computerStringEasy700. Find the smallest missing positive number from an unsorted arrayArrayHashingMedium701. Print all pairs of anagrams in a set of stringsSorting, String, TrieDepth-First Search, RecursiveMedium702. Find total arrangements such that no two balls of the same color are togetherDynamic ProgrammingRecursive, Top-downHard703. Determine whether a BST is skewed from its preorder traversalArray, BSTEasy704. Determine whether two nodes lie on the same path in a binary treeBinary TreeDepth-First Search, RecursiveMedium705. Find height of a binary tree represented by the parent arrayArray, Binary Tree, Dynamic ProgrammingBottom-up, Recursive, Top-downMedium706. In-place merge two height-balanced BSTsBST, Linked ListDepth-First Search, RecursiveHard707. Check if removing an edge can split a binary tree into two equal size treesBinary TreeDepth-First Search, RecursiveEasy708. Find read-write conflicts among given database transactionsArray, SortingMedium709. Construct a complete binary tree from its linked list representationBinary Tree, Linked List, QueueRecursiveEasy710. Find the minimum number of merge operations to make an array palindromeArrayMedium711. Check whether a directed graph is EulerianGraphDepth-First SearchMedium712. Count nodes in a BST that lies within a given rangeBSTDepth-First Search, RecursiveEasy713. Check if a number is a power of 8 or notBit ManipulationMedium714. Check if a number is a perfect squareDivide & Conquer, Programming PuzzlesAlgorithm, Binary SearchEasy715. Shrink an array by removing triplets that satisfy given constraintsArray, Dynamic ProgrammingRecursive, Top-downHard716. Count distinct permutations of an array that sums to a targetArray, Dynamic ProgrammingRecursiveMedium717. Check if a string can be constructed from another stringStringEasy718. Check children-sum property in a binary treeBinary TreeDepth-First Search, RecursiveEasy Thanks for reading. To share your code in the comments, please use our online compiler that supports C, C++, Java, Python, JavaScript, C#, PHP, and many more popular programming languages. Like us? Refer us to your friends and support our growth. Happy coding :) This is the article I wish I had read when I started coding. I will dive deep into 20 problem-solving techniques that you must know to excel at your next interview. They have helped me at work too and even given me ideas for a side project I am working on. Also, the last section includes astep-by-step guideexplaining how tolearn data structures and algorithms, with examples.Furthermore, I recommend you readthis post, where I outlined a high-level strategy to prepare for your next coding interview as well as the top mistakes to avoid.I have grouped these techniques in:Pointer basedRecursion basedSorting and searchingExtending basic data structuresMiscellaneaI will explain each of them, show how to apply them to coding problems, and leave you some exercises so that you can practice on your own. For your convenience, I have copied here the problem statements, but I have left links to all of the exercises. You can copy-paste my solution and play around with it. I strongly recommend you code your solution and see if it passes the tests.Some of the questions are better explained through an image or diagram. For these, I have left a comment asking you to open the link to get a graphical description of the problem.This list is part of the study notes that I took before I applied to Amazon. I hope they will be as useful to you as they have been to me.Pointer based techniques1. Two PointersThis technique is very useful onsorted arraysand arrays whose elements we want togroup.The idea is to use two (or more pointers) to split the array into different areas or groups based on some condition:Elements smaller than, equal to and greater than a certain valueElements whose sum is too small or too largeEtc.The following examples will help you understand this principle.Two sumGiven an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number. The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.Notes:Your returned answers (both index1 and index2) are not zero-based.You may assume that each input would have exactly one solution and you may not use the same element twice.Example:Input: numbers = [2,7,11,15], target = 9Output: [1,2]Explanation: The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.SolutionSince the arrayais sorted, we know that:The largest sum is equal to the sum of the last 2 elementsThe smallest sum is equal to the sum of the first 2 elementsFor any indexiin [0, a.size() 1) => a[i + 1] >= a[i]With this, we can design the following algorithm:We keep 2 pointers:l, starting at the first element of the array, andrstarting at to the last.If the sum of a[l] + a[r] is smaller than our target, we increment l by one (to change the smallest operand in the addition for another one equal or larger than it atl+1); if it is larger than the target, we decrease r by one (to change our largest operand for another one equal or smaller atr-1).We do this until a[l] + a[r] equals our target or l and r point to the same element (since we cannot use the same element twice) or have crossed, indicating there is no solution.Here is a simple C++ implementation:vector twoSum(const vector& a, int target) { int l = 0, r = a.size() - 1; vector sol; while(l < r) { const int sum = a[l] + a[r]; if(target == sum) { sol.push_back(l + 1); sol.push_back(r + 1); break; } else if (target > sum) { ++l; } else { --r; } } return sol;}The time complexity is O(N), since we may need to traverse the N elements of the array to find the solution.The space complexity is O(1), since we only need two pointers, regardless of how many elements the array contains.There are other ways of solving this problem (using a hash table, for example), but I have used it just as an illustration of the two pointer technique.ChallengesHere are two variations of this exercise:three sumandfour sum. They can be solved similarly byreducingthem to this very same problem.This is a very common technique:transform a problem whose solution you dont know to a problem that you can solve.Given a sorted array, nums, remove the duplicates in-place such that each element appears only once and return the new length.Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.Example 1:Given nums = [1,1,2],Output = 2Example 2:Given nums = [0,0,1,1,1,2,2,3,3,4],Output = 5It doesnt matter what values are set beyond the returned length.SolutionThe array issortedand we want to move duplicates to the end of the array, which sounds a lot likegrouping based on some condition. How would you solve this problem using two pointers?You will need one pointer to iterate through the array,i.And a second pointer,n, one to define the area that contains no duplicates: [0,n].The logic is as follows. If the values of the elements at indexi(excepti= 0) andi-1are:The same, we dont do anything this duplicate will be overwritten by the next unique element ina.Different: we adda[i]to the section of the array that contains no duplicates delimited byn, and increment n by one.int removeDuplicates(vector& nums) { if(nums.empty()) return 0; int n = 0; for(int i = 0; i < nums.size(); ++i){ if(i == 0 || nums[i] != nums[i - 1]){ nums[n++] = nums[i]; } } return n;}This problem has linear time complexity and constant space complexity (it is usually the case for problems solved using this technique).Sort colorsGiven an array with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.Note: You are not supposed to use the librarys sort function for this problem.Example:Input: [2,0,2,1,1,0]Output: [0,0,1,1,2,2]SolutionThe groups this time are:Smaller than 1Equal to 1Larger than 1What we can achieve with 3 pointers.This implementation is a bit tricky, so make sure you test it thoroughly.void sortColors(vector& nums) { int smaller = 0, eq = 0, larger = nums.size() - 1; while(eq next != nullptr) { slow = slow->next; fast = fast->next->next; } return slow; }Given a linked list, determine if it has a cycle in it. To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where the tail connects to. If pos is -1, then there is no cycle in the linked list.Example 1:Input: head = [3,2,0,-4], pos = 1Output: trueExplanation: There is a cycle in the linked list, where the tail connects to the second node.SolutionThe simplest solution is to add all the nodes to a hash set. When we traverse the list, if we get to a node that has already been added to the set, there is a cycle. If we get to the end of the list, there are no cycles.This has a time complexity of O(L), beingLthe length of the list, and space complexity of O(L), since in theworstcase no cycles we need to add all the elements of the list to the hash set.Time complexity cannot be improved. However, space complexity can be reduced to O(1). Think for a minute how this can be achieved with two pointers moving at different speeds.Lets call these pointers fast and slow. For each node slow visits, fast will move two nodes forward. Why?If fast reaches the end of the list, the list does not contain any cycles.If there is a cycle, since fast moves twice as fast as slow, it is just a matter of time (iterations, to be more precise) that the fast node catches the slow one, pointing both to the same node, which indicates the existence of a cycle.Now, lets translate this solution into code:bool hasCycle(ListNode *head) { ListNode* slow = head, *fast = head; while(fast){ slow = slow->next; fast = fast->next; if(!fast) break; fast = fast->next; if(slow == fast) return true; } return false;}Find the duplicate numberGiven an array, nums, containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.Example 1:Input: [1,3,4,2,2]Output: 2SolutionThis is the same problem/solution as the previous problems, for arrays instead of linked lists.int findDuplicate(const vector& nums) { int slow = nums[0], fast = slow; do { slow = nums[slow]; fast = nums[nums[fast]]; } while(slow != fast); slow = nums[0]; while(slow != fast){ slow = nums[slow]; fast = nums[fast]; } return slow;}ChallengesHere are more problems that can be solved using this technique:Detect if two linked lists have elements in commonHappy numbers3. Sliding WindowThe sliding window technique eases the task of finding optimalchunks of contiguous datathat meet a certain condition:Longest subarray that Shortest substring containing EtcYou can think of it as another variation of the two pointer technique, where pointers are updated separately based on a certain condition. Below is the basic recipe for this type of problems, in pseudocode:Create two pointers, l, and rCreate variable to keep track of the result (res) Iterate until condition A is satisfied: Based on condition B:update l, r or bothUpdate resReturn resGiven a string, find the length of the longest substring without repeating characters.Example 1:Input: abcabcbbOutput: 3Explanation: The answer is abc, with the length of 3SolutionFind the length of the longest substring without repeating characters sounds a lot likefinding optimal *chunks of contiguous datathat meet a certain condition.*Based on the recipe I described above, you will need:Two pointers,landr, to define our substrings.A variable,sol, to store the length of the longest substring we have seen so far.A way of keeping track of the characters that forms: a set,seen, will be perfect for this.While iterating through the string:If the current character is inseen*you have to increment *lto start removing elements from the beginning of ours.Otherwise, add the character toseen, moverforward and updatesol.int lengthOfLongestSubstring(const string& s) { int sol = 0; int l = 0, r = 0; unordered_set seen; while(r < s.size()) { const auto find = seen.find(s[r]); if(find == seen.end()) { sol = max (sol, r - l + 1); seen.insert(s[r]); ++r; } else { seen.erase(s[l++]); } } return sol;}ChallengesFor more practice, you can try the following problems:Permutation of a stringMax consecutive onesThere might be simpler solutions but focus on using this technique to get a better grasp of it.Recursion based techniques4. Dynamic ProgrammingI already published a long and detailed article on this topic that you can findhere.5. BacktrackingThe idea behind backtracking is to explore all the potential solutions for a problem, in a smart way. It builds candidate solutions incrementally and as soon as it determines that a candidate solution is not viable, itbacktracks to a previous state and tries the next candidate.Backtracking problems present you with a list of choices. Should you:Placethis pieceinthis position?Addthis numberto the set?Trythis numberinthis positionnext?EtcAfter you have picked one of the options, it will get you a new list of choices, until you reach a state where there are no more choices: either you arrived at a solution or there is no solution.Visually, you are moving from the root of a tree with every choice, until you get to a leaf. The basic high-level recipe (in pseudocode) for a backtracking algorithm is the following:boolean backtracking(Node n){ if(isLeaf(n) { if(isSolution(candidate)){ sol.add(candidate); return true; } else { return false; } } //Explore all children for(child in n) { if(backtracking(child)) return true; } return false;}This can of course change depending on the problem:If you needallsolutions, the helper function returns nothing (void) to avoid stopping when we find the first solution.To backtrack, you may have to bring your program to a previous state before you can move forwardAfter you choose a child, you need to detect if the candidate solution is viable or not: the definition of viable depends on the problemEtcBut the core idea is the same: examine, in a systematic way, all paths and backtrack as soon as the current path is no longer viable.N queensThe n-queens puzzle is the problem of placing n queens on an nn chessboard such that no two queens attack each other Given an integer n, return all distinct solutions to the n-queens puzzle.Each solution contains a distinct board configuration of the n-queens placement, where Q and . both indicate a queen and an empty space respectively.Example:Input: 4Output: [[.Q.., Q, Q, ..Q.], [..Q., Q, Q, .Q..]]Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above.SolutionThis is a classic backtracking problem:We need all solutions here, which is why the recursive function returns nothing as I explained in the introduction of this section.Do not worry too much about theisViableSolutionfunction for now. Try to see the recipe I gave (slightly modified) you in action.class Solution {public: vector solveNQueens(int n) { vector solutions; /** This is usually solved with a vector of integers, where each integer represents the position of the queen in that column. This particular problem expects strings. Each string represents a column */ vector board(n, string(n, '.')); solveBoard(solutions, board, 0, n); return solutions; } void solveBoard(vector& solutions, vector& board, int col, int n){ if(col == n){ solutions.push_back(board); return; } for(int row = 0; row < n; row++){ if(isViableSolution(board, row, col)){ board[row][col] = 'Q'; solveBoard(solutions, board, col + 1, n); //Backtracking - we bring our board to the previous state board[row][col] = '.'; } } } bool isViableSolution(vector& board, int row, int col){ int n = board.size(); for(int x = 1; x = 0 && col >= x; x++){ if(board[row-x][col-x] == 'Q') return false; } for(int x = 1; row + x < n && col >= x; x++){ if(board[row+x][col-x] == 'Q') return false; } return true; }};Letter combinationGiven a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent (check the link for diagram). Note that 1 does not map to any letters.Example:Input: 23Output: [ad, ae, af, bd, be, bf, cd, ce, cf].SolutionFor every number in the input, you have several letters to choose from. If you can draw a tree (this is what I do) where the branches are born from the different choices you take, chances are that you can apply backtracking.Note: Before you start solving any problem, try different approaches: dynamic programming, greedy algorithms, divide and conquer, a combination of algorithms and data structures, etc. Coding is thelast step.My solution, in C++:vector letterCombinations(const string &digits) { if(digits.empty()) return {}; const vector letters {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"}; vector sol; string candidate (digits.size(), ' '); h(sol, 0, candidate, letters, digits); return sol;} void h(vector &sol, int idx, string &candidate, const vector &letters, const string &digits){ if(idx == digits.size()){ sol.push_back(candidate); return; } for(const char &c : letters[digits[idx] - '0']) { candidate[idx] = c; h(sol, idx + 1, candidate, letters, digits); }}Since I knew already the size of the solution, I initialize mycandidatewith that size and just modified the character at positionidx. If the size is not known, this can be done instead:string candidate; //instead of string candidate (digits.size(), ' ');for(const char &c : letters[digits[idx] - '0']) { candidate.push_back(c); h(sol, idx + 1, candidate, letters, digits); candidate.pop_back();} Sudoku solverWrite a program to solve a Sudoku puzzle by filling the empty cells. Open the link to get a longer description, including an image of the puzzle.SolutionIn an interview, unless you have plenty of time, you will not need to implementisViableSolution, just to sketch it. I know a colleague who got this question in an on-site.Even though the code is long, it is mostly because ofisViableSolution. Otherwise, it is not very different from other backtracking problems.void solveSudoku(vector& board){ helper(board);} bool helper(vector& board, int row = 0, int col = 0) { if(col == size){ col = 0; ++row; if(row == size){ return true; } } if(board[row][col] != '.'){ return helper(board, row, col + 1); } for(char i = '1'; i nums2[j]) { ++j; } else { ++i; } } return sol;}The time complexity of this approach is O(n log n) even though the two-pointer part is linear and the space complexity is O(1) not including the space needed to store the intersection, of course, in which case we could say it is O(min(length(A), length(B)).Challenges8. IntervalsMost interval related problems I have seen boil down to:Modeling the intervalas an array of two elements, a pair/tuple or a custom class (this is the cleanest option)Sortingthe inputIterating through the sorted array and deciding what to do based on the starts/ends of the intervalsYou can see this as yet another type of problem that can be simplified after sorting the input, which is why I have included it in this section.Im leaving here my solution to two exercises, based on what I have just described. Try them before reading my solutions.Merge intervalsGiven a collection of intervals, merge all overlapping intervals.Example 1:Input: intervals = [[1,3],[2,6],[8,10],[15,18]]Output: [[1,6],[8,10],[15,18]]Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].Solutionvector merge(vector& intervals) { sort(intervals.begin(), intervals.end(), [](const auto& i1, const auto& i2){ return i1[0] < i2[0]; }); int i = 0; vector sol; vector curr(2); while(i < intervals.size()){ curr = intervals[i++]; while(i < intervals.size() && intervals[i][0] x >= 4, which is true for any integer in the range [2,]With this, we can search in [2, x/2] and speed things up a bit.int mySqrt(int x) { if(x == 0 || x == 1) return x; int sol = 1; int l = 2, r = x / 2; while(l x){ r = m - 1; } else { sol = m; l = m + 1; } } return sol;}ChallengesHave fun!10.Breadth-First SearchThis is one of the techniques you need to know to explore trees and graphs. Since many problems can be modeled as graphs, youmust know this technique. To implement it, we just need to use a queueqand add to this same queue the children of the nodes we process fromq.At any given point in the iteration, BFS visits all the nodes at the same distance from the origin. This will become clearer after some of these examples.Word ladderGiven two words (beginWord and endWord), and a dictionarys word list, find the length of the shortest transformation sequence from beginWord to endWord, such that:Only one letter can be changed at a time.Each transformed word must exist in the word list.Notes:Return 0 if there is no such transformation sequence.All words have the same length.All words contain only lowercase alphabetic characters.You may assume no duplicates in the word list.You may assume beginWord and endWord are non-empty and are not the same.Example 1:Input:beginWord = hit,endWord = cog,wordList = [hot,dot,dog,lot,log,cog]Output: 5Explanation: As one shortest transformation is hit -> hot -> dot -> dog -> cog, return its length 5.SolutionI got asked this question during my on-site interview at Amazon. The idea is to model this problem using a graph:Nodes represent wordsEdges connect words that only differ by one letterWith this setup, this problem is equivalent to finding a path between two nodes in a graph, which BFS can solve. Since all edges have the same weight (1), we do not need Dijkstra or any other fancier algorithm.int ladderLength(const string &beginWord, const string &endWord, const vector& wordList) { if(beginWord == endWord) return 1; unordered_set dict(wordList.begin(), wordList.end()); queue todo; todo.push(beginWord); dict.erase(beginWord); int ladder = 1; while (!todo.empty()) { ladder++; int n = todo.size(); for (int i = 0; i < n; i++) { string word = todo.front(); todo.pop(); for (int j = 0; j < word.size(); j++) { char c = word[j]; for (int k = 0; k < 26; k++) { word[j] = 'a' + k; if (dict.find(word) != dict.end()) { if (word == endWord) { return ladder; } todo.push(word); dict.erase(word); } } word[j] = c; } } } return 0;}After you visit the DFS section:What would happen if we use DFS instead? Do you see any benefits/drawbacks?Order level tree traversalGiven a binary tree, return the level order traversal of its nodes values. (ie, from left to right, level by level).Open the link for a graphical description of the problem.SolutionYou only need to add here a little tweak to the standard BFS algorithm: you need to know how many elements in the queue you need to process for each level.This is one approach that can be applied to many other problems.vector levelOrder(TreeNode* root) { if(!root) return {}; vector sol; queue q; q.push(root); vector partial; while(!q.empty()){ int size = q.size(); while(size-->0){ auto n = q.front(); partial.push_back({n->val}); q.pop(); if(n->left) q.push({n->left}); if(n->right) q.push({n->right}); } sol.push_back(partial); partial.clear(); } return sol;}ChallengesLet me propose a different kind of challenge: building something, instead of solving abstract problems. I find it more fun and you can add them to your Github profile. Here are just two examples:Web crawlerusing BFS to explore all the links on a website.Minesweeper11.Depth-First SearchSimilar to BFS in its purpose: explore trees and graphs. DFS is not guaranteed to find the shortest path between two points, but it will find any existing path.It is usually shorter to implement than BFS. Some people find it easier. Others, because of the recursive calls, not so much. It is up to you. Just make sure you think about potential stack overflow issues if the size of the stack starts to get big.Some problems are much easier to be solved with DFS/recursion, that it is worth practicing.Number of islandsGiven a 2d grid map of 1s (land) and 0s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.Example:Input: grid = [[1,1,1,1,0], [1,1,0,1,0], [1,1,0,0,0], [0,0,0,0,0]]Output: 1SolutionI got this problem at my first phone interview at Amazon.As soon as you see a matrix, think of a graph. This problem (and its variations) is very straightforward:Iterate through the matrixFor every 1 you find, increase the counter and sink the islandTo sink the island, you need to visit all the surrounding 1s in the matrix, which is equivalent tovisit all the neighbors of that node and of all its neighbors, which sounds a lot like a recursive problem.You can try to solve this using BFS too, but DFS is much shorter.int numIslands(vector& grid) { int numIslands = 0; for(int r = 0; r < grid.size(); ++r){ for(int c = 0; c < grid[0].size(); ++c){ if(grid[r][c] == '1'){ ++numIslands; sinkIslands(grid, r, c); } } } return numIslands;} const vector dirs {{1, 0}, {-1, 0}, {0, 1}, {0, -1}}; void sinkIslands(vector &m, int r, int c){ m[r][c] = '0'; for(const auto &d : dirs){ const int nr = r + d[0]; const int nc = c + d[1]; if(isValid(m, nr, nc)){ sinkIslands(m, nr, nc); } }} bool isValid(vector &m, int r, int c){ return r >= 0 && r < m.size() && c >= 0 && c < m[0].size() && m[r][c] == '1';}Symmetric treeGiven a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).SolutionMany tree-related problems have relatively straightforward recursive solutions. This problem could be solved using BFS but DFS makes it so much easier.I will leave this one here as an exercise for you. Just use my solution in case you get stuck.bool isSymmetric(TreeNode* root) { if(!root) return true; return helper(root->left, root->right);} bool helper(TreeNode* p, TreeNode* q){ if(!p && !q) return true; if(!p || !q) return false; return p->val == q->val && iS(p->left, q->right) && iS(p->right, q->left);}ChallengesTake the same challenges and exercises I gave you for BFS and try to implement them using DFS instead. Also, for more practice, give a try to the following exercises:Path sumPath sum 2Validate BST12.Topological sortYou can see this algorithm as an application of DFS. Its implementation just needs one minor change to the regular DFS: after processing all the children of a node, add this node to a stack.It is that simple.The best way to intuitively understand what this algorithm achieves is to imagine a bunch of tasks, some of which depend on others (task 1 cannot start until task 2 has finished). A topological sort will list all these tasks preserving this structure of dependencies.Lets solve a problem using this algorithm.Course schedule IIThere are a total of n courses you have to take, labeled from 0 to n-1. Some courses may have prerequisites, for example, to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses. There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.Example 1:Input: 2, [[1,0]]Output: [0,1]Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1].Example 2:Input: 4, [[1,0],[2,0],[3,1],[3,2]]Output: [0,1,2,3] or [0,2,1,3]Explanation: There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3].SolutionThis is the classical topological sort problem. We have a bunch of courses to take and some depend on others. This can be modeled as a directed graph. A topological sort would returnthe ordering of courses you should take to finish all courses. A prerequisite to applying topological sort is that the graph isdirected and acyclic. From the problem description, we can see that the graph is directed. We can detect if it contains any cyclesandcompute a topological sort in the same pass. A more indirect but still valid approach would be to first check whether it has cycles and only if there are no cycles, compute a topological sort.vector findOrder(int numCourses, vector& prerequisites) { vector adj(numCourses, vector(0)); for(const auto &p : prerequisites) { int u = p[0]; int v = p[1]; adj[u].push_back(v); } vector white(numCourses, true), grey(numCourses), black(numCourses); vector sol (0); for(int i = 0; i < numCourses; ++i) { if(white[i] && hasCycle(adj, i, white, grey, black, sol)) { return {}; } } return sol;} bool hasCycle(const vector& adj, int i, vector &white, vector &grey, vector &black, vector& sol) { //We have started exploring this node white[i] = false; grey[i] = true; for(const auto & n : adj[i]){ if(black[n]) continue; if(grey[n] || hasCycle(adj, n, white, grey, black, sol)) return true; } grey[i] = false; black[i] = true; sol.push_back(i); return false;}ChallengesHereyou can find a few more problems to practice.Have fun!Extending basic data structures13. Dequeuest have seen data structure mostly used as another way to implement the sliding window technique you read about earlier in this article. The only difference with a standard FIFO queue is that you can operate (insert and delete elements) onboth endsof the queue.Thats it. Simple.Lets see how such a minor change can simplify some kind of these problems.Sliding window maximumGiven an array, nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window.Note: Open the link for a better understanding of the problem (there is an image).Example:Input: nums = [1,3,-1,-3,5,3,6,7], and k = 3Output: [3,3,5,5,6,7]SolutionWe will use the dequeue to storeindices, not values. We need this to know what elements are still part of the sliding window. For every iteration, there are four things to do.Remove elements in the dequeue which are outside of the current sliding window (one per iteration)Remove allelements in the dequeue which are smaller than the current element we are at since they cannot represent the max of that sliding windowAdd the current element to the dequeueOnce we have completed the first sliding window, we can start adding elements to our solution. By design, the element at the front of our dequeue will contain the maximum of the sliding window, which is what we are interested in.This technique can be applied to find the minimum or other properties of a contiguous block of data in an array.vector maxSlidingWindow(const vector& nums, int k) { vector sol; deque dq; for (int i=0; i= k-1) sol.push_back(nums[dq.front()]); } return sol;}ChallengeDesign your owncircular dequeue, to fully grasp the internals of this data structure.14.TrieThe best way to think of a trie is as an extension of a tree, where you store characters that form words as you move through the different branches of the trie.There are variants where you store suffixes instead of prefixes, where you use compression to reduce the size of the trie, etc. But at its basic, it is another type of tree.They are used everywhere:Auto-completeSpell checkersIP routingLongest prefix/suffix matchingEtcImplement a trieImplement a trie with insert, search, and startsWith methods.SolutionHere is a simple implementation (for an interview, of course) of a trie. Compared to a tree, you just:An extra boolean to indicate whether that node marks the end of a wordA data structure to store pointers to the nodes children: a hash table, an array of characters, etc.class Trie{ private: struct Node{ bool isWord; unordered_map children; Node() : isWord(false) {} }; Node* findNode(const string &word){ Node* curr = root; for(int i = 0; i < word.size() && curr; ++i) curr = curr->children[word[i] - 'a']; return curr; } public: Node* root; /** Initialize your data structure here. */ Trie() { root = new Node(); } /** Inserts a word into the trie. */ void insert(const string &word) { Node * curr = root; for(int i = 0; i < word.size(); ++i){ const char c = word[i] - 'a'; if(!curr->children[c]){ Node* newNode = new Node(); curr->children[c] = newNode; } curr = curr->children[c]; } curr->isWord = true; } /** Returns true if the word is in the trie. */ bool search(const string &word) { Node * curr = findNode(word); return curr ? curr->isWord : false; } /** Returns true if there is any word in the trie that starts with the given prefix. */ bool startsWith(const string &prefix) { Node * curr = findNode(prefix); return curr ? true : false; } };Word search IIGiven a 2D board and a list of words from the dictionary, find all words in the board. Each word must be constructed from letters of sequentially adjacent cells, where adjacent cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.Example:Input: board = [[o,a,a,n], [e,t,a,e], [i,h,k,r], [i,f,l,v]] words = [oath,pea,eat,rain]Output: [eat,oath]SolutionThis might not be the simplest way of solving this problem, but it is a clear application of a trie.At every step, you will have built some candidate string and need to check if it belongs to the dictionary. A hash set containing all the words in the dictionary would give a very good performance. Why bothering with a trie then? Because a trie can tell you whether that path is worth exploring or not, improving the efficiency of our solution.In our previous example, imagine we form the string oa.We can check if this prefix (potential word) exists in our trie. It exists since we have added the word oath. Now imagine we keep moving right through our board and form the string oaa. In our trie, there are no words that contain the prefix oaa, so we can backtrackat this point.With a hash set that contains all the words, you could not do this type of prefix matching, unless you create two different tables: one for prefixes and another one for words.This is a complex problem because it combines different elements and it is not easy to implement, so do not get discouraged if it takes you a few tries (no pun intended) to get it right.const vector dirs {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};struct Trie{ struct Node{ bool isWord; unordered_map children; Node() : isWord(false){} }; Node* root; Trie(){root = new Node();} void insert(const string& w){ Node* curr = root; for(const auto &c : w){ if(curr->children.find(c) == curr->children.end()){curr->children[c] = new Node();} curr = curr->children[c]; } curr->isWord = true; } bool hasPrefix(const string &prefix){ Node* curr = root; for(const auto &c : prefix) { if(curr->children.find(c) == curr->children.end()) return false; curr = curr->children[c]; } lastNode = curr; return true; } bool isValidWord(const string& w){ bool res = lastNode->isWord; lastNode->isWord = false; return res; } void deleteWord(){ lastNode->isWord = false; lastNode = nullptr; } Node* lastNode;}; Trie t;public:vector findWords(vector& board, vector& words) { for(const auto& w : words) t.insert(w); vector sol; for(int row = 0; row < board.size(); ++row){ for(int col = 0; col < board[0].size(); ++col){ helper(board, row, col, sol, candidate); } } return sol; } void addWords(vector &board, int row, int col, vector& sol, string &candidate){ if(t.isValidWord(candidate)){ sol.push_back(candidate); t.deleteWord(); } const char old = board[row][col]; board[row][col] = '-'; for(const auto &d : dirs){ const int nrow = row + d[0]; const int ncol = col + d[1]; if(nrow >= 0 && nrow < board.size() && ncol >= 0 && ncol < board[0].size() && board[nrow][ncol] != '-' && t.hasPrefix(candidate + board[nrow][ncol])){ candidate.push_back(board[nrow][ncol]);addWords(board, nrow, ncol, sol, candidate);candidate.pop_back(); } } board[row][col] = old; }};ChallengesDesign your ownautocomplete system!15. Two instances of the same data structureSome problems can be solved by using two different instances of the same data structure, so it is worth keeping it in mind when you get stuck in a problem.I will group most of them in this section. Median of Data StreamMedian is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle values.For example,[2,3,4], the median is 3[2,3], the median is (2 + 3) / 2 = 2.5Design a data structure that supports the following two operations:void addNum(int num) Add an integer number from the data stream to the data structure.double findMedian() Return the median of all elements so far.Solutionclass MedianFinder { priority_queue L; priority_queue H; public: MedianFinder() { } void addNum(int x) { if(L.empty() == false && x > L.top()) { H.emplace(x); } else { L.emplace(x); } if(H.size() > L.size() + 1) { L.emplace(H.top()); H.pop(); } else if(L.size() > H.size() + 1) { H.emplace(L.top()); L.pop(); } } double findMedian() { if(H.size() == L.size()) { return (H.top() + L.top()) * 0.5; } else { return H.size() > L.size() ? H.top() : L.top(); } }};ChallengesIn increasing order of difficulty:Miscellanea16. Bit manipulationThis section deserves a separate article. Here I will list some basictricks and common bit manipulation problems.This isthe most comprehensive siteI have found on this topic. Use it as a reference.Missing numberGiven an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.Example 1:SolutionThis problem can be solved just by using the XOR operator:Any bit XORed with itself produces a 0 -> a ^ a = 0Any bit XORed with 0 produces the original bit -> a ^ 0 = aXOR is associative = a ^ (b ^ c) = (a ^ b) ^ cIf we XOR all the numbers in the array (integers in the range [0,n]) with all the integers in [0 to n], the pairs will producezeroesand the missing number will be XORed with 0 (resulting in itself), solving our problem.int missingNumber(const vector& nums) { int res = nums.size(); for(int i = 0; i < nums.size(); ++i) res ^= (i ^ nums[i]); return res; }Power of twoGiven an integer, write a function to determine if it is a power of two.SolutionI got this one in an interview.Powers of two can be expressed in binary as a leading 1 and a set of 0s:With this, it is simple to figure out if a number is a power of two or not. You can achieve fast if you know what the following line does (I am sure you can figure it out on your own):This trick is worth knowing since it is used a lot.bool isPowerOfTwo(int n) { return n > 0 ? (n & (n - 1)) == 0 : false; }Number of 1sWrite a function that takes an unsigned integer and returns the number of 1 bits it has (also known as the Hamming weight).Example 1:Input: 00000000000000000000000000001011Output: 3Explanation: The input binary string 00000000000000000000000000001011 has a total of three 1 bits.SolutionThis problem is very straightforward: iterate through all the bits in the input and count how many of them are 1s.Try to use the trick I showed you in the previous exercise to improve the performance of this algorithm (in the average case).int hammingWeight(uint32_t n) { int sum = 0; while (n != 0) { sum++; n &= (n - 1); } return sum; }ChallengesThese are for you to practice the previous techniques:Single numberGray codeHamming distancePower of 417. Top K ElementsThis is another very frequent type of problem. You are given a list of elements and have to return the top K, definingtopas:The largest/smallestThe closest/furthest to a pointThe most frequent in the listEtcI have seen some of the following (or some sort of variation) asked in interviews.There is no single data structure that will always give the right solution, but the following elements are very useful:Hash tablePriority queueSorting (the input or as an intermediate step)Priority queues usually provide a better complexity.Top k frequent wordsGiven a non-empty list of words, return the k most frequent elements. Your answer should be sorted by frequency from highest to lowest. If two words have the same frequency, then the word with the lower alphabetical order comes first.Example 1:Input: [i, love, leetcode, i, love, coding], k = 2Output: [i, love]Explanation: i and love are the two most frequent words. Note that i comes before love due to a lower alphabetical order.SolutionPretty straightforward: count how many times each word appears (using a hash table) andsomehowreturn the k most common elements.For this last part, you either:Put all the elements with its frequencies in an array and sort itOr use a priority queueIt is worth knowing this second approach since it can be applied to other problems.Here is a simple solution in C++ using a priority queue.vector topKFrequent(const vector& words, int k) { map freq; for(const auto &w : words) freq[w]++; auto l = [](const pair &p1, const pair &p2){ if(p1.first == p2.first) return p1.second < p2.second; else return p1.first > p2.first; }; priority_queue pq(l); for(auto it = freq.begin(); it != freq.end(); ++it){ if(pq.size() < k){ pq.push({it->second, it->first}); } else if(l({it->second, it->first}, pq.top())){ pq.pop(); pq.push({it->second, it->first}); } } vector sol (k); while(!pq.empty()){ sol[--k] = pq.top().second; pq.pop(); } return sol; }VariationK closest points to the originWe have a list of points on the plane. Find the K closest points to the origin (0, 0). Here, the distance between two points on a plane is the Euclidean distance.You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)Example 1:Input: points = [[1,3],[-2,2]], K = 1Output: [[-2,2]]SolutionThe obvious solution is to compute the distance to every single point, add them to an array, sort and get the top k. This is fine, but it can be improved with a priority queue.We use amax heap, where the root of the heap is the furthest point. For every point, we compute the distance from the origin and the root points to the point that is the furthest from the origin amongst all the other points. If we find a point closer to this one, it may still be further than the rest, but we need to drop our current top and add this new point to the heap.typedef pair pivi;public:vector kClosest(const vector& points, int K) { auto l = [] (const pivi& p1, const pivi &p2) { return p1.first < p2.first; }; priority_queue pq(l); for(const auto &p : points){ //Avoid taking the square root. Won't change the result and it's faster const int d = p[0] * p[0] + p[1] * p[1]; if(pq.size() < K) pq.push({d, {p[0], p[1]}}); else if(pq.top().first > d){ pq.pop(); pq.push({d, {p[0], p[1]}}); } } vector sol; while(!pq.empty()){ sol.push_back({pq.top().second.first, pq.top().second.second}); pq.pop(); } return sol; }18. K-way mergeMerge two sorted listsMerge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.Example:Input: 1->2->4, 1->3->4Output: 1->1->2->3->4->4SolutionWe can solve this problem with the same Two pointer technique we saw at the beginning of this article. In this case, we traverse linked lists instead of arrays, but the same ideas apply.ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) { ListNode dh(1), *curr = &dh; while(l1 && l2){ if(l1->val val) { curr->next = l1; l1 = l1->next; } else { curr->next = l2; l2 = l2->next; } curr = curr->next; } if(l1){ curr->next = l1; } else if(l2) { curr->next = l2; } return dh.next; }Merge k sorted listsGiven an array of linked-lists lists, each linked list is sorted in ascending order.Merge all the linked-lists into one sort linked-list and return it.Example 1:Input: lists = [[1,4,5],[1,3,4],[2,6]]Output: [1,1,2,3,4,4,5,6]Explanation: The linked-lists are:[1->4->5, 1->3->4, 2->6]merging them into one sorted list:1->1->2->3->4->4->5->6SolutionThis is the general version of the previous problem. As I said, you can reduce this problem to many instances of the previous problem by merging lists 2 at a time. However, here I will present a more efficient solution.Now, instead of two sorted lists, we have k. We need to create a list from picking the next minimum element of all the lists. Since they are sorted, it will be the first element of one of these lists. Luckily, there is a data structure that returns its minimum element in O(1) and has an insertion complexity of O(log K): a priority queue.For every list, we add to the priority queue a pair containing:The head of that listAn index to remember to which list that element belongsAfter we pop an element, we add its value to our solution and add to the priority queue the new head of that list (the pair stores the value and the index of the list).With this information, try to solve the problem. You will learn much more from it than from reading my solution without trying first.ListNode* mergeKLists(vector& lists) { typedef pair pni; auto comp = [](const pni& p1, const pni& p2) { return p1.first->val > p2.first->val; }; priority_queue pq (comp); for(int i = 0; i < lists.size(); ++i){ if(lists[i]){ pq.push({lists[i], i}); lists[i] = lists[i]->next; } } ListNode dh(-1), *curr = &dh; while(!pq.empty()){ const auto p = pq.top(); pq.pop(); curr->next = p.first; curr = curr->next; ListNode* next = lists[p.second]; if(next){ int idx = p.second; pq.push({next, idx}); lists[p.second] = lists[p.second]->next; } } return dh.next; }19.Rolling hashRabin Karpis a great example of how to design a simple and efficient algorithm using intelligently a rolling hash function. It can be used to find a stringsin a textt.The basic ideas that you need to remember to be able to reconstruct this algorithm are the following:This is an improvement over the brute force method, where you comparesand every candidate substring,c, which is inefficient.If two strings are the same, they will produce the same hash.The inverse is not true: two different strings may produce the same hash.Using a rolling hash function we can compute the hash of a new string in O(1)If we put all this together, we will efficiently compute hashes and only comparecandswhen they have the same hash, reducing the number of comparisons and thus the average complexity of our algorithm (worst case, we need to compare all substrings and are back to the brute force method).Find string in textReturn the index of the first occurrence of needle in a haystack, or -1 if needle is not part of the haystack.Example 1:Input: haystack = hello, needle = llOutput: 2Example 2:Input: haystack = aaaaa, needle = bbaOutput: -1SolutionBased on the above paragraphs, try to code the Rabin-Karp algorithm on your own to solve this problem.int strStr(const string& haystack, const string& needle) { int nsize = needle.size(); if(nsize == 0) return 0; int hsize = haystack.size(); if(nsize > hsize) return -1; int msbPower = 1; int nhash = 0; int hhash = 0; const int mod = 10000009; //A big prime number for(int i = 0; i lps[j - 1] if p[i] == p[j]: j += 1 lps[i] = j return lps lps = build_lps(pattern) i = j = 0 while i < len(text): if text[i] == pattern[j]: i += 1 j += 1 if j == len(pattern): return i - j else: if j != 0: j = lps[j-1] else: i += 1 return -1 Time Complexity: O(n + m) 19. Floyd's Cycle Detection When to Use: Detect cycles in linked lists. Example: Fast/slow pointers. def has_cycle(head): slow = fast = head while fast and fast.next: slow = slow.next fast = fast.next.next if slow == fast: return True return False Time Complexity: O(n) 20. Meet-in-the-Middle When to Use: Split large problems into halves (e.g., subset sum). Example: Subset sum for large `n`. def subset_sum(arr, target): def generate_subsets(arr): subsets = [] n = len(arr) for mask in range(1