Algorithms Questions

When does the worst case of QuickSort occur?

In quickSort, we select a pivot element, then partition the given array around the pivot element by placing pivot element at its correct position in the sorted array. The worst case of quickSort occurs when one part after partition contains all elements and other part is empty. For example, if the input array is sorted and if the last or first element is chosen as a pivot, then the worst occurs.

What Is The Difference Between A Backtracking Algorithm And A Brute-force One?

Due to the fact that a backtracking algorithm takes all the possible outcomes for a decision, it is similar from this point of view with the brute force algorithm. The difference consists in

What are the three laws of recursion algorithm?

All recursive algorithm must follow three laws:

It should have a base case

A recursive algorithm must call itself

A recursive algorithm must change its state and move towards the base case.

What is a binary search tree?

Binary Search Tree has some special properties e.g. left nodes contains items whose value is less than root , right sub tree contains keys with higher node value than root, and there should not be any duplicates in the tree.

How does quicksort work?

and all elements with values greater than the pivot come after it (equal values can go either way). This is also known as partitioning. After partitioning the pivot is in its final position.

3) Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values. If the array contains only one element or zero elements then the array is sorted.

How do Insertion sort, Selection sort, Heapsort, Quicksort, and Merge sort work?

Insertion sort takes elements of the array sequentially and maintains a sorted subarray to the left of the current point. It does this by taking an element, finding its correct position in the sorted array, and shifting all following elements by 1, leaving a space for the element to be inserted.

Selection sort is a combination of searching and sorting. During each pass, the unsorted element with the smallest (or largest) value is moved to its proper position in the array. The number of times the sort passes through the array is one less than the number of items in the array.

Heapsort starts by building a max heap. A binary max heap is a nearly complete binary tree in which each parent node is larger or equal to its children. The heap is stored in the same memory in which the original array elements are. Once the heap is formed, it completely

element being in the first place, and the following elements being sequentially larger.

Quicksort is performed by taking the first (leftmost) element of the array as a pivot point. We then compare it to each following element. When we find one that is smaller, we move it to the left. The moving is performed quickly by swapping that element with the first element after the pivot point and then swapping the pivot point with the element after it. After going through the whole array, we take all points on the left of the pivot and call quicksort on that subarray, and we do the same to all points on the right of the pivot. The recursion is performed until we reach subarrays of 0-1 elements in length.

Merge sort recursively halves the given array. Once the subarrays reach trivial length, merging begins. Merging takes the smallest element between two adjacent subarrays and repeats that step until all elements are taken, resulting in a sorted subarray. The process is repeated on pairs of adjacent subarrays until we arrive at the starting array, but sorted.

What are the key advantages of Insertion Sort, Quicksort, Heapsort, and Mergesort? Discuss best, average, and worst-case time and memory complexity?

Insertion sort has an average and worst runtime of O(n^2), and the best runtime of O(n). It doesn't need any extra buffer, so space complexity is O(1). It is efficient at sorting extremely short arrays due to a very low constant factor in its complexity. It is also extremely good at

Heapsort and Mergesort maintain that complexity even in worst case scenarios, while Quicksort has the worst-case performance of O(n^2).

Quicksort is sensitive to the data provided. Without usage of random pivots, it uses O(n^2) time for sorting a full sorted array. But by swapping random unsorted elements with the first element, and sorting afterward, the algorithm becomes less sensitive to data would otherwise cause worst-case behavior (e.g. already sorted arrays). Even though it doesn't have lower complexity than Heapsort or Merge sort, it has a very low constant factor to its execution speed, which generally gives it a speed advantage when working with lots of random data.

Heapsort has reliable time complexity and doesn't require any extra buffer space. As a result, it is useful in software that requires reliable speed over optimal average runtime, and/or has limited memory to operate with the data. Thus, systems with real-time requirements and memory constraints benefit the most from this algorithm.

Merge sort has a much smaller constant factor than Heapsort but requires O(n) buffer space to store intermediate data, which is very expensive. Its main selling point is that it is stable, as compared to Heapsort which isn't. In addition, its implementation is very parallelizable.

What are Divide and Conquer algorithms? Describe how they work. Can you give any common examples of the types of problems where this approach might be used?

independently. Once we've solved all of the pieces, we take all of the resulting smaller solutions and combine them into a single integrated comprehensive solution.

This process can be performed recursively; that is, each "subproblem" can itself be subdivided into even smaller parts if necessary. This recursive division of the problem is performed until each individual problem is small enough to become relatively trivial to solve.

Some common examples of problems that lend themselves well to this approach are binary search, sorting algorithms (e.g., Merge Sort, Quicksort), optimization of computationally complex mathematical operations (Exponentiation, FFT, Strassen's algorithm), and others.

Why do you think quicksort is better than merge sort?

Quicksort has O(n2) worst-case runtime and O(n log n) average case runtime while the worst and best case for merge sort is O(n log n)
However, it's superior to merge sort in many scenarios because many factors influence an algorithm's runtime, and, when taking them all together, quicksort wins out.

In particular, the often-quoted runtime of sorting algorithms refers to the number of comparisons or the number of swaps necessary to perform to sort the data. This is indeed a good measure of performance, especially since it's independent of the underlying

locality, and this makes it faster than merge sort in many cases.

In addition, it's very easy to avoid quicksort's worst-case run time of O(n2) almost entirely by using an appropriate choice of the pivot – such as picking it at random (this is an excellent strategy).

What is A, what are its implementation details, and what are its advantages and drawbacks in regard to traversing graphs towards a target?*

A* is an algorithm for pathfinding that doesn't attempt to find optimal solutions, but only tries to find solutions quickly and without wandering too much into unimportant parts of the graph.

It does this by employing a heuristic that approximates the distance of a node from the goal node. This is most trivially explained on a graph that represents a path mesh in space. If our goal is to find a path from point A to point B, we could set the heuristic to be the Euclidean distance from the queried point to point B, scaled by a chosen factor.

This heuristic is employed by adding it to our distance from the start point. Beyond that, the rest of the implementation is identical to Dijkstra.

The algorithm's main advantage is the quick average runtime. The main disadvantage is the fact that it doesn't find optimal solutions, but any solution that fits the heuristic.

There can be many solutions for this. The best solution is to use min heap. We Build a Min Heap MH of the first k elements. For each element, after the kth element (arr[k] to arr[n-1]), compare it with root of MH, if the element is greater than the root then make it root and call heapify for MH, Else ignore it. Finally, MH has k largest elements and root of the MH is the kth largest element.

You are given an array of sorted words in an arbitrary language, you need to find order (or precedence of characters) in the language. For example if the given arrays is {“baa”, “abcd”, “abca”, “cab”, “cad”}, then order of characters is ‘b’, ‘d’, ‘a’, ‘c’. Note that words are sorted and in the given language “baa” comes before “abcd”, therefore ‘b’ is before ‘a’ in output.
Similarly we can find other orders.
This can be solved using two steps: First create a graph by processing given set of words, then do topological sorting of the created graph.