Week 2: Competitive Coding QSTP

Sorting

SUMMER 2021

Following document ONLY introduces reader, fundamentally to various sorting algorithms and their implementations. There is a good article present on Topcoder which comprehensively covers sorting and associated techniques, after getting a basic idea of sorting techniques, readers are recommended to go through the Topcoder article (link for which is provided at the end of this document) .

Selection Sort

The basic idea of selection sorts lies behind the fact that it finds the minimum element in each iteration (of the unsorted array) and puts the minimum element into its correct location. The process repeats and finally we get the sorted array. Technically speaking, every element is compared to every other element and the time complexity of the algorithm is $O(N^2)$. Following is the pseudo code for selection sort

Read pseudo code and practice: here

Bubble Sort

Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements and then swapping their positions if they exist in the wrong order. The average and worst case complexity of Bubble sort is O(N), however with a simple modification, we can take the best case complexity to be O(N) by simply maintaining a variable which tells us if the swap occurred in the current iteration or not. So if the array is sorted, N − 1 comparisons will happen and our swap variable will tell us that no swap occurred and hence, we can stop. This happens when the array is already sorted.

Read pseudo code and practice: here

Solve: Problem Link

Insertion Sort

The basic idea of insertion sort lies on the fact that in every iteration, one element from the original array is used up and we aim to find its correct location in the final sorted array. It works similar to how we sort playing cards. The algorithm iterates the input elements by growing the sorted array at each iteration. The average case complexity of the algorithm is $O(N^2)$ and the best case is O(N) (Occurs when the array is already sorted). Following is the pseudo-code for Insertion Sort –

Read pseudo code and practice: here

Merge Sort

1.4 Merge Sort Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.

Idea

• Divide the unsorted list into N sublists, each containing 1 element.

• Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. N will now convert into N 2 lists of size 2.

• Repeat the process till a single sorted list of obtained.

Note: The merge function simply merges the two arrays. It is important to note that the time complexity of mergesort algorithm is always O(NlogN). It is recommended that you try to code this atleast twice. You can try practicing merge sort here.

Quick sort

Quicksort also works in a divide and conquer fashion like merge sort. The idea of quicksort is to select a pivot element and partition the array in such a way that all elements to the left of the pivot are less than it and all elements to the right of it are more than it. Usually we pick up either the first or the last element as the pivot element. However, you are free to choose the pivot. The average case time complexity of the quicksort algorithm is O(NlogN). However, in the worst case, the time complexity can also go to $O(N^{2)}$ which does not happen in mergesort. Also, Quicksort does not use any auxilliary space unlike merge sort where an extra O(N) space is used. So as usual, there is always a trade-off between memory and time complexity. A good comparison of mergesort and quicksort is done here. This is very important from interviews perspective.

Read pseudo code and practice: here

Solve: Problem Link

In-Built Sort Function

Uses a combination of Quick Sort, Heap Sort and Insertion Sort to sort an array in O(N logN) time. Usage :

Array

int arr[n];

sort(arr, arr + n);

sort(arr, arr+n, greater<int>()); (sorts in DESCREASING ORDER)

Vector

vector<int>v;

// fill vector with elements

v.push_back(100); v.push_back(13); v.push_back(312); v.push_back(-1);

sort(v.begin(), v.end()); (SORTS elements in vector)

Custom Comparator Function

```
bool compare(int a, int b)

{

        return a>b;

}

sort( arr, arr+n, compare );
```

Suggested Read :
https://www.topcoder.com/community/competitive programming/tutorials/sorting/

Suggested Problems:

Descending Sort

Crofts Game

Chef and Card Trick

Counting Inversions**

Check overlapping intervals**

Ryouko's Memory Note