Divide and Conquer

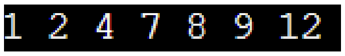
Assignment Solutions

Q1. Given an array where all its elements are sorted in increasing order except two swapped elements, sort it in linear time. Assume there are no duplicates in the array.
Input: arr[] = [3, 8, 6, 7, 5, 9, 10]
Output: arr[] = [3, 5, 6, 7, 8, 9, 10]

Solution :

Code : ASS_Code1.java

Output :



```
1 2 4 7 8 9 12
```

Approach :

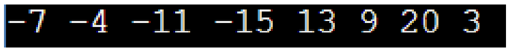
- The idea is to start from the second array element and compare every element with its previous element.
- We take two pointers, x and y, to store the conflict's location. If the previous element is greater than the current element, update x to the previous element index and y to the current element index.
- If we find that the previous element is greater than the current element, update y to the current element index.
- Now that we have got the x and y indices, swap the elements at index x and y.

Q2. Given an array of positive and negative integers, segregate them in linear time and constant space. The output should print all negative numbers, followed by all positive numbers.
Input: arr[] = {19, -20, 7, -4, -13, 11, -5, 3}
Output: arr[] = {-20 ,-4, -13, -5, 19 ,11 ,3, 7}

Solution :

Code : ASS_Code2.java

Output :



```
-7 -4 -11 -15 13 9 20 3
```

Approach :

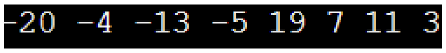
- The approach is much similar to the concept of quick sort algorithm where in this case we are using 0 as our pivot element because we know that any number less than 0 is termed as negative number and any number greater than 0 is termed as positive number.
- We have to make one pass of the partition process. The resultant array will satisfy the given constraints.
- In the partition function , each time we find a negative number, `pIndex` is incremented and that element would be placed before the pivot.

Q3. Given an array of positive and negative integers, segregate them in linear time and constant space. The output should print all negative numbers, followed by all positive numbers. The relative order of elements must remain the same.
Input: arr[] = {19, -20, 7, -4, -13, 11, -5, 3}
Output: arr[] = {-20 ,-4, -13, -5, 19 ,7 ,11, 3}

Solution :

Code : ASS_Code3.java

Output :



```
-20 -4 -13 -5 19 7 11 3
```

Approach :

- In the previous question, we discussed how to segregate positive and negative integers in linear time and constant space using quicksort's partitioning logic.
- The problem with this approach is that the relative order of elements has been changed in the array .In this question we are restricted to not change the relative order of the array elements. We will segregate positive and negative integers while maintaining their relative order using the logic of the merge sort algorithm.
- While merging the left and right subarray, we have to merge in a way that negative elements of both left and right subarrays are copied first, followed by positive elements of left and right subarrays.

Q4. Given two arrays of equal size n and an integer k. The task is to permute both arrays such that the sum of their corresponding element is greater than or equal to k i.e a[i] + b[i] >= k. The task is to print "Yes" if any such permutation exists, otherwise print "No".
Input : a[] = {2, 1, 3},
b[] = { 7, 8, 9 },
k = 10.
Output : Yes
Input : a[] = {1, 2, 2, 1},
b[] = { 3, 3, 3, 4 },
k = 5.
Output : No

Solution :

Code : ASS_Code4.java

Output :

```
No
```

Approach :

- The idea is to sort one array in ascending order and another array in descending order and if any index does not satisfy the condition a[i] + b[i] >= K then print "No", else print "Yes".
- If the condition fails on sorted arrays, then there exists no permutation of arrays that can satisfy the inequality. Proof,
- Assume asort[] be sorted a[] in ascending order and bsort[] be sorted b[] in descending order.
- Let new permutation b[] is created by swapping any two indices i, j of bsort[],
- Case 1: i < j and element at b[i] is now bsort[j].
- Now, asort[i] + bsort[j] < K, because bsort[i] > bsort[j] as b[] is sorted in decreasing order and we know asort[i] + bsort[i] < k.
- Case 2: i > j and element at b[i] is now bsort[j].
- Now, asort[j] + bsort[i] < k, because asort[i] > asort[j] as a[] is sorted in increasing order and we know asort[i] + bsort[i] < k.

Q5. An interval is represented as a combination of start time and end time. Given a set of intervals, check if any two intervals intersect.
Input: arr[] = {{1, 3}, {5, 7}, {2, 4}, {6, 8}}
Output: Yes
The intervals {1, 3} and {2, 4} overlap
Input: arr[] = {{1, 3}, {7, 9}, {4, 6}, {10, 13}}
Output: No

Solution :

Code : ASS_Code5.java

Output:

```
No
```

Approach :

- A Simple Solution is to consider every pair of intervals and check if the pair intersects or not.
- A better solution is to Use Sorting.
- Find the overall maximum element. Let it be max_ele.
- Initialize an array of size max_ele with 0.
- For every interval [start, end], increment the value at index start, i.e. arr[start]++ and decrement the value at index (end + 1), i.e. arr[end + 1]- -.
- Compute the prefix sum of this array (arr[]).
- Every index, i of this prefix sum array will tell how many times i has occurred in all the intervals taken together. If this value is greater than 1, then it occurs in 2 or more intervals.
- So, simply initialize the result variable as false and while traversing the prefix sum array, change the result variable to true whenever the value at that index is greater than 1.