Assignment Solution

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, 7 ,11 ,19 ,3}

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 an integer array nums, return the number of reverse pairs in the array.

A reverse pair is a pair (i, j) where:

0 <= i < j < nums.length and

nums[i] > 2 * nums[j].

Example 1:

Input: nums = [1,3,2,3,1]

Output: 2

Explanation: The reverse pairs are:

(1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 * 1

(3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 * 1

Example 2:

Input: nums = [2,4,3,5,1]

Output: 3

Explanation: The reverse pairs are:

(1, 4) --> nums[1] = 4, nums[4] = 1, 4 > 2 * 1

(2, 4) --> nums[2] = 3, nums[4] = 1, 3 > 2 * 1

(3, 4) --> nums[3] = 5, nums[4] = 1, 5 > 2 * 1

solution :

Code : ASS_Code4.java

Output :

```
The number of reverse pairs are : 2
```

Approach :

- Here, we have used the concept of dividing the problem into smaller parts and solving for the same cause.
- we know that by dividing any array into two sub arrays, the indices in the first(left) half of the array would be lesser than the indices in the right half.
- From here we can compare two subarrays at any point of time for the condition a[i] > 2*a[j] where index i will be from the first subarray and index j will be from the second subarray.
- After checking we can merge the two sorted halves as we always do in the merge sort. Sorting here will ensure that if any particular ‘i’

index from the left subarray qualifies for the reverse pair with any index ‘j’ from the right subarray then all the remaining elements from the first subarray will qualify for the reverse pair criteria since all the elements in the first half at index greater than ‘i’ will be greater than the element at index ‘i’. This is ensured by sorting.

- For any index ‘i’ if the criteria is not satisfied then it will definitely be not setting for elements at indices greater than ‘j’. This information is also available because of sorted arrays. Then we need to increase the ‘i’ value.
- This way for any 2 subarrays we can calculate the total number of reverse pairs.
- After calculation we will merge the two subarrays to get the result for bigger arrays and so on until we get back to our original array.

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.