Offline Assignment 1

1. **Counting Inversions in an Array** **(10)**

Given an array[]. The task is to find the inversion count of a[]. Where two elements a[i] and a[j] form an inversion if a[i] > a[j] and i<j.

```
Input: arr[]  = {8, 4, 2, 1}

Output: 6

Explanation: Given array has six inversions: (8,4), (4,2), (8,2), (8,1), (4,1), (2,1)

Input: arr[]  = {1, 20, 6, 4, 5}

Output: 5

Explanation: Given array has five inversions: (20,6), (20,4), (20,5), (6,4), (6,5)
```

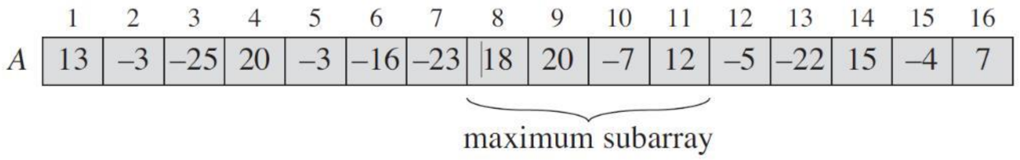
You can modify the mergesort algorithm in the following way to solve this problem:

1. Divide the array into two equal/almost equal halves in each of the recursive step until the base case is reached.
2. Create a merge function that counts the number of inversions when two halves of the array are merged in the following way:
    a. Let us consider i be the index of the first half and j be the index of the second half.
    b. If a[i] > a[j[then there are (mid – i) inversions because the left and right subarrays are sorted, so all the remaining elements in the left subarray (a[i+1], a[i+2], a[i+3]) will be greater than a[i]
3. Create the required recursive function that divides the array into right and left halves and find the answer by summing the number of inversions in the first half and the number of inversions in the second half by simple summation.
    a. The base case is when there is only one element in the given half
4. Print the final answer.

2. **Maximum subarray sum** **(10)**

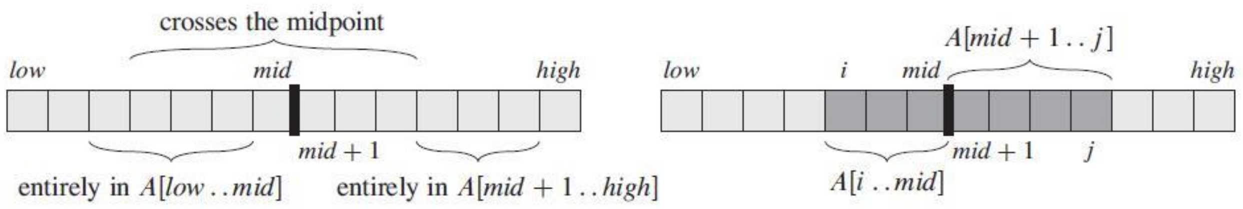
You are given a one-dimensional array that may contain both +ve and -ve integers, find out the sum of continuous subarray which has the largest sum.

For example,



Steps to follow:

i. Each time divide the array into two halves.
   a. Recursive call the first half to return you the maximum subarray sum of that portion.
   b. Recursive call the second half to return you the maximum subarray sum of that portion.
   c. Maximum subarray may exist around the mid-point. So, calculate the maximum subarray sum across the split boundary.



How to calculate the maximum crossing subarray sum:

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = −∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = −∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

ii. Base condition: If the array contains only 1 item then the maximum subarray sum is the item itself.

3. **Rotation Count:** **(10)**

Given a rotated sorted array of distinct integers, your task is to find out the total number of rotations.

For example,

{ 15, 17, 1, 2, 6, 11 } is rotated 2 times.
{ 7, 9, 11, 12, 5 } is rotated 4 times.

Observation: The index of the minimum element represents the total number of rotations. (check yourself)

Steps to follow:

i. If the array is empty array then the array has no rotations i.e. return 0.

ii. If the array contains only one element then the index of that element is the rotation count.

iii. If the array contains more than one element, then find out the middle element

   a. If the middle element is smaller than its immediate left element then this middle element is the minimum element so return the index of the middle element.

   b. If the middle element is smaller than the rightmost element, then the minimum element will be found in the left subarray. So, search the left array recursively.

   c. If the middle element is not smaller than the rightmost element, then the minimum element will be found in the right subarray. So, search the right array recursively.