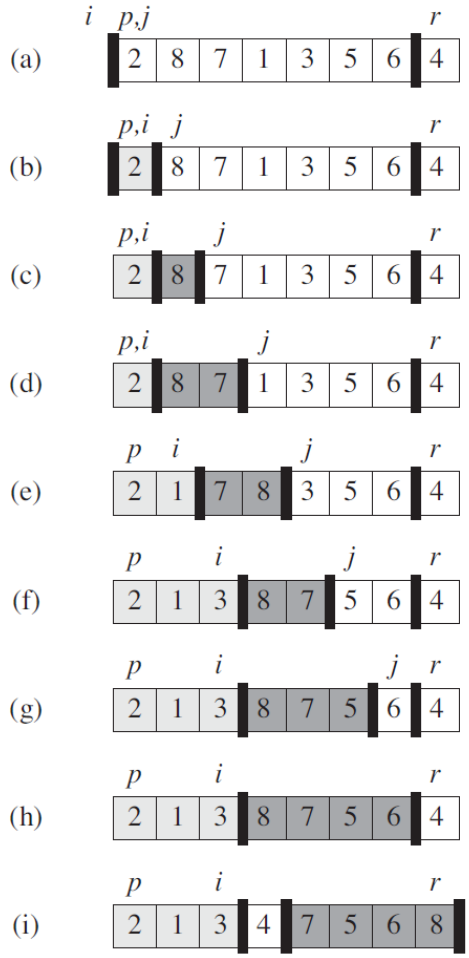
Offline Assignment 3

1. **Quick Sort**

In quick sort, we use a **pivot** element and partition the input array such that the items smaller or equal than the pivot element stays on the left side and the items greater than the pivot element stays in the right side.

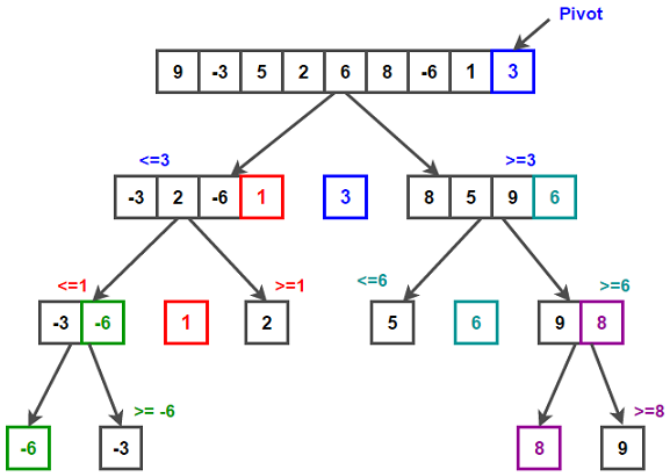
The partition works as follow:




```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

After a partition, the pivot element will be placed between the left subarray and right subarray. Then again call the quicksort for sorting your left half and again call quick sort for sorting your right half.

Base condition: If the array is empty i.e. startind>endind then return (you have no elements to sort) and also if the array contains only one element then also return (only 1 element doesn't need any more sorting).



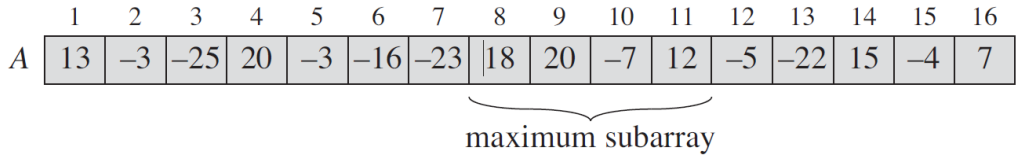
Template:

```
#include <iostream>
using namespace std;

int partition_arr(int arr[], int startind, int endind){
    ///Partition the array here and return the index of the pivot item after
partition
}

void QuickSort(int arr[], int startind, int endind){
    if(startind>endind){
        ///empty array
    }
    else if(startind==endind){
        ///array with 1 element
    }
    else{
        ///Partition the array

        ///QuickSort the left subarray

        ///QuickSort the right subarray
    }
}

int main()
{
    int arr[]={2,8,7,1,3,5,6,4};
    int sz=sizeof(arr)/sizeof(int);

    QuickSort(arr,0,sz-1);

    ///printing the output
    for(int ind=0;ind<sz;ind++){
        cout<< arr[ind] <<" ";
    }
    cout<<endl;

    return 0;
}
```

2. **Maximum subarray sum**

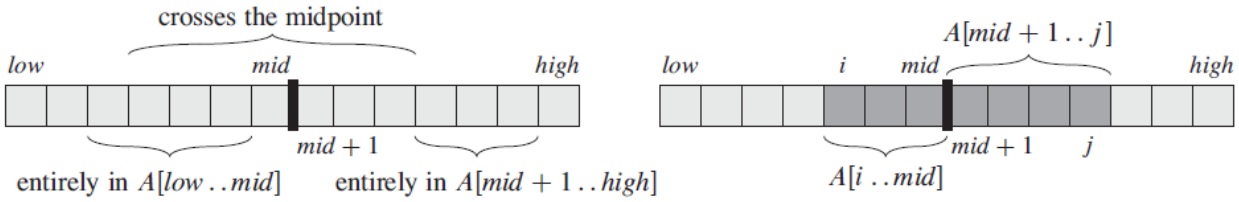
You are given a one-dimensional array that may contain both +ve and -ve integers, find out the sum of continuous subarray which has the largest sum.

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)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  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:**

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.

[Miss at your own risk]

[Any kind of plagiarism is strictly prohibited.]