Practical - 2

Aim : Write programs to implement the following using Divide and Conquer design Str.

Core : implement following algos. and print numbas of comparisons and swap/shifts for each cases

case 1: already sorted
case 2: reverse sorted
case 3: random order

Algo 1: Quick Sort (take last element as pivot)

```
quicksort (arr[], low, high)
    if (low < high)
        pi = partition (arr, low, high);
        quicksort (arr, low, pi - 1)
        quicksort (arr, pi+1, high)

Partition (arr[], low, high)
    Pivot = arr[high]
    i = (low - 1)
    for (j = low ; j <= high - 1; j++)
        if (arr[j] < pivot) {
            i++;
            swap 'arr[i] and arr[j]
    swap arr[i+1] and arr[high]
    return i+1
```

Code:

```
int partition(vector<int> &arr, int low, int high, int swap,
              int &comp) {
    int pivot = arr[high];
    int i = low - 1;
    for(int j = low; j <= high-1; j++) {
        comp
        swap++;
        if(arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
            swap++;
            comp++;
        }
        swap++;
        swap(arr[i+1], arr[high]);
        return(i+1);
    }

Void quicksort(vector<int> &arr, int low, int high,
               int &s, int &c) {
    if(low < high) {
        int pi = partition(arr, low, high, s, c);
        quicksort(arr, low, pi-1, s, c);
        quicksort(arr, pi+1, high, s, c);
    }
}
```

Case 1: 7, 9, 12, 13, 17

swap => 14

Comp = 20

Case 2: 17, 13, 12, 9, 7

swap => 8

Comp => 14

Case 3: 7, 17, 13, 9, 12

swap => 6

Comp => 9

→ time complexity in quick sort based on the which pivot element you choose

Algo 2: Merge sort

```
Void merge(vector<int> &arr, int left, int mid, int right){
    int arr1 = mid - left + 1;
    int arr2 = right - mid;
    int *larr = new int[arr1];
    int *rarr = new int[arr2];

    for(int i=0; i<arr1; i++)
        larr[i] = arr[left+i];
    for(int j=0; j<arr2; j++)
        rarr[j] = arr[mid+1+j];

    int idx1=0, idx2=0, midx=left;
```

```
while(idx1<arr1 && idx2 <arr2) {
    if(larr[idx1] <= larr[idx2]){
        arr[midx] = larr[idx1];
        idx1++;
    }
    else {
        arr[midx]= larr[idx2];
        idx2++;
    }
    midx++;
}
while (idx1 < arr1) {
    arr[midx] =larr[idx1];
    idx++;
    midx++;
}
while ( idx2 < arr2) {
    arr[midx] = larr[idx1];
    idx++;
    midx++;
}
while (idx2 <arr2){
    arr[midx] = larr[idx2];
    idx2++;
    midx++;
}
delete[] larr;
delete[] larr;
};
```

```
Void mergeSort (vector<int>&a1, int ~~const~~ begin,
                int, end)
    functionCall++;
    if (begin >= end)
        return;
    int mid = begin + (end - begin)/2;
    mergeSort(a1, begin, ~~end~~ mid);
    mergeSort (a1, mid+1, end);
    merge(a1, begin, mid, end);
}
```

Case 1: 7, 9, 12, 13, 17
Function call = 9

Case 2: 17, 13, 12, 9, 7
function call = 9

Case 3: 7, 17, 13, 9, 12
function call = 9

there are no comp and swaping in merge sort all we do is divide and combine and for that we creating new array.
↳ merge sort ~~fit~~ lies in blind sort no matter in which order your data is it always ~~call~~ take same time

Algo 3 : Study and implementation of maximum sum sub array problem.

method 1: Check the all the sub array sum (O(n²))

```
int start = 0, end = 0; ans = INT_MIN;
int MaxSumofSubAr (int vector<int>& v) {
    for (int i=0 ; i<v.size() ; i++) {
        int sum = v[i];
        if (sum>ans) {
            ans = sum;
            start = i;
            end = i;
        }
        for (int j=i+1 ; j<v.size() ; j++) {
            sum += v[j];
            if (sum > ans) {
                ans = sum;
                start = i;
                endd = j;
            }
        }
    }
    return ans;
}
```

In: -2, 5, -3, 6, -1

Ou: Sum : 8

Start: 1

end : 3

Method 2: Divide And conquer. (O(n log n))

```
int start = 0;
int end = 0;
int ans = INT_MIN;

int solver(vector<int> &v, int i, int j){
    if(i>=j){
        return 0;
    }
    int mid = i + (j-i)/2; // finding the mid
    // going in left subarray ~~int~~ include mid in the
    int s1 = solver(v, i, mid);
    // going in Right subarray,
    int s2 = solver(v, mid+1, j);
    int ss1 = INT_MIN; // finding ss
    int ss2 = INT_MIN;
    int l = mid;
    int sum = 0;

    for(int k = mid; k>=i; k--){
        sum += v[k];
        ss1 = max(ss1, sum);
        if(sum == ss1){
            l = k;
        }
    }
```

```
sum = 0;
int r = mid;
for(int k = mid+1; k <= j; k++) {
        sum += V[k];
        s32 = max(s32, sum);
        if (sum == s32) {
                r = k;
        }
}

int s3 = s3l + s3r;
if (s1 > ans && s1 > s2 && s1 > s3) {
        start = i;          // finding starting point
        end = mid;          // finding ending point
}
else if (s2 > ans && s2 > s1 && s2 > s3) {
        start = mid+1;
        end = j;
}
else if (s3 > ans && s3 > s1 && s3 > s2) {
        start = r;
        end = r;
}
return ans = max(max(s1, s2), s3);
}
```

Ex: 12000, 41, 61, -12, -5312, 14141, -70

Sum: 128919 , start = 0 , end = 5