Name: Md Habibur Rony
Student ID: 984582
Weekday: Week 1- Day 5

Answer to the Q. No. R-4.2:

```
Algorithm margeSort(s,c)
Input:S is the secquence of n elements and C is the comparator
Output: S the sorted sequence of n elements

if s.size() = 1 then
        return s

(s1,s2)<--pertition(s,n/2)

s1<--margeSort(s1,c)
s2<--margeSort(s2,c)

S<--marge(s1,s2, c)
return s
```

Answer to the Q. No. R-4.5:

```
Algorithm GetCountRemovingDuplicate(A, B)
Input: A,B sequence of elements
Output: count value of the merging elements

S1 <- RemoveDuplication(A)                         O(n)
S2 <- RemoveDuplication(B)                         O(n)

i<--0                                              O(1)
while i<s1.size() do                               O(n)
        s.insert(s1[i])                            O(n)
        count <--count+1                           O(n)
        i<--i+1                                    O(n)

i<--0                                              O(1)
while i<s2.size() do                               O(n)
        s.insert(s2[i])                            O(n)
        count <--count+1                           O(n)
        i<--i+1

                                                   O(n)
```

```
return count                                              T(n) = O(n)

Algorithm RemoveDuplication(S)
Input: S sequence of elements
Output: S1 Sequence of elements without duplication

previous<--s[0]                                           O(1)
s1.insert(s[0])                                           O(1)
i<--0                                                     O(1)

while i<s.size() do
    if previous  != s[i] then                             O(n)
        s1.insert(s[i])                                   O(n)
    endIf                                                 O(n)
    i<--i+1

return s1                                                 O(n)
                                                          O(1)

                                                          T(n) = O(n)
```

Answer to the Q. No. C-4.9:
The best case of the Quicksort is O(n log n) and worst case is O(n^2) . If the pivot is in the middle of the element range then it is good with probability 1/2. Therefore, in this case the run time will be O(n logn)

Answer to the Q. No. C-4.10:

```
Algorithm CountingVoteForGettingWiner(s)
Input: S is a list of elements
Output: Winer ID
ss<-MargeSort(s,c)                                   O(nlogn)
previousVote<--0                                     O(1)
previousId<--ss[0]                                   O(1)
CurrentVote<--0                                      O(1)
sz<-s.sidze()                                        O(1)
currentSize <--0                                     O(1)
winerId<-Nul                                         O(1)

while currentSize <sz  do                            O(n)
    if previousId = ss[courentSize] then             O(n)
        CurrentVote <-- CurrentVote+1                O(n)
    else
```

```
                if currentVote >previousVote then         O(n)
                        previousVote<- correntVote        O(n)
                        previousId<- currentId            O(n)
                        currentVote<-1                    O(n)
                endIf

        endIf
                currentId<-- ss[sz]                       O(n)
                currentSize <- currentSize+1              O(n)

if currentVote>previousVote then                          O(1)
        winerId<- currentId                               O(1)
else
        winerId<- previousId                              O(1)
endIf                                                     O(1)
return winerId                                            T(n) = O(nlogn)
```