<u>Datastructures And Algorithms</u>
<u>CS 202</u>
<u>Assignment-1 Report</u>

<u>Sorting Algorithms:</u>

1) Insertion Sort:

T(n) = O(n^2);
S(n) = O(1)

PseudoCode:

```
for j = 2 to n
        key = A [j]
        j = i – 1
        while i > 0 and A[i] > key
                A[i+1] = A[i]
         i = i – 1
        A[j+1] = key
```

Remarks:

Insertion sort is based on the idea that one element from the input elements is consumed in each iteration to find its correct position. In class it was demonstrated with the example of cards picked in the sorted order.

2) Merge Sort:

T(n) = O(nlogn)
S(n) = O(n)

PseudoCode:

Merge (A, p, q, r):

```
1.n1= q – p +1
n2= r – q
let L [1.. n1+ 1 ] and L [1.. n2+ 1 ] be new arrays
for i=1 to n1
                L[ i ] = A [ p + i -1]
for j=1 to n2
                R[ j ]= A[ q + j ]
L [n1+ 1 ] = infinity
R [n2+ 1 ] = infinity
i = 1
j = 1
for k = p to r
if L[ i ]≤R [ j ]
                A[ k ] = L[ i ]
                i = i + 1
else A[ k ] = R [ j ]
                j = j + 1
```

MergeSort(A,low,high):

```
if low<high:
```

```
            mid = (low+high)/2
            MergeSort(A,low,mid)
            MergeSort(A,mid+1,high)
            Merge(A,low,mid,high)
```
Remark:
It is based on divide and conquer strategy.

3) Quick Sort:

T(n)(worst) = O(n^2), average = O(nlogn)
S(n) = O(1)

PseudoCode:

```
Quicksort(A, low, high)
      if (low < high)
            pivot-location = Partition(A,low,high)
            Quicksort(A,low, pivot-location - 1)
            Quicksort(A, pivot-location+1, high)


Partition(A,low,high)
      pivot = A[low]
      leftwall = low
      for i = low+1 to high
            if (A[i] < pivot) then
                  leftwall = leftwall+1
                  swap(A[i],A[leftwall])
      swap(A[low],A[leftwall])
```

Remark:
It is also based on divide and conquer strategy.

4) Heap Sort:

T(n) = O(nlogn)
S(n) = O(n)

pseudoCode:

```
     Heapsort(A)
       BuildHeap(A)
       for i <- length(A) downto 2 {
         exchange A[1] <-> A[i]
         heapsize <- heapsize -1
         Heapify(A, 1)
     BuildHeap(A)
       heapsize <- length(A)
       for i <- floor( length/2 ) downto 1
   Heapify(A, i)
     Heapify(A, i)
```

```
le <- left(i)
ri <- right(i)
if (le<=heapsize) and (A[le]>A[i])
   largest <- le
else
   largest <- i
if (ri<=heapsize) and (A[ri]>A[largest])
   largest <- ri
if (largest != i) {
   exchange A[i] <-> A[largest]
   Heapify(A, largest)
```

Remark:
The children of each node is smaller then itself. Root is the max value of the array.
Balance factor = {1,0}