Assignment - 4

1.) Write an algorithm / code for selection sort.

→ Algorithm :-

1. Repeat (size - 1) times
   (i) minimum ← first element
   (ii) for each unsorted elements
        if element < minimum
        minimum ← element
   (iii) Swap (minimum to element)
2. STOP

2.) Write an algorithm / code for Bubble sort.

→ Algorithm :

1. Repeat (size - 1) times
   (i) for each element
        if left element < right element
        then swap left element and right element
2. END

3. Write an algorithm / code for merge sort.

→ MERGE (A, p, q, r)

n1 = q - p + 1

n2 = r - q

let L[1 ... n1 + 1] and R[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] = infinite

R[n2 + 1] = infinite

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

MERGE SORT (A, p, r)

if p < r

then q ← (p + r) / 2

MERGE SORT (A, p, q)

MERGE SORT (A, q + 1, r)

MERGE (A, p, q, r)

4) Write an algorithm / code for Insertion sort

→ Algorithm :

```
for i = 1 to size
    key ← arr[i]
    j ← i - 1

repeat while (key < arr[j] and j >= 0)
    arr[j+1] ← arr[j]
    --j
arr[j+1] ← key
End for

STOP
```

5) Write an algorithm / code for Quick sort.

→ Algorithm :

```
Procedure Pivot(T[i...j]; var l)
{ Rearranges the elements in array T[i...j] and
returns a value l such that, at end, i <= l <= j,
T[k] <= p ∀ i <= k < l, T[l] = p, and
T[k] > p ∀ l < k <= j, p is the initial
value T[i] }

p ← T[i]
k ← i;
l ← j+1
```

Repeat $k \leftarrow k+1$ until $T[k] \geq p$
Repeat $l \leftarrow l-1$ until $T[l] \leq p$

while $k < l$
    swap $T[k]$ and $T[l]$
    Repeat $k \leftarrow k+1$ until $T[k] \geq p$
    Repeat $l \leftarrow l-1$ until $T[l] \leq p$
swap $T[i]$ and $T[l]$

Procedure quickSort ($T[i \ldots j]$)
if $j-i$ is sufficiently small then insert $T[i \ldots j]$
else
    Pivot ($T[i \ldots j], l$)
    quickSort ($T[i \ldots l-1]$)
    quickSort ($T[l+1, \ldots j]$)

6) Write an algorithm/code for Heap sort.

→ Algorithm :

- maxHeapify (array, i)
    L = left (i)
    R = Right (i)
    if L = heap_size (array) and array [L] > array [i]
        Largest = L
    else
        Largest = i

if R = heap-size [arr] and arr [R] > arr [largest]
largest = R

if largest ≠ i
exchange arr [i] with arr [largest]
MaxHeapify (arr, largest)
End

- Build Max Heap (arr)
heap size (arr) = length (arr)
for i = length (arr) /2 to 1
MaxHeapify (arr, i)
END

- HeapSort (arr)
Build max Heap (arr)
for i = length (arr) to 2
exchange arr [i] with arr [i]
heap-size [arr] = heap-size [arr] - 1
MaxHeapify (arr, 1)
End.

7.) Write an algorithm /code for linear search.

for i = 0 to last index of array
if A [i] equals key
return i

return -1

Q) Write an algorithm / code for Binary Search.

```
while left <= right
    middle = index halfway between left, right

    if A[middle] matches key
        return middle
    else if key < A[middle]
        right = middle - 1

    else
        left = middle + 1

return -1
```