Pseudocode of Linear Search

```
LINEAR SEARCH(A, key)    // Pseudocode of Linear Search
1. found = 0
2. for i = 1 to A.length
3.         if A[ i ] = key
4.             found = 1
5.             return i
6. if found = 0
7.     return 0
```

Binary Search – Algorithm

```
BinarySearch (A, m, n, k)
   if (m>n) return -1;    //Base Case
   mid=(m+n)/2
   if A[mid]=k  return mid;     //Base Case
   else if k < A[mid]
      BinarySearch(A, m, mid-1, k)
   else if k > A[mid]
      BinarySearch(A, mid+1, n, k)
```

Insertion Sort

```
INSERTION SORT(A)
for j=2 to A.length
        key = A[ j ];
        //Insert A[ j ] into the sorted sequence A[1...j-1]
        i = j-1
        while i > 0 and A[ i ] > key
                A[ i+1 ]=A[ i ]
                i=i-1
        A[ i+1 ]= key
```

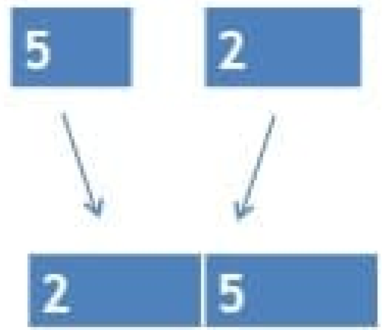
Merge Sort – Recursive Algorithm

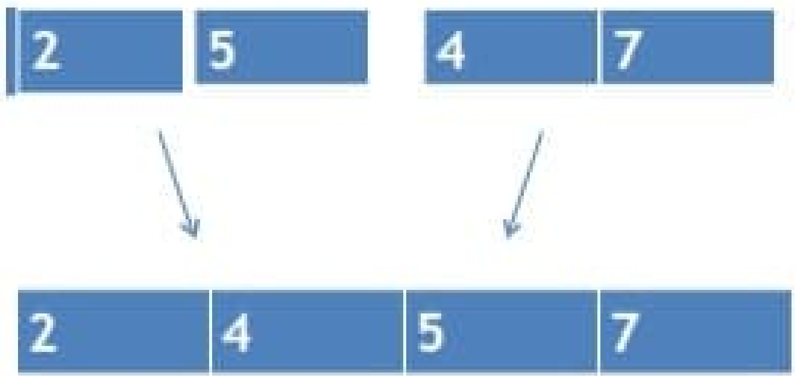
```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Ref: CLRS Book

Merge function

```
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] = ∞        Sentinel - a special value
R[n2 + 1] = ∞
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
```





Ref: CLRS Book

Pseudocode of Quick Sort

```
QUICKSORT(A, p, r)

if p < r
    then q <- PARTITION(A, p, r)
         QUICKSORT(A, p, q-1)
         QUICKSORT(A, q+1, r)
```

Pseudo code : *PARTITION*

```
PARTITION (A, p, r)
x = A[r]
i= p - 1
for j = p to r - 1
    do if A[j] <= x
        then  i=i+1
        Exchange A[i] with A[j]
Exchange A[i +1] with A[r]
return i +1
```

MAX-HEAPIFY

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
        largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
        largest = r
if largest ≠ i
        exchange A[i] with A[largest]
        MAX-HEAPIFY(A, largest)
```

BUILD-MAX-HEAP

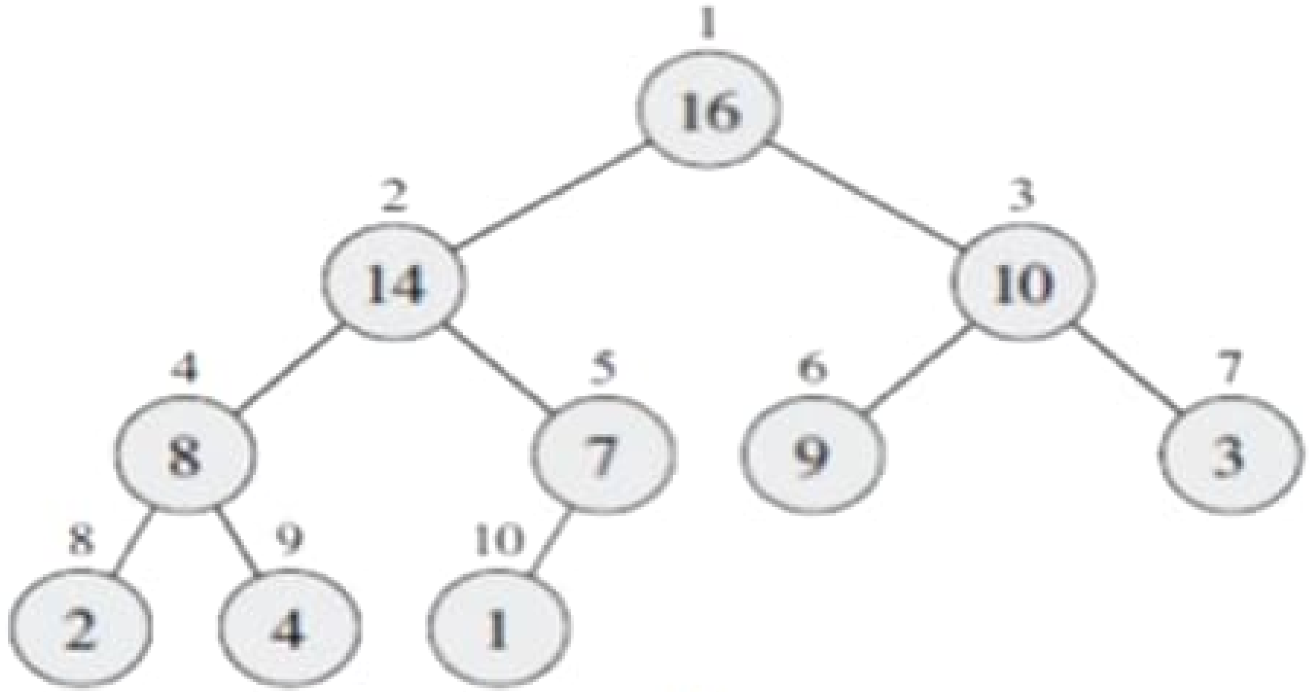
Pseudocode for BUILD-MAX-HEAP

```
BUILD-MAX-HEAP(A)
  A.heap-size = A.length
  for i = ⌊A.length/2⌋ downto 1
      MAX-HEAPIFY(A, i)
```

Heap Sort: ALGORITHM

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```

MAXIMUM(A) ?



(a)

HEAP-MAXIMUM(A)
return A[1]

Extract-Max operations

HEAP-EXTRACT-MAX(A)

```
if A.heap-size < 1
error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size - 1
MAX-HEAPIFY(A, 1)
return max
```

HEAP-INCREASE-KEY

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

MAX-HEAP-INSERT

MAX-HEAP-INSERT(A, key)

1 $A.heap\text{-}size = A.heap\text{-}size + 1$
2 $A[A.heap\text{-}size] = -\infty$
3 HEAP-INCREASE-KEY$(A, A.heap\text{-}size, key)$