Page No.
Date:

EXPERIMENT-6

AIM To implement HeapSort and analyze its time complexity.

THEORY

HeapSort is a comparison based sorting technique based on binary heap data structure. It is similar to selection sort where we first find the max element & place the max element at the end. We repeat the same process for remaining elements.

```
HeapSort (A)
Build MaxHeap (A)
  heapsize ← length (A)
   for j ← n down to 2
       A[i] ⇔ A[j]
       heapsize ← heapsize - 1
       MaxHeapify (A, 1)

Build MaxHeap (A)
   for i ← ⌊n/2⌋ down to 1
       MaxHeapify (A, i)
         l ← 2i
         r ← 2i+1
         if [ A[largest] < A[r] and r ≤ heapsize ]
           largest ← r
```

Complexity Analysis

The recurrence relation for the MAX-HEAPIFY function is given by

$$T(n) = \underbrace{T(2n/3)}_{\substack{\text{Time to run MaxHeapify()}\\ \because \text{The children's subtrees each have size at most } 2n/3}} + \underbrace{\Theta(1)}_{\substack{\text{Time to fix up the relationships among the elements}\\ A[i],\ A[left(i)],\ A[right(i)]}}$$

The worst case occurs when the last row of the tree is exactly half full and the running time of MAX-HEAPIFY can therefore be described by the above recurrence.

Using Master Theorem for $T(n) = aT(n/b) + \Theta(n^c)$

where $a = 1$, $b = \frac{3}{2}$ $c = 0$

$\log_b a = \log_{3/2} 1 = 0$ $\log_b a = c$

$\therefore$ By Case 2 of Master Theorem

$$T(n) = \Theta(n^c \log_2 n)$$
$$= \Theta(\log_2 n)$$

The Time complexity of the build heap function is $\Theta(n)$ since it contains a for loop that runs for $n/2$ times

$\therefore$ The overall time complexity of heapsort algorithm is

$T(n) = \Theta(n \log ...$

```
if ( largest ≠ i )
    A[ largest ] ↔ A[ i ]
    MaxHeapify (A, largest )
```

Page No.
Date:

EXPERIMENT-7

AIM To find the largest Common subsequence of two strings and analyse the time complexity of algorithm.

THEORY

```
LCS - length (X, Y)
   m ← length (X)
   n ← length (X)
   for i ← 0 to m
      C[i, 0] ← 0
     for j ← 1 to m
         for j ← 1 to n
            if Xi = yj
               C[i,j] ← C[i-1, j-1] + 1
               b[i, j] ← "↖"
            else if C[i-1, j] ≥ C[i, j-1]
               C[i, j] ← C[i-1, j]
                  b[i, j] ← "↑"
            else
                  C[i, j] ← C[i, j-1]
                     b[i, j] ← "←"
  return C and b
```

Page No.
Date:

PRINT - LCS (b, x, i, j)
// The initial invocation is PRINT-LCS (b, x, length (x), length ((y))

if i = 0 or j = 0
  return
if b[i, j] = "↖"
  PRINT - LCS (b, x, i-1, j-1)
  Print x_j
else if b[i,j] = "↑"
  PRINT - LCS (b, x, i-1, j)
else
  PRINT - LCS (b, x, 1, j-1)

Scanned by CamScanner