Lecture No – 16 to 18

Heap Sort & Priority Queue

Branch/Section: IT-5, CSE-3 & CSE-10, B.Tech, 5th Sem. **Faculty: Prof. Anil Kumar Swain**

Lecture Summary

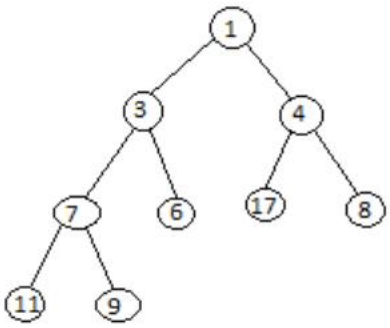
1. Heap Sort : Introduction, Building Max-Heap & Min-Heap, Heap-Sort, Analysis of Time Complexity
2. Priority Queues: Max-Priority Queues & Min-Priority Queues

1 Heap Sort

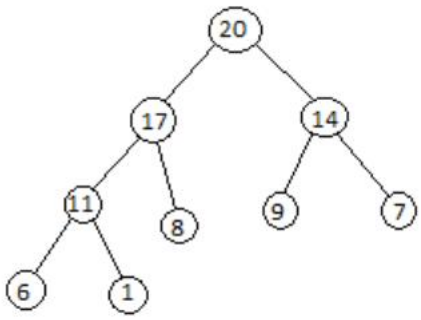
1.1 Introduction

- Heap Sort is a popular and efficient sorting algorithm in computer programming. Learning how to write the heap sort algorithm requires knowledge of two types of data structures - **arrays and trees.**
- Heaps can be used in sorting an array.
- A Binary Heap is a **Complete Binary Tree** where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap.
- In other words, Heap is a special tree-based data structure, that satisfies the following special heap properties :
    a) **Shape Property:**Heap data structure is always a **Complete Binary Tree** (The tree is completely filled on all levels except possibly the last level, which is filled from the left to right).
    b) **Value Property:** Value in each node of the tree is greater than or equal to the values in its two children nodes, called **max. Heap. Or** Value in each node of the tree is smaller than or equal to the values in its two children nodes, called **min. Heap.**

Example:



Min-Heap



Max-Heap

1.2 How to create a Max-Heap from an array of n elements

- **How to create a complete binary tree from an unsorted list (array)?**
  - Select first element of the list to be the root node. (First level - 1 element)
  - Put the second element as a left child of the root node and the third element as a right child. (Second level - 2 elements)
  - Put next two elements as children of left node of second level. Again, put the next two elements as children of right node of second level (3rd level - 4 elements).
  - Keep repeating till you reach the last element.

- **Relationship between array indexes and tree elements:** Complete binary tree has an interesting property that we can use to find the children and parents of any node.

If root is at index 1	If root is at index 0
/*Algorithm that returns the parent index of a node having index i*/. PARENT(i) { return i/2; } **/*Algorithm that returns the left child index of a node having index i*/.** LEFT (i) { return 2*i; } **/*Algorithm that returns the right child index of a node having index i*/.** RIGHT (i) { return 2*i+1; }	**/*Algorithm that returns the parent index of a node having index i*/.** PARENT(i) { return (i-1)/2; } **/*Algorithm that returns the left child index of a node having index i*/.** LEFT (i) { return 2*i+1; } **/*Algorithm that returns the right child index of a node having index i*/.** RIGHT (i) { return 2*i+2; }

- **Maintaining the heap property:** In order to maintain the max-heap property, we call the procedure MAX-HEAPIFY. That is Starting from a complete binary tree, we can modify it to become a Max-Heap by running a function called heapify on all the non-leaf elements of the heap.
- MAX-HEAPIFY lets the value at A[i] "float down" in the max-heap so that the subtree rooted at index i obeys the max-heap property.

MAX-HEAPIFY	BUILD-MAX-HEAP
/***Max-Heapify :** Given a tree that is a heap except for node i, Max-Heapify function arranges node i and it's subtrees to satisfy the heap property.*/ MAX-HEAPIFY(A, n, i) { l ← LEFT(i);	/***Build Max-Heap :** Using MAX-HEAPIFY() we can construct a max-heap by starting with the last node that has children and iterating back to the root calling MAX-HEAPIFY() for each node which ensures that the max-heap property will be maintained at each step for all evaluated nodes.*/

```
    r ← RIGHT(i);
    if  (l ≤ n and A[l] > A[i])
        largest = l;
    else
        largest = i;
    if  (r ≤ n and A[r] > A[largest])
        largest = r;
    if (largest != i)
    {
       A[i] ↔ A[largest]; // swaping
       MAX-HEAPIFY(A, n, largest);
    }
}
```

Time Complexity of MAX-HEAPIFY on a node of height h is O(h).

```
BUILD-MAX-HEAP(A, n)
{
   for (i ← n/2 down to 1)
   {
      MAX-HEAPIFY(A, n, i);
    }
}
```

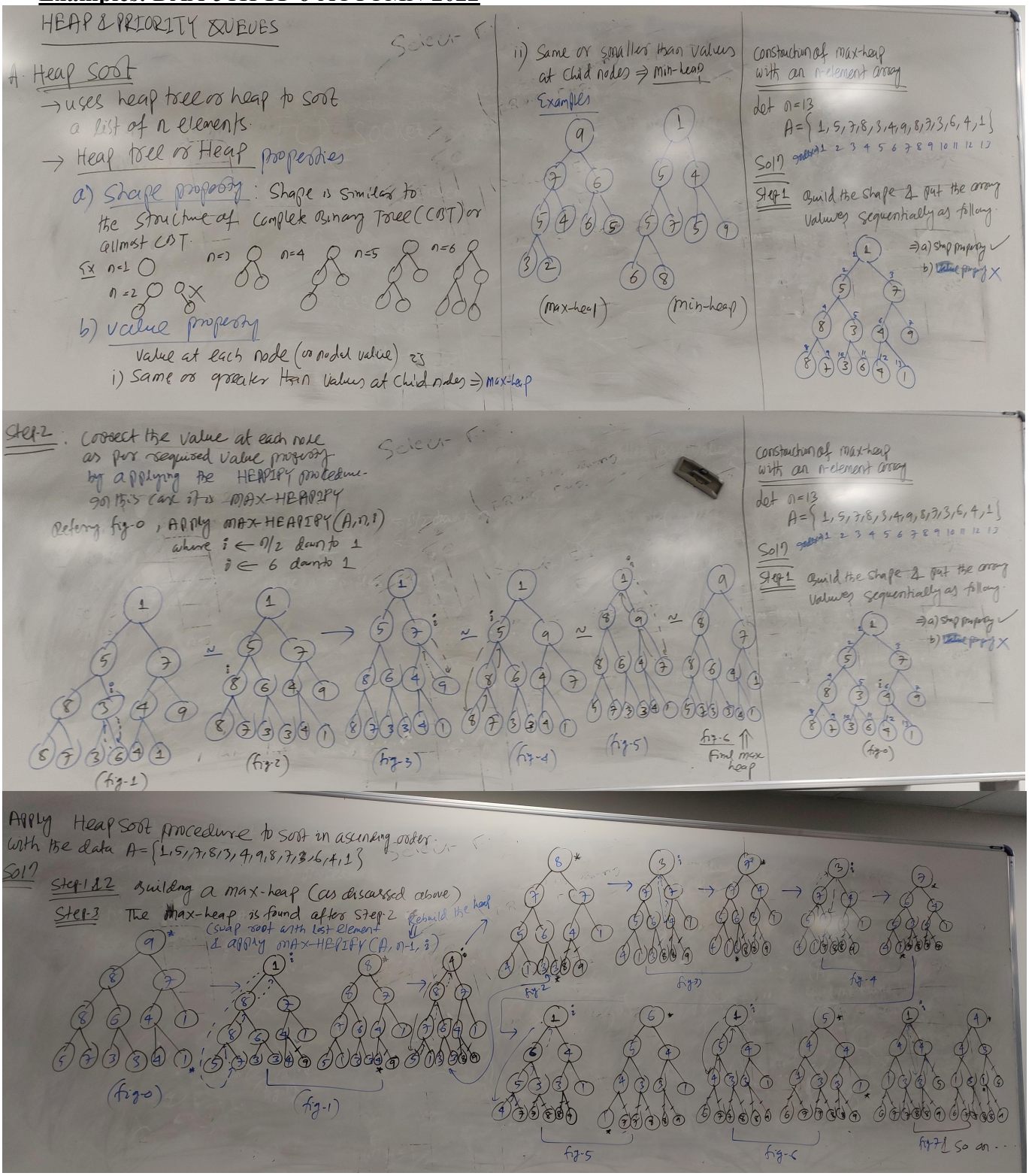
Time Complexity of BUILD-MAX-HEAP procedure is O(n).

```
HEAP-SORT(A, n)
{
  BUILD-MAX-HEAP(A, n);
  for (i ← n down to 2)
  {
       A[1] ↔ [i]; // swap last element with root
       MAX-HEAPIFY(A, i-1, 1);
   }
}
```

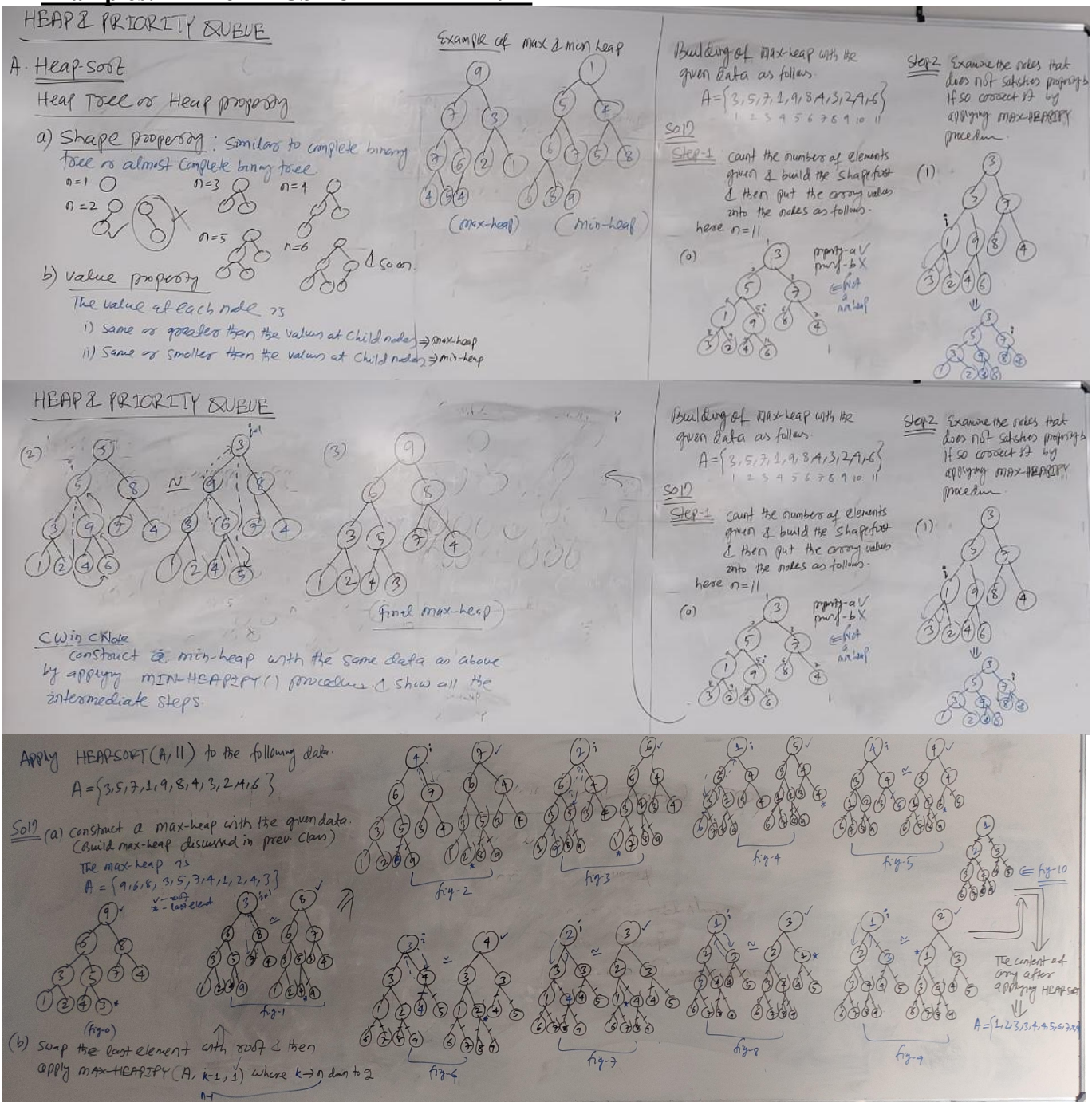
Time Complexity of HEAP-SORT Procedure is O(n log n).

1.3 Class Discussion - Examples on MAX-HEAP-SORT

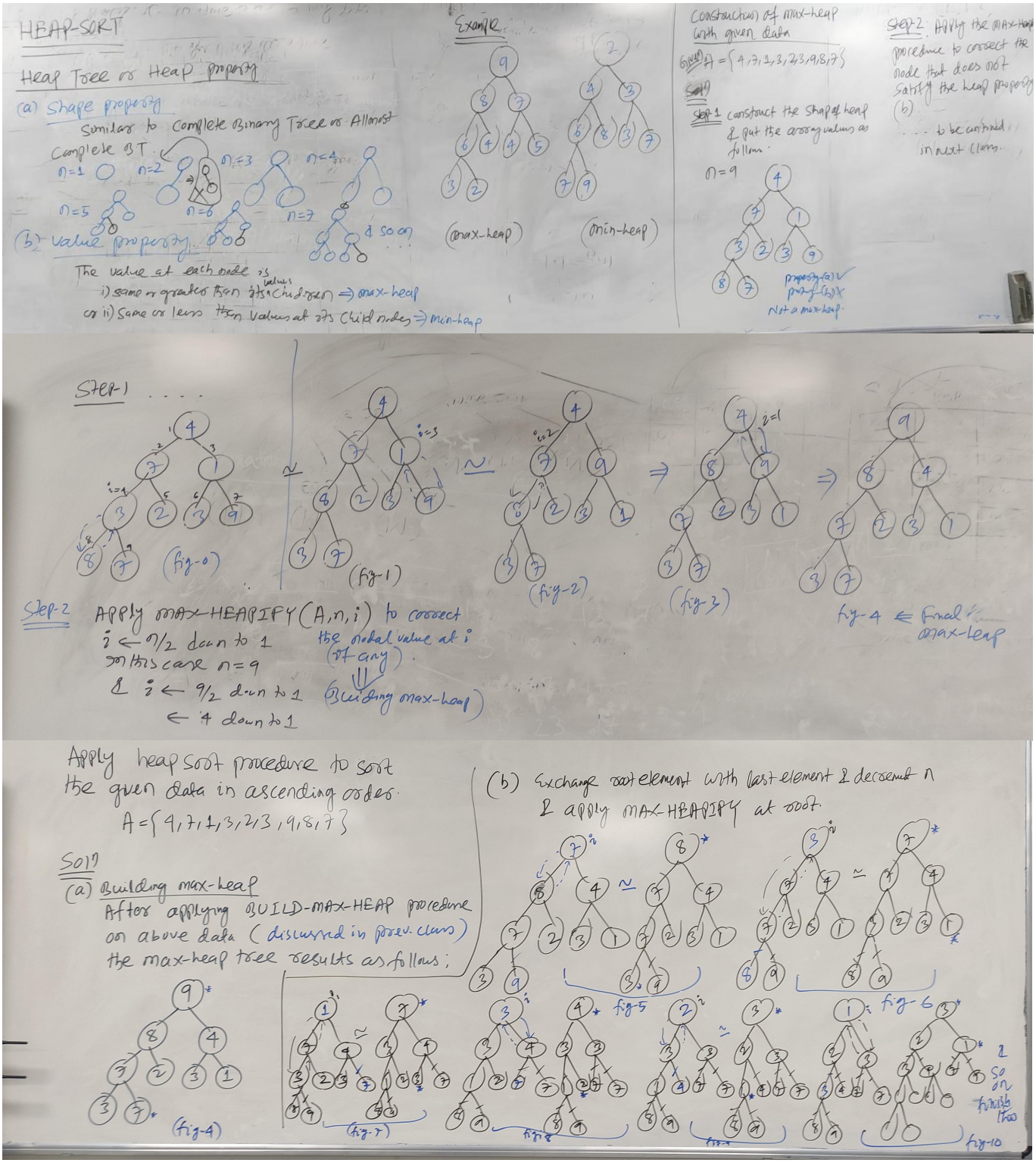
- **Examples:** DAA-5TH-**IT-6**-AUTUMN-2022



- **Examples:** DAA-5TH-**CSE-3**-AUTUMN-2022



- **Examples:** DAA-5TH-**CSE-10**-AUTUMN-2022



2 Priority Queues

2.1 Introduction

- A priority queue is a data structure for maintaining a set S of elements, each with an associated value called a key. There are two priority queues. Max and min priority queue.
- A max-priority queue implemented by max-heap, supports the following operations:

Max Priority Queue
1. **HEAP-MAXIMUM (A, n)** is an algorithm that returns the largest value stored in an n-element max heap A. 2. **HEAP-EXTRACT-MAX (A, n)** is an algorithm that rebuild the heap after removing the largest value and finally returns that largest value stored in an n-element max heap A. 3. **HEAP-INCREASE-KEY (A, n, i, key)** is an algorithm that rebuild the heap if value at index i increases to the new value key, which is assumed to be at least as large as i's current key value, else display appropriate error message. 4. **MAX-HEAP-INSERT (A, n, key)** is an algorithm that rebuild the heap after inserting a new value key to an n-element max heap A.

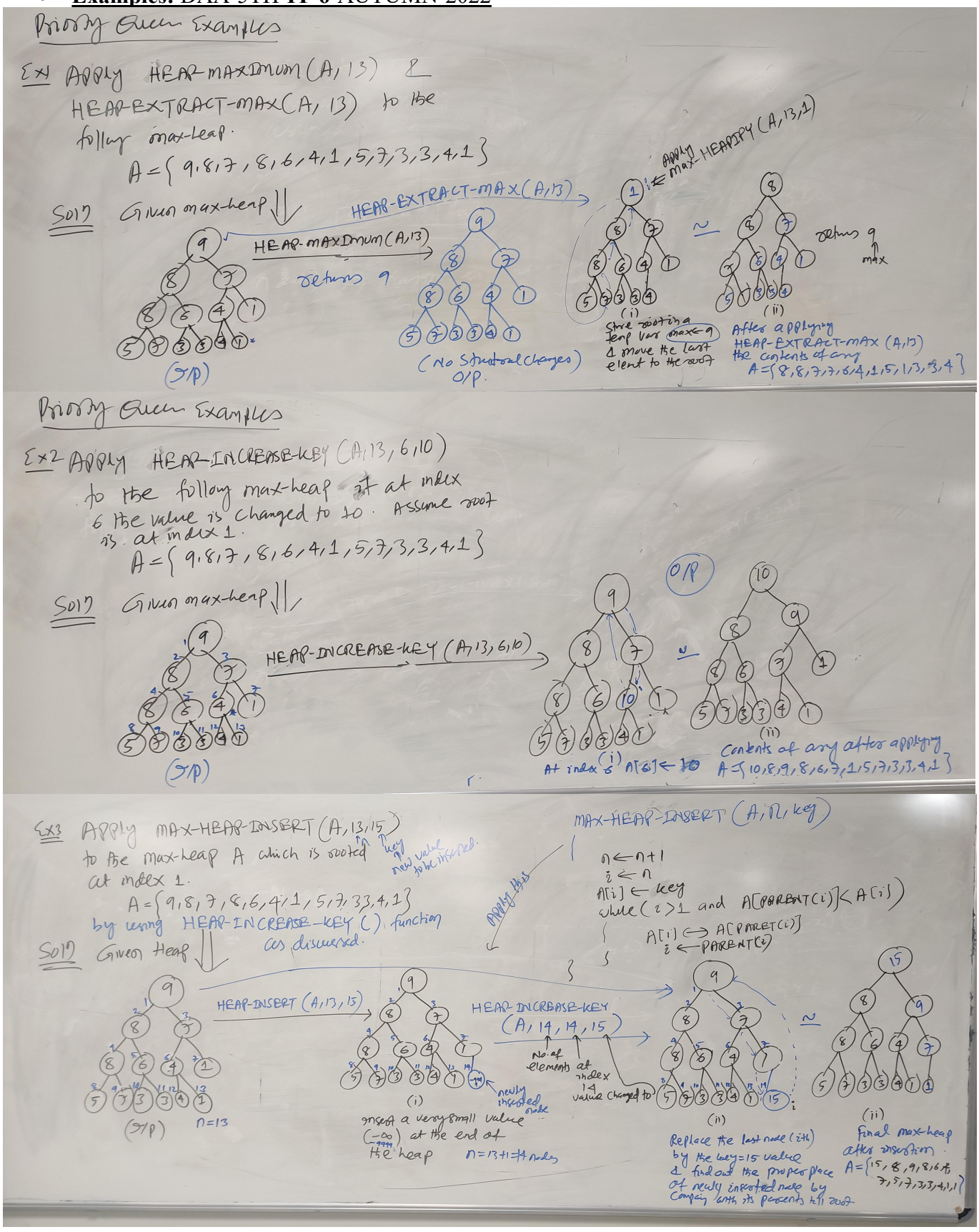
- The definitions of **Max Priority Queue** algorithms are:

HEAP-MAXIMUM () Algorithm	HEAP-EXTRACT-MAX () Algorithm
HEAP-MAXIMUM (A, n) { return A[1]; }	HEAP-EXTRACT-MAX (A, n) { if (n < 1) { Print "error-heap underflow" exit; } //Store root element in a temp. variable max max ← A[1]; //Replace root by last element A[1] ← A[n]; n ← n - 1; MAX-HEAPIFY(A, n, 1); return max; }
Time Complexity: O (1)	**Time Complexity:** O (lg n)

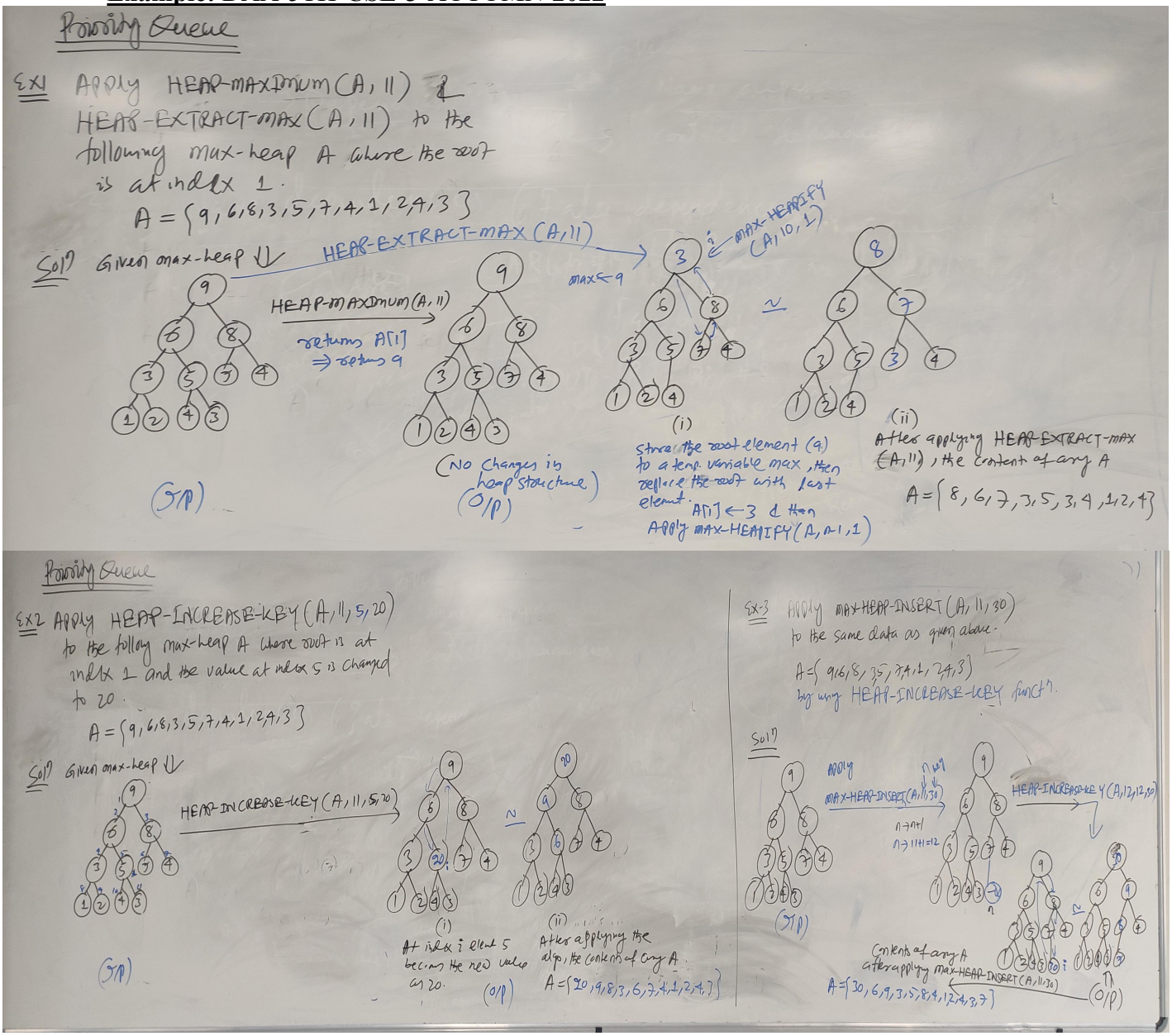
HEAP-INCREASE-KEY() Algorithm	MAX-HEAP-INSERT() Algorithm
HEAP-INCREASE-KEY (A, n, i, key) { if (key < A[i]) { Print "error-new key is smaller than current key"; Exit; } A[i] ← key; while (i>1 and A[PARENT(i)] < A[i]) { A[i] ↔ A[PARENT(i)]; i ← PARENT(i); } }	MAX-HEAP-INSERT (A, n, key) { n ← n + 1; A[n] ← -∞; HEAP-INCREASE-KEY (A, n. n, key); }
Time Complexity: O (lg n)	**Time Complexity:** O (lg n)

2.2 Class Discussion - Examples on MAX-PRIORITY QUEUES

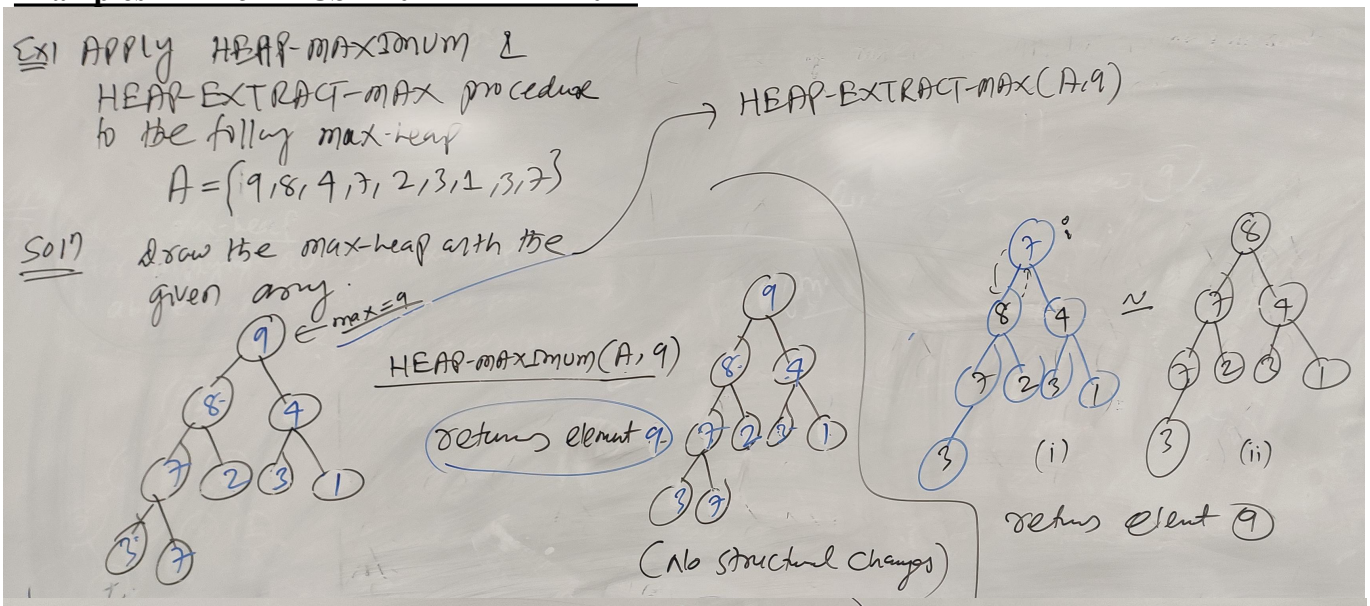
- **Examples:** DAA-5TH-**IT-6**-AUTUMN-2022

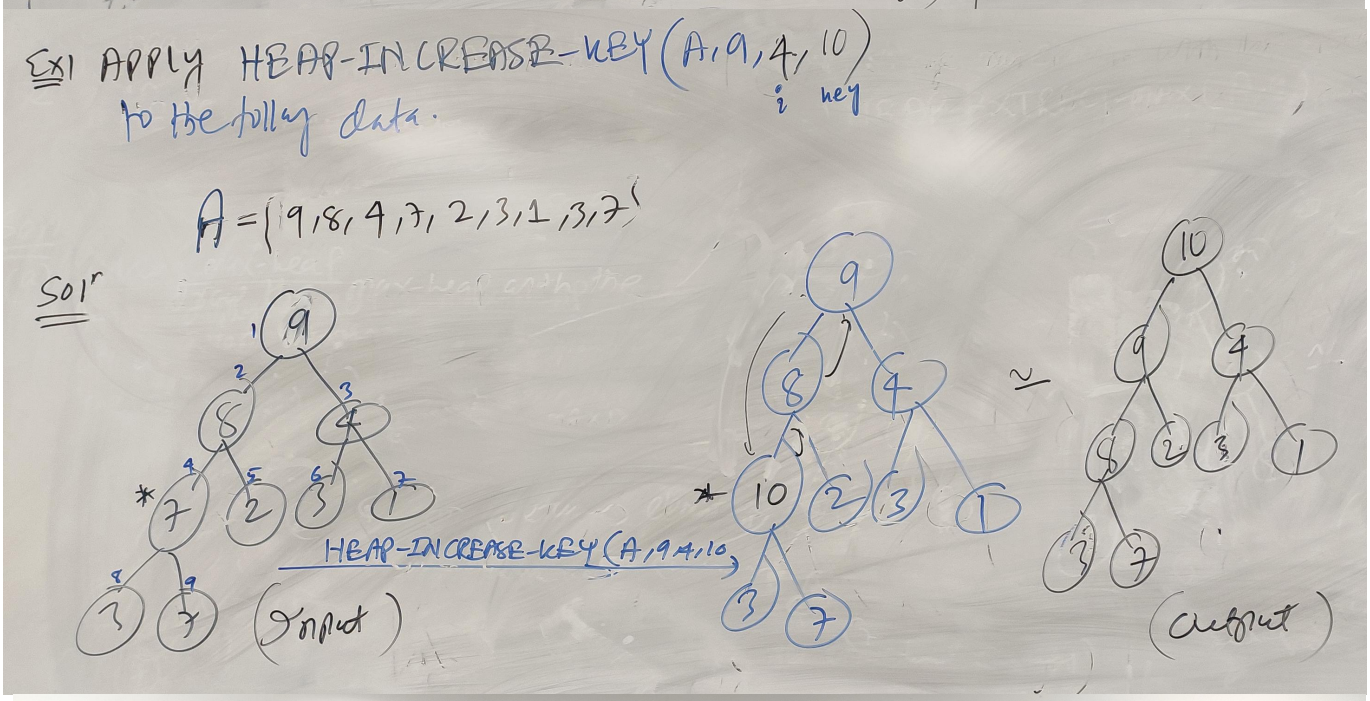


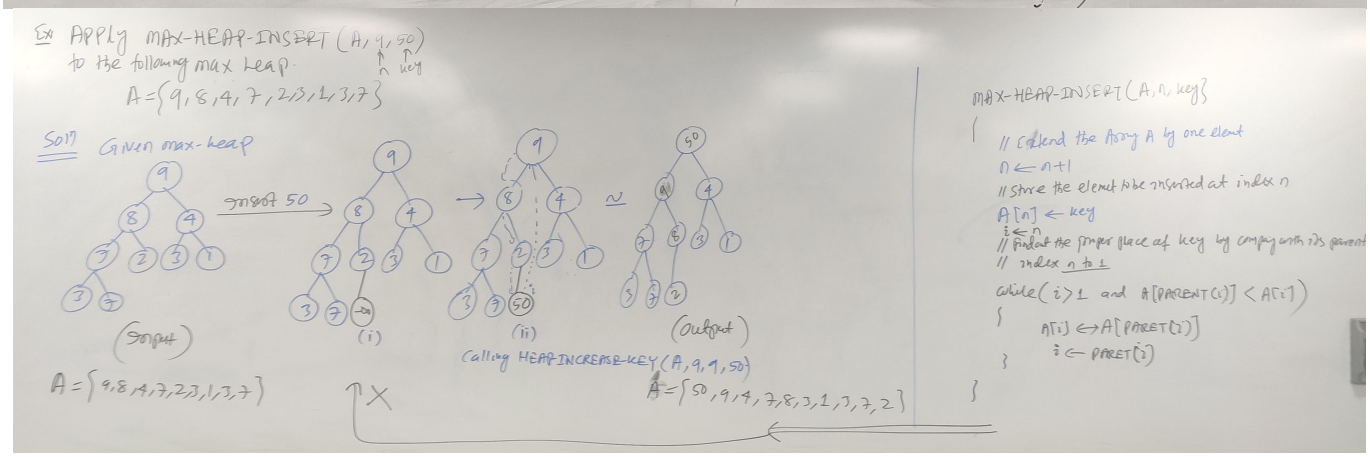
- **Example:** DAA-5TH-**CSE-3**-AUTUMN-2022



- **Examples**-DAA-5TH-**CSE-10**-AUTUMN-2022







3 Sample Questions & Answers/Solutions

3.1 Consider a complete binary tree where the left and the right subtrees of the root are max-heaps. What is upper bound to convert the tree onto a heap? (2015-MID-REG-KIIT-UNIV)

Solution

Answer: $O(\log_2 n)$

Explanation:

In the worst case applying MAX-HEAPIFY(1) will terminate at leaf that visits the heights of the tree.

3.2 Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted, or almost sorted (at most two elements are misplaced).

A. Quick Sort B. Heap Sort C. Merge Sort D. Insertion Sort

(2016-MID-REG-KIIT-UNIV)

Solution

Answer:

Option: D

If sorted is assumed in proper order

Option: B and D

If sorted is assumed in reverse order

Explanation:

sorted in proper order (almost)

- Quick sort performs worse case time complexity ($O(n^2)$), Merge and Heap performs its best case time complexity ($O(n \log n)$) and Insertion sort performs its best case time complexity that is $O(n)$. Hence Insertion sort will give the best solution.

sorted in reverse order (almost)

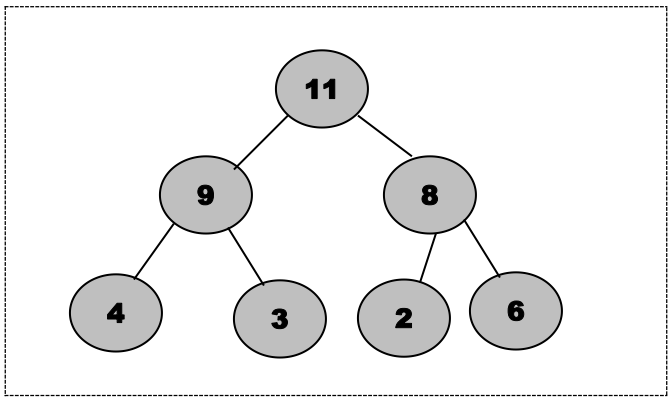
- Quick sort and insertion sort performs worse case time complexity ($O(n^2)$), Merge and Heap performs its best case time complexity ($O(n \log n)$) and Hence Merge and Heap sort will give the best solution.

3.3 A priority queue is implemented as a Max-Heap. Initially, it has 5 elements. The level order traversal of heap is as 11, 9, 6, 4, 3. Two new elements 2 and 8 and inserted in the heap in that order. Find out the level order traversal of the heap after the insertion of the elements?

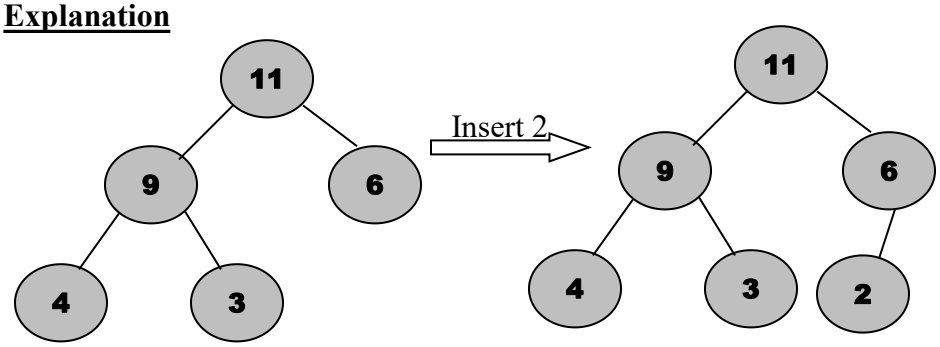
(2016-MID-REG-KIIT-UNIV)

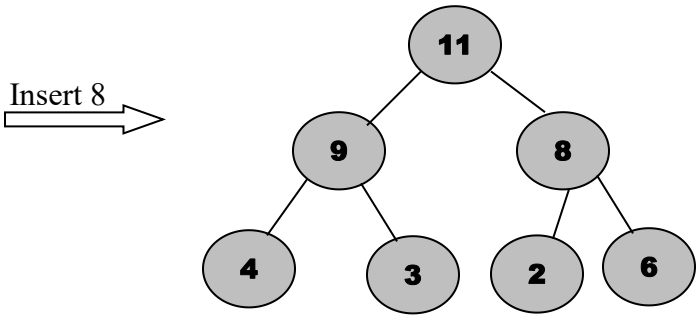
Solution

The level order traversal of the heap after the insertion of the elements 2 and 8 is: 11, 9, 8, 4, 3, 2, 6



Explanation



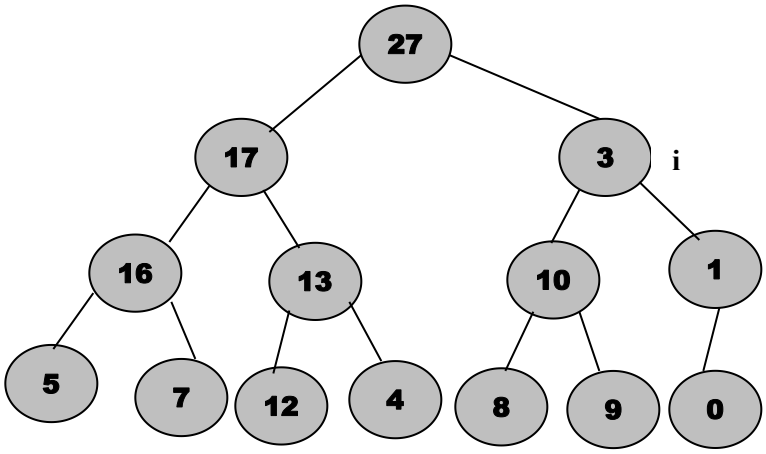


3.4 Write the algorithm for MAX-HEAPIFY(A,i). Illustrate the operation of MAX-HEAPIFY(A,3) on the array A={27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0}

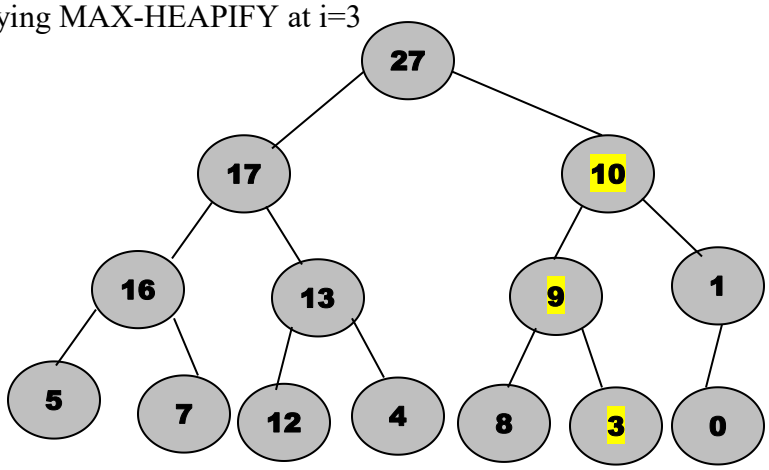
(2016-MID-REG-KIIT-UNIV)

Solution

Write here the **MAX-HEAPIFY(A,i) algorithms here.**



Applying MAX-HEAPIFY at i=3



3.5 Write pseudocode for the procedure HEAP-EXTRACT-MIN. Explain with the given min-heap level order traversal A={10, 20, 15, 22, 30, 18, 17, 40, 28}.

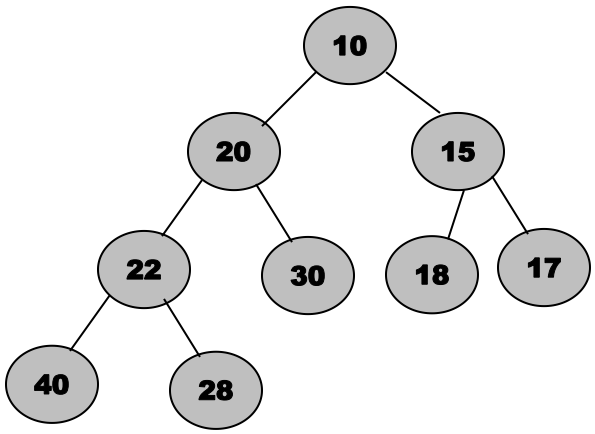
(2016-MID-REG-KIIT-UNIV)

Solution

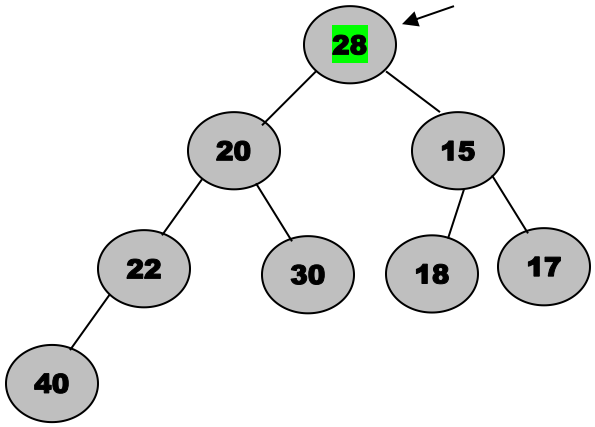
```
//This algorithm assumes the array representing min-heap starts at index 1
HEAP-EXTRACT-MIN(A)
{
  if heap-size[A] < 1
      error "Heap Underflow"
 min←A[1]
 A[1]←A[heap-size[A]]
 heap-size[A]← heap-size[A]-1
 MIN-HEAPIFY(A,1)
 return min
}
```

Explanation

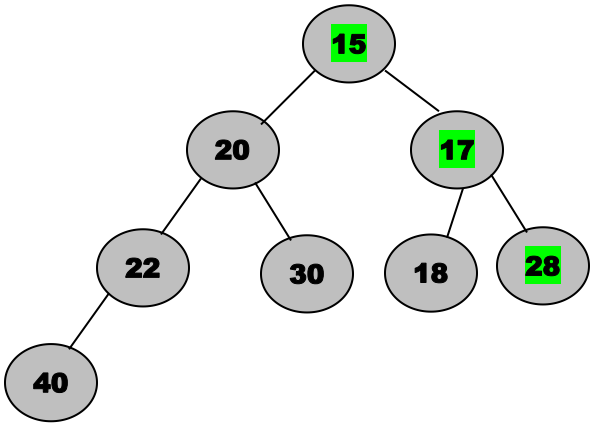
The given level order traversal A={10, 20, 15, 22, 30, 18, 17, 40, 28} yield the following min-heap tree.



Applying HEAP-EXTRACT-MIN on the above min-heap will return the minimum value that is on the root (i.e. 10) and this node can be replaced by the last element (i.e. 28) a now apply MIN-HEAPIFY on this node that is MIN-HEAPIFY(A,1)



After applying MIN-HEAPIFY(A,1) that is nothing but to re-build the heap, becomes



3.6 Show that there are at most $n/2^{h+1}$ nodes of height h in any n-element heap.

(2016-MID-REG-KIIT-UNIV)

Solution

Height of a node - Longest distance from a leaf to that node

Depth of a node – Distance from root to that node

If the heap is not a full binary tree (bottom level is not full), then the nodes at a given level (depth) don't all have the same height. For example, although all nodes at depth d have height 0, nodes at depth d−1 can have either height 0 or height 1.

Proof By induction on h

Basis: Show that it is true for h = 0 (i.e.No of leaf nodes$\leq\lceil n/2^{h+1}\rceil=\lceil n/2\rceil$

The tree leaves (nodes at height 0) are at depths d and d−1. They consist of

- all nodes at depth d, and
- the nodes at depth d−1 that are not parents of depth-d nodes.

Let x be the number of nodes at depth d□hat is, the number of nodes in the bottom (possibly incomplete) level.

Rest (n-x) nodes above the bottom level form a complete binary tree, and a complete binary tree has an odd number of nodes (1 less than a power of 2). Thus if n is odd, x is even, and if n is even, x is odd.

Proving the base case:

a) **If x is even**, there are x/2 nodes at depth d−1 that are parents of depth d nodes, hence $2^{d-1}-x/2$ nodes at depth d−1 that are not parents of depth-d nodes. Thus,
Total no. of height-0 nodes= nodes at depth d + nodes at depth d-1

$$= x + (2^{d-1}-x/2)$$
$$= 2^{d-1}+x/2$$
$$= \lceil (2^d+x-1)/2\rceil \quad (\text{as x is even})$$
$$= \lceil n/2\rceil$$

($n = 2^d + x - 1$ because the complete tree down to depth d−1 has $2^d - 1$ nodes and depth d has x nodes.)

Hence proved.

b) **If x is odd**, by an argument similar to the even case, we see that
Total no. of height-0 nodes= nodes at depth d + nodes at depth d-1

$$= x + (2^{d-1}-(x+1)/2)$$
$$= 2^{d-1}+(x-1)/2$$
$$= (2^d+x-1)/2$$
$$= n/2$$
$$= \lceil n/2\rceil \quad (\text{because x odd} \Rightarrow \text{n even})$$

Hence proved.

Inductive step: Show that if it is true for height h−1, it is true for h.
Let n_h be the number of nodes at height h in the n-node tree T .
Consider the tree T1 formed by removing the leaves of T . It has $n1 = n-n_0$ nodes.
We know from the base case that $n_0 = \lceil n/2 \rceil$ so $n1 = n-n_0 = n-\lceil n/2 \rceil = \lfloor n/2 \rfloor$
Now, the nodes at height h in T would be at height h − 1 if the leaves of the tree were removed—that is, they are at height h−1 in T1. Letting $n1_{h-1}$ denote the number of nodes at height h − 1 in T1, we have
$n_h = n1_{h-1}$
By induction, we can bound $n1_{h-1}$:
$n_h = n1_{h-1} \le \lceil n1/2^h \rceil = \lceil \lfloor n/2 \rfloor / 2^h \rceil \le \lceil (n/2)/2^h \rceil = \lceil n/2^{h+1} \rceil$ Hence proved.

3.7 Fill in the blanks.

_____ is the best case Time Complexity to find out the smallest element in a binary MAX-HEAP containing n numbers.

(2017-MID-REG-KIIT-UNIV)

Solution

O(n) (Smallest element can be found from n/2+1 to n in max-heap A[1..n]

3.8 Write the algorithm to build a MAX-HEAP? Describe your Algorithm to build a MAX-HEAP from the array A = {5, 7, 8, 2, 1, 0, 3, 9, 4, 5, 6} in a step by step process. Derive the time complexity of building a MAX-HEAP.

(2017-MID-REG-KIIT-UNIV)

Solution

```
BUILD-MAX-HEAP(A, n)
{
   for i ← n/2 down to 1
       MAX-HEAPIFY(A, i, n)
}
```

```
MAX-HEAPIFY(A, n, i)
{
   l ← LEFT(i )
  r ← RIGHT(i )
  if l ≤ n and A[l] > A[i ]
        largest ←l
  else largest ←i
  if r ≤ n and A[r ] > A[largest]
       largest ←r
  if largest ≠ i
  {
     exchange A[i ] ↔ A[largest]
    MAX-HEAPIFY(A, largest, n)
  }
}
```

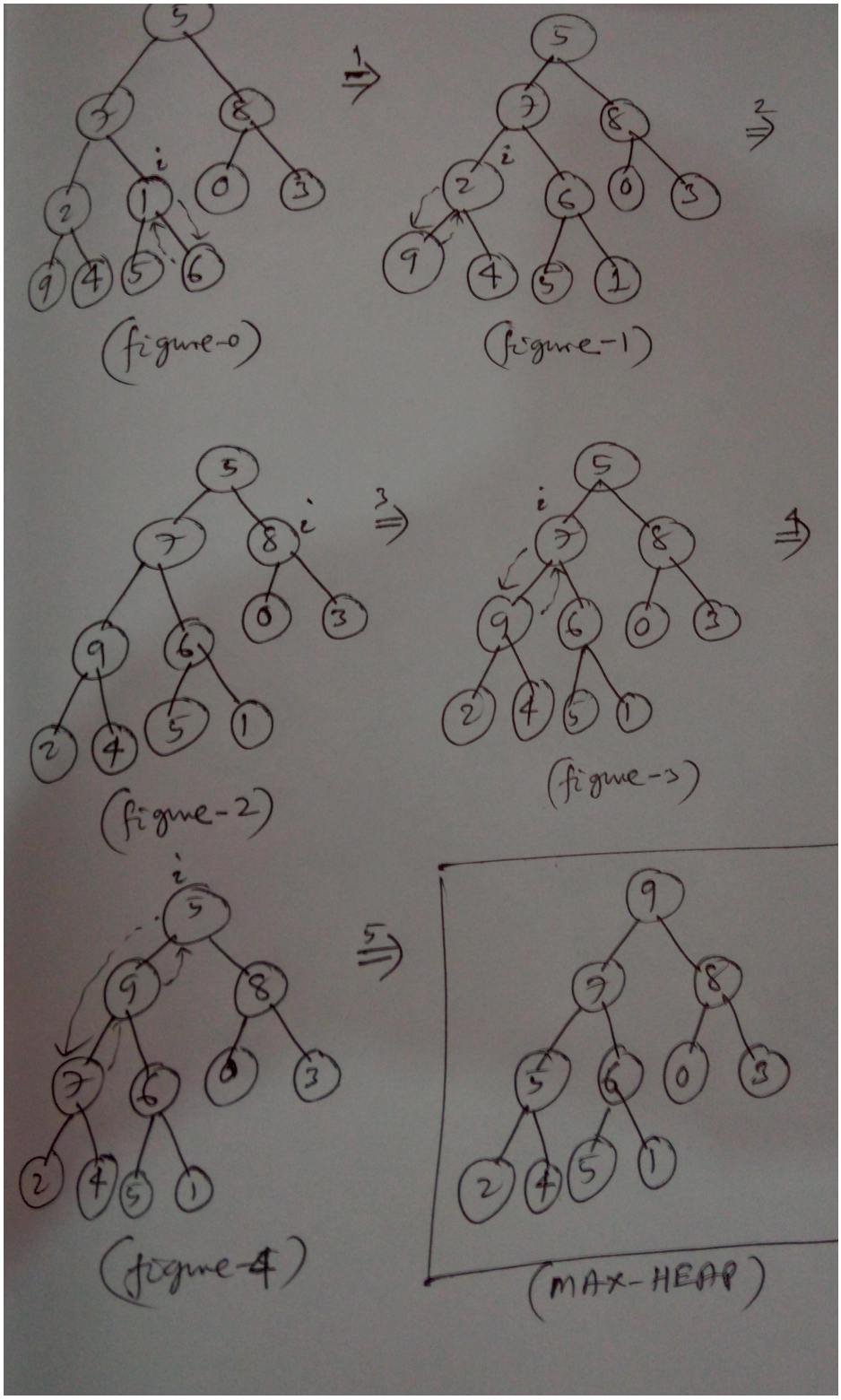
Time Complexity of BUILD-MAX-HEAP

Simple bound: O(n) calls to MAX-HEAPIFY, each of which takes O(log n) time ⇒ O(n log n).

Tighter analysis: T(n) = O(n)

Description of algorithm step by step on given array A = {5, 7, 8, 2, 1, 0, 3, 9, 4, 5, 6} to build a MAX-HEAP

i denotes the index of the node where MAX-HEAPIFY(i) is applied.



3.9 Write the Algorithm for the procedure HEAP-EXTRACT-MAX(A), where the procedure removes and returns the element of MAX-HEAP with largest key. Illustrate the operation of HEAP-EXTRACT-MAX on the Heap A={1, 3, 2, 6, 4, 5, 7, 8, 9}

(2017-MID-REG-KIIT-UNIV)

Solution

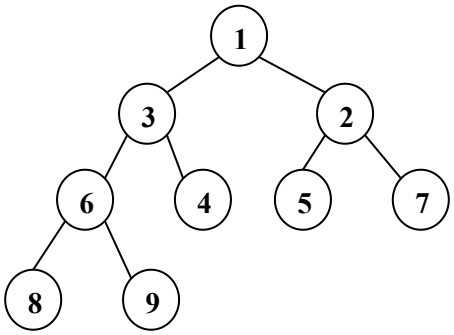
/*Algorithm to remove and returns the largest element of max-heap A*/

```
HEAP-EXTRACT-MAX(A, n)
{
     if n < 1
        then error-heap-underflow"
     max ← A[1]
    A[1] ← A[n]
    MAX-HEAPIFY(A, 1, n – 1)  //remakes heap
    return max
}
```

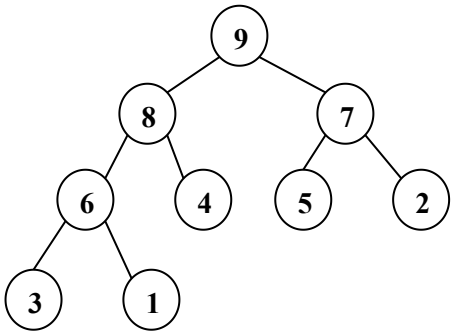
Illustrate of operation of HEAP-EXTRACT-MAX on the Heap A={1, 3, 2, 6, 4, 5, 7, 8, 9}

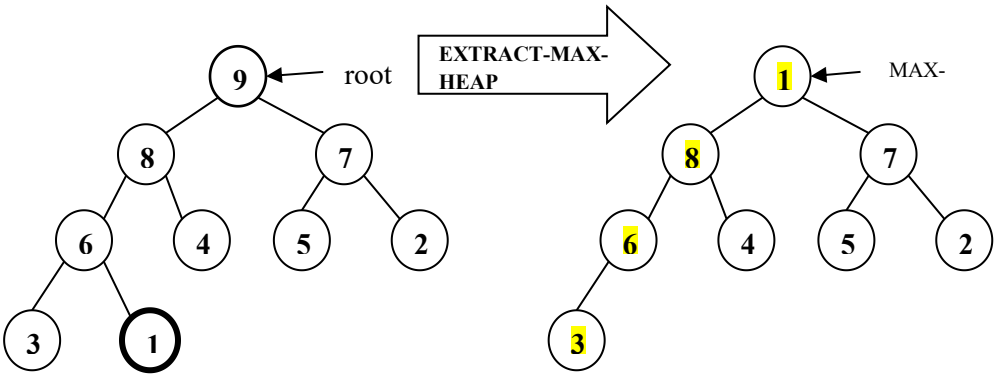
- The given heap is not a max-heap, it is a min-heap.
- First convert the given min-heap to max-heap, then apply the above algorithm.
- Method: Apply MAX-HEAPIFY(A, i, n) for i=n/2 down to 1.

Given Min-Heap

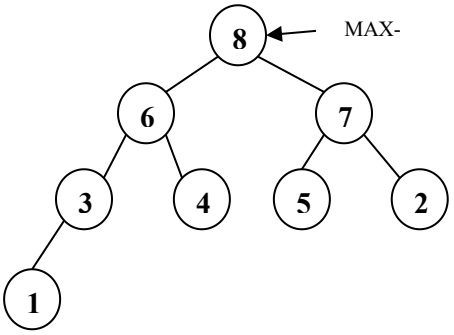


Max-Heap after conversion





(Now re-building the heap)



(Final heap after the operation of EXTRACT-MAX-HEAP)

After the operation of HEAP-EXTRACT-MAX on the Heap A={1, 3, 2, 6, 4, 5, 7, 8, 9}, the algorithm will return the largest value as 9 and the array (max-heap) becomes A={8, 6, 7, 3, 4, 5, 2, 1}

3.10 What is the effect of calling MIN-HEAPIFY(A, i) for i > size[A]/2?
(2020-MID-REG-KIIT-UNIV)

Solution
No effect. All nodes at index i > size[A]/2 are leaves.

3.11 What is the effect of calling MAX-HEAPIFY(A, i) for i > size[A]/2?
(2020-MID-REG-KIIT-UNIV)

Solution
No effect. All nodes at index i > size[A]/2 are leaves.

3.12 Where in a min-heap might the largest element reside, assuming that all elements are distinct?
(2020-MID-REG-KIIT-UNIV)

Solution
The largest element must be a leaf node.

3.13 Where in a max-heap might the smallest element reside, assuming that all elements are distinct?
(2020-MID-REG-KIIT-UNIV)

Solution
The smallest element must be a leaf node.

3.14 Write HEAPIFY() procedure and derive its time complexity. The elements of a heap structure are given as < 21, 1, 17, 8, 9, 6, 7, 4, 3, 8, 5 >. Find the node i, where the procedure HEAPIFY(i) should be applied to covert the given sequence into a max-heap. Show all the steps for performing HEAPIFY(i) operation on the above sequence.

(2020-MID-REG-KIIT-UNIV)

Solution

Max-Heapify Procedure

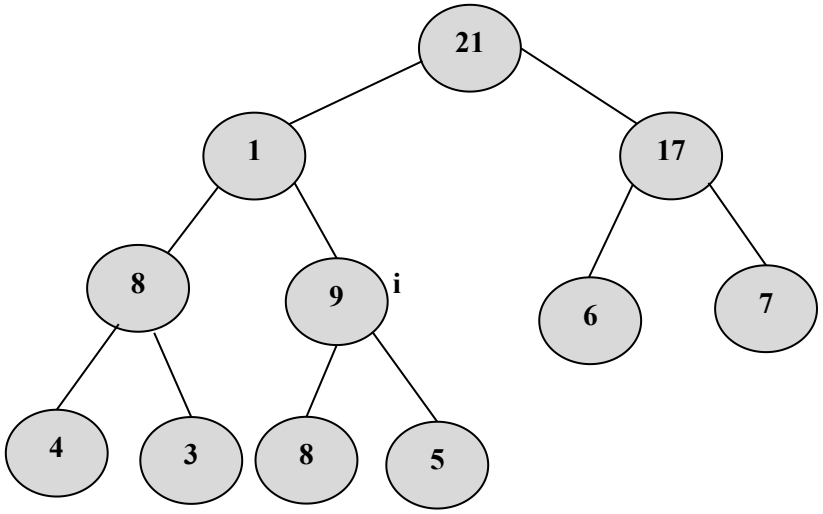
/***Max-Heapify :** Given a tree that is a heap except for node i, Max-Heapify function arranges node i and it's subtrees to satisfy the heap property.*/

```
MAX-HEAPIFY(A, n, i)
{
   l ← LEFT(i);
   r ← RIGHT(i);
   if  (l ≤ n and A[l] > A[i])
       largest = l;
   else
       largest = i;
   if  (r ≤ n and A[r] > A[largest])
       largest = r;
if (largest != i)
{
      A[i] ↔ largest]; // swaping
      MAX-HEAPIFY(A, n, largest);
   }
}
```

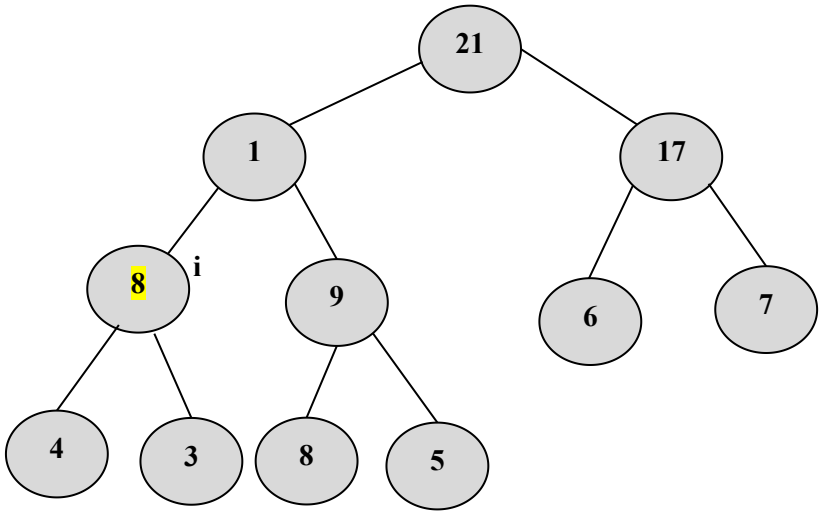
Building Max-Heap (Illustrate of operation of MAX-HEAPIFY on the array A={21, 1, 17, 8, 9, 6, 7, 4, 3, 8, 5}

- First arrange the elements of the array into a heap shape (complete binary tree structure).
- Apply MAX-HEAPIFY(A, n, i) for i=n/2 down to 1 for the above heap shape. In this case it is starting at index i=11/2=5

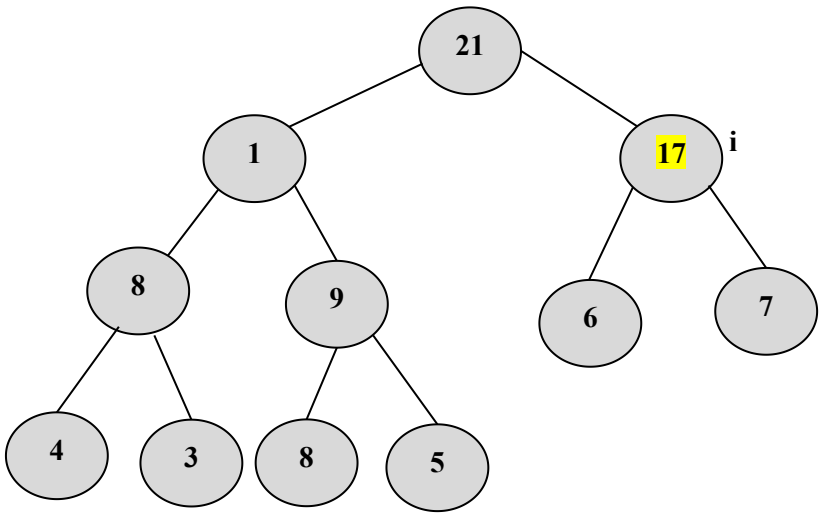
Heap Shape with given array A



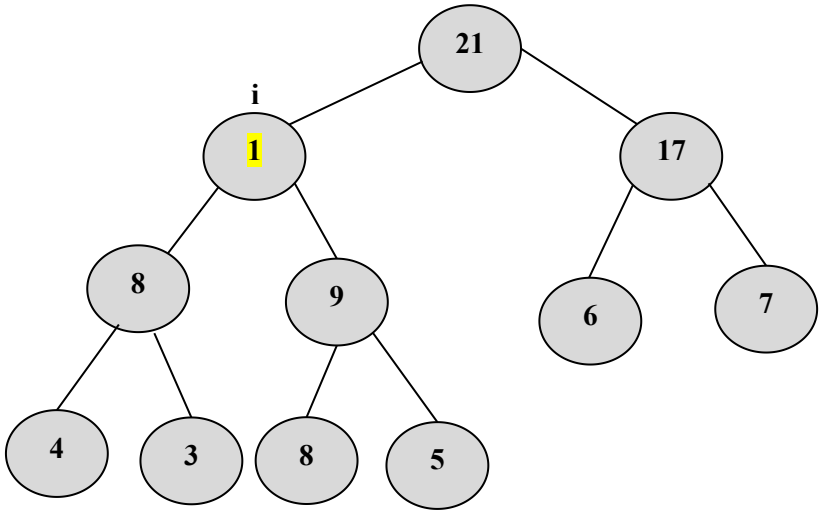
Heap after applying MAX-HEAPIFY(A, 11, 5) on above shape



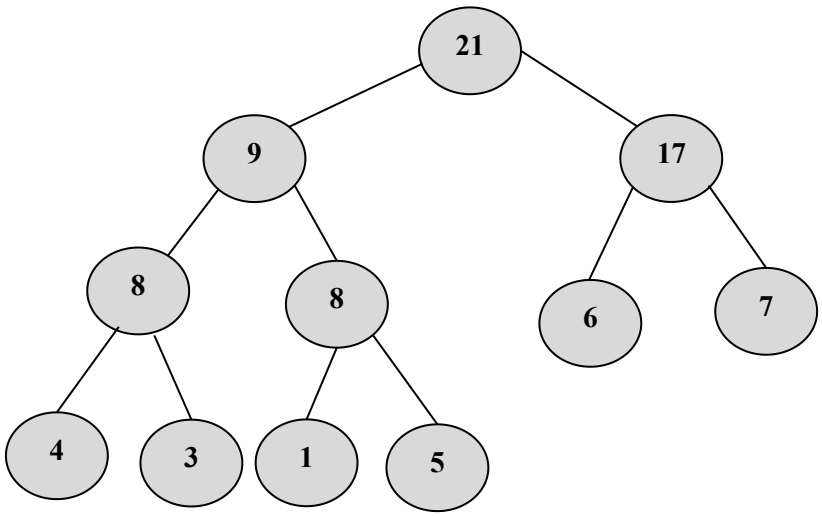
Heap after applying MAX-HEAPIFY(A, 11, 4) on above shape



Heap after applying MAX-HEAPIFY(A, 11, 3) on above shape



Heap after applying MAX-HEAPIFY(A, 11, 1) on above shape



(This is the Max Heap)

Time Complexity of Heapify

- Time complexity for MAX-HEAPIFY is O(log n)
- Time complexity for Building a Binary Heap is O(n)

3.15 What is the minimum time required to merge two max-heaps, each having n elements, into one max heap?
A. O(1) B.O(log n) C.O(nlog n) D.O(n)
(2021-MID-REG-KIIT-UNIV)

Answer
D

3.16 What will be the content of the array if 15 is inserted to an max-heap A={20, 10, 8, 6, 7, 5, 3, 3, 2}.
A.{20, 15, 8, 6, 10, 5, 3, 3, 2, 7} **B.{20, 15, 10, 5, 8, 6, 7, 3, 2, 3}**
C.{20, 10, 15, 5, 8, 6, 7, 3, 2, 3} **D.{20, 15, 6, 8, 10, 5, 3, 3, 2, 7}**
(2021-MID-REG-KIIT-UNIV)

Answer
A

3.17 What will be the content of the array if 15 is inserted to an max-heap A={20, 8, 10, 5, 3, 6, 7, 3, 2}.

A.{20, 15, 8, 6, 10, 5, 3, 3, 2, 7} **B.{20, 15, 10, 5, 8, 6, 7, 3, 2, 3}**
C.{20, 10, 15, 5, 8, 6, 7, 3, 2, 3} **D.{20, 15, 6, 8, 10, 5, 3, 3, 2, 7}**

(2021-MID-REG-KIIT-UNIV)

Answer
B

3.18 What will be the content of the array if 1 is inserted to an min-heap A={2, 3, 3, 5, 7, 6, 8, 10, 20}.

A.{1, 2, 3, 5, 3, 6, 8, 10, 7, 20} **B.{1, 2, 3, 3, 6, 5, 7, 10, 20, 8}**
C.{1, 2, 3, 5, 3, 6, 8, 10, 20, 7} **D.{1, 2, 3, 6, 3, 5, 7, 10, 20, 8}**

(2021-MID-REG-KIIT-UNIV)

Answer
C

3.19 What will be the content of the array if 1 is inserted to an min-heap A={2, 3, 3, 6, 8, 5, 7 10, 20}.

A.{1, 2, 3, 5, 3, 6, 8, 10, 7, 20} **B.{1, 2, 3, 3, 6, 5, 7, 10, 20, 8}**
C.{1, 2, 3, 5, 3, 6, 8, 10, 20, 7} **D.{1, 2, 3, 6, 3, 5, 7, 10, 20, 8}**

(2021-MID-REG-KIIT-UNIV)

Answer
D

3.20 a)Write an algorithm MAX-HEAP-CHANGE(A n, i, key) that rebuilds the n-element max-heap if the value at index i is changed to the value as key.
b)Apply this algorithm to the max-heap A={20, 15, 18, 10, 8, 12, 9, 6, 4, 8, 10} if at index 5 the value is changed to 25 and then at index 2 of modified heap the value is changed to 3. Assume root is at index 1. Show the process & result in max-heap diagram.

Solution

a)Sample Solution

Method-1 : By using known Coreman book algorithms	**Method-2 :**
MAX-HEAP-CHANGE(A n, i, key) { if (key<A[i]) { A[i] ← key; MAX-HEAPIFY(A, n, i) } else if(key>A[i]) HEAP-INCREASE-KEY (A, n, i, key) } Where, HEAP-INCREASE-KEY (A, n, i, key) { if (key < A[i]) {	MAX-HEAP-CHANGE(A n, i, key) { if (key<A[i]) { A[i] ← key; MAX-HEAPIFY(A, n, i) } else if(key>A[i]) { A[i] ← key; MAX-HEAPIFY-UP(A, n, i) } } Where, /*MAX-HEAPIFY-UP rearranges the nodes from index i max-possiblely to root to satisfy the max heap

```
      Print "error-new key is smaller than
current key";
       Exit;
     }
     A[i] ← key;
    while (i>1 and A[PARENT(i)] < A[i])
    {
        A[i] ↔ A[PARENT(i)];
        i ← PARENT(i);
    }
}
```

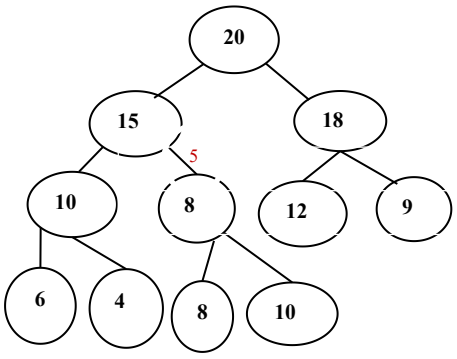
```
property.*/

MAX-HEAPIFY-UP(A, n, i)
{
 while (i>1 and A[PARENT(i)] < A[i])
 {
    A[i] ↔ A[PARENT(i)];
    i ← PARENT(i);
 }
}
```

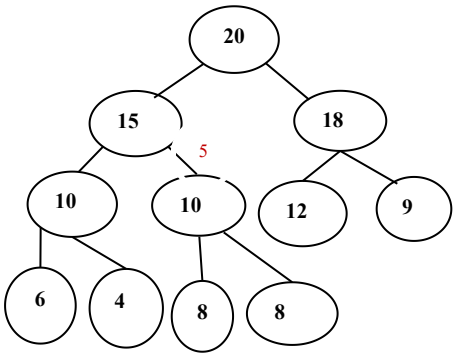
/***Max-Heapify :** Given a tree that is a max-heap of n-element array, except for node i, Max-Heapify function arranges node i and it's subtrees to satisfy the heap property.*/

```
MAX-HEAPIFY(A, n, i)
{
   l ← LEFT(i);
   r ← RIGHT(i);
   if  (l ≤ n and A[l] > A[i])
       largest = l;
   else
       largest = i;
   if  (r ≤ n and A[r] > A[largest])
       largest = r;
if (largest != i)
{
      A[i] ↔ A[largest]; // swaping
      MAX-HEAPIFY(A, n, largest);
   }
}
```

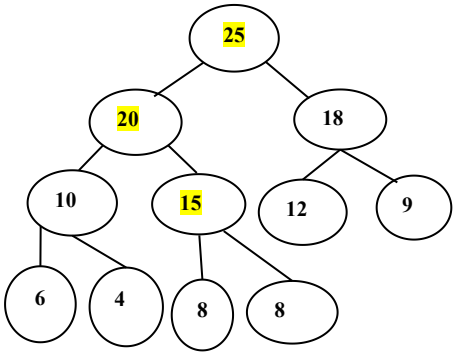
b) **The given Max Heap corresponds to array A={20, 15, 18, 10, 8, 12, 9, 6, 4, 8, 10} is as follows.**



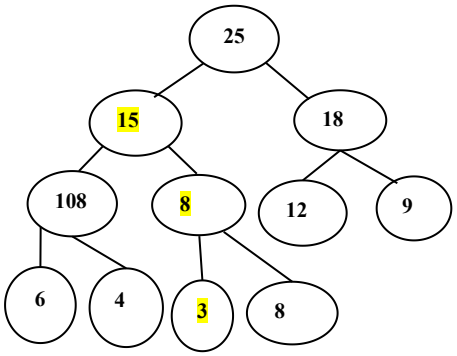
But value at index 5 does not satisfy the max-heap property. So applying MAX-HEAPIFY(A, 11, 5), the given heap is converted into max-heap as follows.



if at index 5 the value is changed to 25, the above max-heap becomes



if at index 2 the value is changed to 3, the above max-heap becomes



This is the final max-heap A={25, 15, 18, 10, 8, 12, 9, 6, 4, 3, 8}

4 Home Assignments

4.1 Write Min-Heap and Min-Priority algorithms with reference to Max-heap and Max-priority Queue algorithms already discussed in the class.

a) MIN-HEAPIFY(A, n, i)
b) BUILD-MIN-HEAP(A, n)
c) HEAP-SORT(A, n)
d) HEAP-MINIUM(A, n)
e) HEAP-EXTRACT-MIN(A, n)
f) HEAP-DECREASE-KEY(A, n, i, key)
g) MIN-HEAP-INSERT(A, n, key)

4.2 Build a max-heap and min-heap by taking individual digits of your roll numbers in sequence as the content of array A. If your roll number is 2005127, then assume array A={ 2, 0, 0, 5, 1, 2, 7}.

4.3 Apply Min-Priority-Queue algorithms to the min-heap constructed in step 2.

Any mistake/typo error found, mail to anil.swainfcs@kiit.ac.in or anilkumarswain@gmail.com mentioning the error with page number.

---------- THE END ----------