**Practical 13**

**Name: Shantanu Sethi**

**Roll no. 163**

**Aim: To implement Heap tree**

**Objectives:**

1. To create max heap tree

**Theory:**

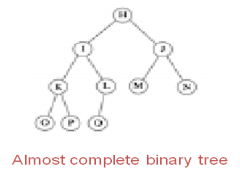
• Almost complete binary tree

– The tree is almost complete if all its levels, except possibly the last, has maximum number of nodes, and if all nodes at the last level appear as far left as possible.

– Each node that has a right child also has a left child.

– A node having a left child does not require to have a right child.

– Leaves are at level d or d-1



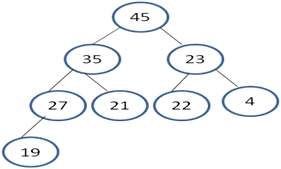
**Heap :**

A binary tree is a heap tree if it is an almost complete binary tree and has following properties:

· it is empty or

· the key in the root is larger than or equal to either child and both subtrees have the heap property .(max-heap)

( Heap property (max-heap): Key in the root is larger than or equal to either child)

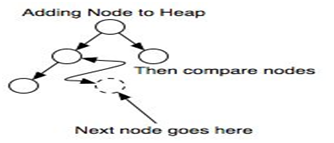


**Insertion in Heap tree:**

• The insertion algorithm consists of two steps

– Insert the new node in end(the new last node)

– Restore the heap-order property (Reheap Up/ Upheap)



**Reheap Up:**

• Push the new node upward, swapping with its parent until the new node reaches an acceptable location. i.e. one of the following conditions must be satisfied.

– The parent has a key that is >= new node, or

– The node reaches the root

**Algorithm:**

**Insert:**

Algorithm InsertHeap (heap <array of data type>, last<index>, data<datatype>)

Pre: heap is an array of data working as heap, last is index of last element in heap, data is data to be inserted in heap

Return : returns true if data inserted, false otherwise

1. if (heap full)
   1. return false
2. last = last+1
3. heap[last] = data
4. reheapUp (heap, last)
5. return true

**Reheap Up:**

Algorithm reheapUp (heap <array of data type>,newNode<index>)

Pre: heap is an array of data working as heap, newNode is index of new element inserted in heap

Return : new node placed at proper position

1. if (newNode not zero)
   1. parent = (newNode-1)/2
   2. if (heap[newNode] > heap[parent] )

1. swap(newNode, parent) //exchange elements at newNode and parent index

* 1. reheapUp (heap, parent)

1. return

**Program:**

**package** Heap;

**import** java.util.Scanner;

**public** **class** HeapTree {

Scanner sc = **new** Scanner(System.***in***);

**private** **int** n;

**private** **int**[] heap;

**private** **int** last;

**public** HeapTree() {

System.***out***.println("Enter the length of heap tree:");

last = -1;

}

**public** **void** get() {

n = sc.nextInt();

heap = **new** **int**[n];

**for**(**int** i = 0; i < n; i++) {

heap[i] = 0;

}

}

**public** **void** insertHeap() {

**if**(last >= n-1) {

System.***out***.println("\nHeap is full");

}

**else** {

System.***out***.println("\nEnter the element: ");

**int** data = sc.nextInt();

last++;

heap[last] = data;

reheapUp(last);

}

}

**private** **void** reheapUp(**int** newIndex) {

**if**(newIndex != 0) {

**int** parent = (newIndex - 1)/2;

**if**(heap[newIndex] >= heap[parent]) {

**int** temp = heap[newIndex];

heap[newIndex] = heap[parent];

heap[parent] = temp;

reheapUp(parent);

}

}

}

**public** **void** display() {

**for**(**int** i = 0; i <= last; i++) {

System.***out***.println(heap[i] + "\t");

}

System.***out***.println();

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("184 - Abhinav Singh");

HeapTree h = **new** HeapTree();

h.get();

System.***out***.println("\n1: Insert 2: Display 3: Exit");

**int** ch;

**do** {

System.***out***.print("\nEnter choice: ");

ch = sc.nextInt();

**switch** (ch) {

**case** 1:

h.insertHeap();

**break**;

**case** 2:

h.display();

**break**;

**case** 3:

System.***out***.println("Exiting...");

**break**;

**default**:

System.***out***.println("Invalid choice");

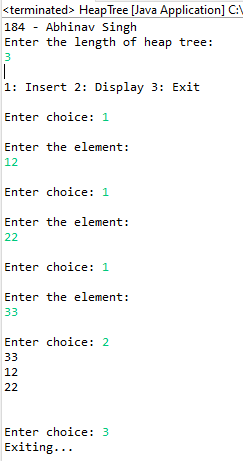
}

} **while** (ch != 3);

}

}

**OutPut:**



**Conclusion**: Successfully implemented max heap tree