Practice 02:

Implementation of Binary Heap

Code:

**package** test;

**public** **class** BinaryHeap {

**private** **int**[] Heap;

**private** **int** size;

**private** **int** maxsize;

**private** **static** **final** **int** ***FRONT*** = 1;

// Constructor of this class

**public** BinaryHeap(**int** maxsize)

{

**this**.maxsize = maxsize;

**this**.size = 0;

Heap = **new** **int**[**this**.maxsize + 1];

Heap[0] = Integer.***MIN\_VALUE***;

}

// Returning the position of the parent for the node currently

**private** **int** parent(**int** pos) { **return** pos / 2; }

// Returning the position of the left child for the node currently

**private** **int** leftChild(**int** pos) { **return** (2 \* pos); }

// Returning the position of the right child for the node currently

**private** **int** rightChild(**int** pos)

{

**return** (2 \* pos) + 1;

}

// Returning true if the passed node is a leaf node

**private** **boolean** isLeaf(**int** pos)

{

**if** (pos > (size / 2) && pos <= size) {

**return** **true**;

}

**return** **false**;

}

// To swap two nodes of the heap

**private** **void** swap(**int** fpos, **int** spos)

{

**int** tmp;

tmp = Heap[fpos];

Heap[fpos] = Heap[spos];

Heap[spos] = tmp;

}

// To heapify the node

**private** **void** minHeapify(**int** pos)

{

// If the node is a non-leaf node and greater than any of its child

**if** (!isLeaf(pos)) {

**if** (Heap[pos] > Heap[leftChild(pos)]

|| Heap[pos] > Heap[rightChild(pos)]) {

// Swap with the left child and heapify the left child

**if** (Heap[leftChild(pos)]

< Heap[rightChild(pos)]) {

swap(pos, leftChild(pos));

minHeapify(leftChild(pos));

}

// Swap with the right child and heapify the right child

**else** {

swap(pos, rightChild(pos));

minHeapify(rightChild(pos));

}

}

}

}

// To insert a node into the heap

**public** **void** insert(**int** element)

{

**if** (size >= maxsize) {

**return**;

}

Heap[++size] = element;

**int** current = size;

**while** (Heap[current] < Heap[parent(current)]) {

swap(current, parent(current));

current = parent(current);

}

}

// To print the contents of the heap

**public** **void** print()

{

**for** (**int** i = 1; i <= size / 2; i++) {

// Printing the parent and both childrens

System.***out***.print(

" PARENT : " + Heap[i]

+ " LEFT CHILD : " + Heap[2 \* i]

+ " RIGHT CHILD :" + Heap[2 \* i + 1]);

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

}

}

// To remove and return the minimum element from the heap

**public** **int** remove()

{

**int** popped = Heap[***FRONT***];

Heap[***FRONT***] = Heap[size--];

minHeapify(***FRONT***);

**return** popped;

}

// Main method

**public** **static** **void** main(String[] arg)

{

BinaryHeap minHeap = **new** BinaryHeap(15);

minHeap.insert(5);

minHeap.insert(3);

minHeap.insert(17);

minHeap.insert(10);

minHeap.insert(84);

minHeap.insert(19);

minHeap.insert(6);

minHeap.insert(22);

minHeap.insert(9);

// Print all elements of the heap

minHeap.print();

// Removing minimum value from above heap and printing it

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

}

}