**COMP 203 Lab 10**

**Array Implementation of Min Heap (100pt)**

A diagram of a tree

Description automatically generated

**SOLUTION**

public class MinHeap { //2pt

private int[] Heap;

private int sizeofHeap;

private int maxsizeofArray;

public MinHeap(int maxsizeofArray) {//3pt

this.maxsizeofArray = maxsizeofArray;

this.sizeofHeap = 0;

this.Heap = new int[maxsizeofArray + 1];

}

private int leftChild(int pIndex) {//5pt

return 2 \* pIndex + 1;

}

private int rightChild(int pIndex) {//5pt

return 2 \* pIndex + 2;

}

private int parent(int cIndex) {//5pt

return (cIndex - 1) / 2;

}

public void insert(int element) {//10pt

if (sizeofHeap >= maxsizeofArray) {

System.out.println("Heap is full. Cannot insert more elements.");

return;

}

Heap[sizeofHeap++] = element;

int current = sizeofHeap - 1;

while (current > 0 && Heap[current] < Heap[parent(current)]) {

swap(current, parent(current));

current = parent(current);

}

}

private void swap(int i, int j) {

int temp = Heap[i];

Heap[i] = Heap[j];

Heap[j] = temp;

}

private void upHeap(int index) {//10pt

while (index > 0 && Heap[index] < Heap[parent(index)]) {

swap(index, parent(index));

index = parent(index);

}

}

private void downHeap(int index) {//10pt

int smallest = index;

int left = leftChild(index);

int right = rightChild(index);

if (left < sizeofHeap && Heap[left] < Heap[smallest]) {

smallest = left;

}

if (right < sizeofHeap && Heap[right] < Heap[smallest]) {

smallest = right;

}

if (smallest != index) {

swap(index, smallest);

downHeap(smallest);

}

}

public int remove() {//10pt

if (sizeofHeap == 0) {

System.out.println("Heap is empty. Cannot remove element.");

return -1;

}

int removedValue = Heap[0];

Heap[0] = Heap[--sizeofHeap];

downHeap(0);

return removedValue;

}

public int min() {//5pt

if (sizeofHeap == 0) {

System.out.println("Heap is empty.");

return -1;

}

return Heap[0];

}

public int size() {//5pt

return sizeofHeap;

}

public void printMinHeap() {//10pt

for (int i = 0; i < sizeofHeap/2; i++) {

System.out.print(" PARENT : " + Heap[i] +

" LEFT CHILD : " + ((leftChild(i) < sizeofHeap) ? Heap[leftChild(i)] : "null") +

" RIGHT CHILD : " + ((rightChild(i) < sizeofHeap) ? Heap[rightChild(i)] : "null"));

System.out.println();

}

}

public static void main(String[] args) {

MinHeap minHeap = new MinHeap(10); //1pt

// Creating the given min Heap //9pt 9+1=10

minHeap.insert(1);

minHeap.insert(2);

minHeap.insert(3);

minHeap.insert(17);

minHeap.insert(19);

minHeap.insert(36);

minHeap.insert(7);

minHeap.insert(25);

minHeap.insert(100); //2pt (upheap , insert tests)

// Testing the methods

minHeap.printMinHeap();//3pt leftchild, rightchild, print tests

System.out.println("Min element: " + minHeap.min());//1pt

System.out.println("Heap size: " + minHeap.size());//1pt

int removedValue = minHeap.remove();//2pt min and downheap tests

System.out.println("Removed element: " + removedValue);

System.out.println("Min element: " + minHeap.min());//1pt

minHeap.printMinHeap();

}

}