// Chris clemmons

// CSC310

//Professor Kim

//3/27/19

//

package chris.clemmons.hw6;

import java.util.\*;

class pQueue{

private List<Integer> pq=new LinkedList<>(); // create pq as a global variable

//method to add elements to queue

void add(int x){

if(pq.isEmpty()){ //checks if queue is empty

pq.add(x);

}

else{

for (int i =0;i < pq.size();i++) {//checks size of queue

if (x <=pq.get(i)) {//checks if x is smaller than queue at i

pq.add(i, x);//calls method recursively

break;

}

else if(i ==pq.size()-1){ //checks if queue position is last

pq.add(x); //calls method recursively

break;

}

}

}

}

int min(){ //method to get the min value

return pq.get(0); //gets value at position 0

}

int removeMin(){ //method to remove min

int rem = pq.get(0); //temp variable for min

pq.remove(0);

return rem; // return temp variable

}

void sort(){ // method to sort priority queue

System.out.print("Priority Queue: "); // prints out operation

while(pq.size() >0){ //checks if pq is full

System.out.print(pq.get(0) +" "); // prints value at 0

pq.remove(0); // removes value

}

System.out.println();

}

}

class binHeap{

private List<Integer> heap = new ArrayList<>(); // create bin heap as a global variable

int parent(int x){ //parent node

return (x -1) /2;

}

int left(int x){ // left node

return 2\*x+ 1;

}

int right(int x){ // right node

return 2 \* x + 2;

}

boolean hasLeft(int x){ //checks for left node

return 2 \* x + 1 <heap.size() && heap.get(2 \*x +1) != null;

}

boolean hasRight(int x){ //checks for right node

return 2\*x+2< heap.size() && heap.get(2\*x+2) != null;

}

void insert(int x){ // method to insert into heap

heap.add(x); //call add

upheap(heap.size()-1); //increase heap size

}

void swap(int x, int y){ // swap method

Integer temp =heap.get(y); // sets up temp var

heap.set(y,heap.get(x)); // sets y to x

heap.set(x, temp); // sets x to y

}

int find\_min(){ //method to finr the min of a heap

return heap.get(0);//return value at position 0

}

int remove\_min(){ //method to remove value at 0

int temp = heap.get(0);// temp var with the value at 0

swap(0, heap.size()-1); //calls swap method

heap.remove(heap.size()-1);//remove value at the last postion

downheap(0);

return temp; // returns temp var

}

boolean is\_empty(){ // method to check whether a heap is empty

if(heap.size()==0){//check heap size

return true;

}

else{

return false;

}

}

int size(){// method to return size of heap

return heap.size();

}

void upheap(int pos){ // upheap method

if(pos !=0){//checks if at 0

if(heap.get(pos)< heap.get(parent(pos))){// checks if val of pos is smaller than parent(pos)

swap(pos, parent(pos)); // call swap method

upheap(parent(pos));//call upheap recursively

}

}

}

void downheap(int pos){ // downheap method

Integer childOne= null;// sets chlid 1 &2 as null

Integer childTwo = null;

if(hasLeft(pos))//sets childOne to left node

childOne =left(pos);

if(hasRight(pos)) { //sets childTwo to right node

childTwo =right(pos);

}

int smallest= -1;

if(childOne != null && childTwo!= null ) { // checks whether child 1 & 2 are empty

smallest =heap.get(childOne) < heap.get(childTwo) ? childOne : childTwo;

}

else if(childOne!= null ) //sets spallest to child One

smallest=childOne;

else if(childTwo!= null ) // sets smallest to child two

smallest=childTwo;

else

return;

if(heap.get(smallest) < heap.get(pos))

{

swap(pos,smallest); //calls swap method

downheap(smallest); //calls method recursively

}

}

}

public class ChrisClemmonsHw6 {

static void heap(int[] heap, int size, int i){

//sets up variables

int largest =i;

int l= 2\*i+1;

int r = 2\*i+2;

if(l<size && heap[l]> heap[largest]){//checks if heap at l is larger

largest = l;

}

if(r<size && heap[r] >heap[largest]){ //checks if heap at r is larger

largest= r;

}

if(largest!= i){// checks if largest is not equal to i

int swap = heap[i]; //sets up temp var

heap[i] =heap[largest]; //checks if i = largest

heap[largest] =swap;// repace largest with i

heap(heap,size, largest);// calls method recursively

}

}

static void heapSort(int[] heap){ // heapSort method

int n =heap.length;//sets n to length

for (int i=(n/2); i>= 0;i--) {// runs half of n

heap(heap, n, i); //calls heap

}

for (int i= n-1; i >=0; i--) { // runs length of heap

int temp= heap[0]; //sets up temp var

heap[0] =heap[i]; //replace pos 0

heap[i] =temp; // sets i to temp var

heap(heap, i, 0); // calls heapIt

}

System.out.print("Heapsort: ");

for (int i = 0; i < heap.length; i++) { // prints out heap by position

System.out.print(heap[i]+" ");

}

System.out.println();

}

public static void main(String[] args) {

pQueue p =new pQueue(); // create a new pQueue

int[] a = {1,6,3,7,5,9}; //create a

for (int i= 0; i < a.length;i++) {//populate p with a

p.add(a[i]);

}

p.sort(); // calls the sorting method in p

System.out.println("Bin Heap: ");

binHeap min = new binHeap(); // create a new binHeap

//insert into bin heap

min.insert(5);

min.insert(7);

min.insert(3);

min.insert(11);

//remove values from bin heap

System.out.println(min.remove\_min());

System.out.println(min.remove\_min());

System.out.println(min.remove\_min());

System.out.println(min.remove\_min());

int[] heap = {1,7,5 ,2,6, 4}; //create heap

heapSort(heap); //call heapSort with heap

}

}