Heapsort

Course *Algoritmer og Datastrukturer*
*DM*507

Troels Blicher Petersen - trpet15 -
troels@newtec.dk
Mark Jervelund - mjerv15 -
mark@jervelund.com

IMADA

April 10, 2016

Contents

Heapsort.java 1
PQHeap.java 1

Heapsort.java

```
import java.util.Scanner;

/**
 * Created by Mark jervelund          <Mark@jervelund.com>
<Mjerv15>
 *          & Troels Blicher Petersen <troels@newtec.dk>
<trpet15>
 */
public class Heapsort {
    public static void main(String[] args) {

        /**
         * Creates a new heap.
         */
        PQHeap pqHeap = new PQHeap();

        /**
         * scans incoming data.
         */
        Scanner sc = new Scanner(System.in);

        /**
         * While loop inserts elements in the heap from the scanner.
         */
        if (sc != null) {
            while (sc.hasNext()) {
                int num = sc.nextInt();
                pqHeap.insert(new Element(num, null));
            }
        }

        while (true) {
            try {
                int key = pqHeap.extractMin().key;
                System.out.print(key + "␣");
            } catch (ArrayIndexOutOfBoundsException e) {
                break;
            }
        }
    }
}
```

PQHeap.java

```
package Heapsort;

import java.util.ArrayList;
```

```
/**
 *
 * Created by Mark jervelund           <Mark@jervelund.com>
<Mjerv15>
 *          & Troels Blicher Petersen <troels@newtec.dk>
<trpet15> on 09-Mar-16.
 */
public class PQHeap implements EQ {

    private ArrayList<Element> A;
    private int n;
    private int left;
    private int right;
    private int smallest;

    /**
     * Creates a heap of without any elements.
     */
    public PQHeap() {
        A = new ArrayList<>();
    }

    /**
     * Makes a min heap.
     *
     * @param A An array holding the elements of the heap.
     * @param i An integer telling where to do "work" in the heap.
     */
    private void MinHeapify(ArrayList<Element> A, int i) {
        left = ((i+1)*2)-1;
        right = (i+1) *2;
        if (left <= n && A.get(left).key < A.get(i).key) {
            smallest = left;
        } else {
            smallest = i;
        }
        if (right <= n && A.get(right).key < A.get(smallest).key) {
            smallest = right;
        }
        if (smallest != i) {
            Exchange(i, smallest);
            MinHeapify(A, smallest);
        }
    }

    /**
     * Swaps two elements in the global element-array.
     *
     * @param index1  Position of the first element.
```

```
     * @param index2 Position of the second element.
     */
    private void Exchange(int index1, int index2) {
        Element temp = A.get(index1);
        A.set(index1,A.get(index2));
        A.set(index2,temp);
    }

    /**
     * Inserts an element in array A.
     *
     * @param A   An array holding the elements of the heap.
     * @param i   Where to insert the new element.
     * @param key The element to be inserted.
     */
    private void HeapIncreaseKey(ArrayList<Element> A, int i, Element key) {
        if (key.key >= A.get(i).key) {
            A.set(i,new Element(key.key, A.get(i).data));
            while (i > 0 && A.get(i / 2).key > A.get(i).key) {
                Exchange(i, i / 2);
                i = i / 2;
            }
        }
    }

    /**
     * Takes the minimum element out of the heap. When doing so
     * it afterwards returns this element.
     *
     * @return The minimum element of the heap.
     * @throws Exception if there is no elements to be extracted from
     *                   the array.
     */
    @Override
    public Element extractMin() {
        n = A.size() - 1;
        if (n < 0) {
            throw new ArrayIndexOutOfBoundsException();
        }
        Element min = A.get(0);
        A.set(0,A.get(n));
        MinHeapify(A, 0);
        A.remove(n);
        return min;
    }

    /**
     * Inserts an element in the heap and increases the heap size.
     * It uses HeapIncreaseKey to insert the element at an allowed
     * position.
```

```
     *
     * @param e The element to be inserted.
     */
    @Override
    public void insert(Element e) {
        n++;
        A.add(e);
        HeapIncreaseKey(A, n - 1, e);
    }

    /**
     * Method to return the heap.
     *
     * @return Retuns the heap for outside usage.
     * <p/>
     * CAUTION: This returns the current heap and thus this
     * heap is only sorted if the method Sort() has been
     * called beforehand.
     */
    public ArrayList<Element> getHeap() {
        return A;
    }
}
```