Noah Buchanan
Problem Set 3
Algorithms

September 3, 2020

1. Runtime of buildMinHeap(): $O(n)$ A very small amount of nodes will actually travel the lgn distance, what I mean by this is the distance to possibly travel down, decreases as the number of nodes on each row increases. At a height of 0 only 1 node could travel the entire lgn distance, at a height of 1 only 3 nodes could travel lgn-1 distance, the more nodes, the less distance they must cover. So the lgn distance tends to run closer to constant time especially the lower you get, therefore: $O(n * constant) = O(n)$

2. Runtime of heapifyUp(): $O(lgn)$ This method at its very worse can only travel log_3n distance because the furthest distance you can even get between two nodes is log_3n distance since the amount of nodes on each row is divided by 3 each time you move up.

3. Runtime of heapifyDown(): $O(lgn)$ This method at its very worse can also only travel log_3n for the exact same reasons as heapifyUp(), when traveling up or down a tree the max distance possible to cover is log(base=number of child nodes on each parent) n, so in this case log_3n.

4. Runtime of getSortedHeap(): $O(nlgn)$ This method does not run the same as buildMinHeap() because in the case of getSortedHeap, we never move down the heap, thus the potential to travel lgn distance is there through the entire loop 0-n, thus lgn time operation performed n times = $O(nlgn)$.

UAHeap class source code:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.Arrays;

/*  Student Name:        Noah Buchanan
```

```
 *  Username:               ua505               <--- this needs to be correct
 *  Date:                       September 1
 *  Class:          CS 3103 - Algorithm Design
 *  Filename:       UAHeap.java
 *  Description:    Implementation of a priority queue using a heap (by A. Mackey)
 */

public class UAHeap {

        static final int INITIAL_SIZE = 1; // just to declare the array with the default

        private int[] a; // This will be used to store the values of the heap

        int heapSize = 0;

        // Default, no-argument constructor
        public UAHeap() {
                this(INITIAL_SIZE);
        }

        // Construct a UAHeap with the size provided
        public UAHeap(int size) {
                a = new int[size];
        }

        // this method simply returns the array a, so do not modify this one
        public int[] getArray() {
                return this.a;
        }

        public static void main(String[] args) {
                if (args.length < 3) {
                        System.out.println("Invalid syntax:   java  UAHeap  inputfile  o
                        System.exit(100);
                }

                int inputSize = 0;

                try {
                        inputSize = Integer.parseInt(args[2]);
                } catch (Exception ex) {
                        System.out.println("Invalid input size");
                        System.exit(100);
                }

                UAHeap h = new UAHeap(inputSize);
                h.loadData(args[0]);

                if (h.getArray().length < 20) {
                        System.out.println("Start:  " + Arrays.toString(h.getArray()));
                        h.buildMinHeap();
                        System.out.println("Heap:   " + Arrays.toString(h.getArray()));
                        System.out.println("\n\n");
                        h.getSortedHeap();
                        h.saveDataToFile(args[1]);
                } else {
                        System.out.println("Step 1: Loading the array");
                        h.buildMinHeap();

                        System.out.println("Step 2: Running Heapsort");

                        System.out.println("\n\n");
                        h.getSortedHeap();
                        h.saveDataToFile(args[1]);


                        System.out.println("Complete. Saved output to " + args[1]);
                }
                System.out.println();
        }

        // this method should return the height of your heap (you need to calculate it)
        public int getHeight() {
                return (int) (Math.log10(heapSize) * 3.333333);
        }

        // Retrieves the min value from the heap
        public int getMinValue() {
                return a[0];
        }

        // Removes and returns the min value from the heap while preserving the heap
        // properties
        public int removeMinValue() {
                swap(0,heapSize-1);
                heapSize--;
                heapifyDown(0);
                return a[heapSize];
        }

        // Builds the heap structure by starting from ((int) n/2)
```

```
public void buildMinHeap() {
        for (int i = (int) heapSize / 3; i >= 0; i--) {
                heapifyDown(i);
        }
}

// Insert a value into the heap
public void insertValue(int value) {
        a[heapSize] = value;
        heapSize++;
        heapifyUp(heapSize-1,value);
}

// Returns the number of elements within the heap
public int size() {
        return heapSize;
}

// Reorganizes an element at the given index moving downward (if needed)
public void heapifyDown(int index) {
        int left = left(index);
        int middle = middle(index);
        int right = right(index);
        int smallest;

        if (left <= (heapSize-1) && a[left] < a[index]) {
                smallest = left;
        } else {
                smallest = index;
        }

        if (right <= (heapSize-1) && a[right] < a[smallest]) {
                smallest = right;
        }

        if (middle <= (heapSize-1) && a[middle] < a[smallest]) {
                smallest = middle;
        }

        if (smallest != index) {
                swap(index,smallest);
                heapifyDown(smallest);
        }

}
```

```
// Decreases the value at the specified index and moves it upward (if needed).
public void heapifyUp(int index, int value) {
        if(a[index] < a[parent(index)]) {
                swap(a[index],a[parent(index)]);
                heapifyUp(parent(index),value);
        }
}

// Loads data into the heap from a file (line-delimited)
public void loadData(String filename) {
        try {
                BufferedReader br = new BufferedReader(new FileReader(filename))

                String line = "";
                int i = 0;
                while ((line = br.readLine()) != null) {
                        a[i] = Integer.parseInt(line);
                        i++;
                        heapSize++;
                }
                br.close();
        } catch (Exception e) {
                System.out.println("load failed");
        }

}

// Writes the contents of the heap into the specified file (line-delimited)
public void saveDataToFile(String filename) {
        try {
                BufferedWriter bw = new BufferedWriter(new FileWriter(filename))

                for (int i = 0; i < a.length; i++) {
                        bw.write(a[i] + "");
                        bw.newLine();
                }
                bw.close();
        } catch (Exception e) {
                System.out.println("save failed");
        }

}

// Returns the values from the heap in ascending order (and saves it to the
// array a)
public int[] getSortedHeap() {
```

```
                int[] b = new int[heapSize-1];
                int i = 0;
                while(heapSize> 1) {
                        b[i] = removeMinValue();
                        i++;
                }
                a = b;
                return a;
        }

        /***********************************************************************
         * Location for additional methods needed to complete this problem set.
         ***********************************************************************/

        public void swap(int x, int y) {
                int hold = a[x];
                a[x] = a[y];
                a[y] = hold;
        }

        public int left(int i) {
                return 3 * i + 1;
        }

        public int middle(int i) {
                return 3 * i + 2;
        }

        public int right(int i ) {
                return 3 * i + 3;
        }

        public int parent(int i) {
                return (int) (i-1)/3;
        }

}
```