Assignment 1: Coding Question

Graded

Student
Tyler NGUYEN

Total Points
98 / 100 pts

Autograder Score
32.0 / 32.0

Passed Tests
Test extracting from { 1, 2, 3 } (1/1)
Whitebox test small heap (2/2)
Test empty() (1/1)
Test full() (1/1)
Whitebox test { 1, 2, 3 } (3/3)
Test randomly generated contents (8/8)
Whitebox test with heap size 10 (8/8)
Test inserting { 1, 2, 3 } (1/1)
Test fixed contents (4/4)
Test extractMin() exception (1/1)
Test insert() exception (1/1)
Test min() exception (1/1)

Question 2

Manual Grading **66** / 68 pts

– **0 pts** Correct

– **68 pts** Wrong

– **2 pts** Constructor argument validation:
You should guard against the passed size being negative and throw an exception beforehand.

Autograder Results

Test extracting from { 1, 2, 3 } (1/1)

Whitebox test small heap (2/2)

Test empty() (1/1)

Test full() (1/1)

Whitebox test { 1, 2, 3 } (3/3)

Test randomly generated contents (8/8)

Whitebox test with heap size 10 (8/8)

Test inserting { 1, 2, 3 } (1/1)

Test fixed contents (4/4)

Test extractMin() exception (1/1)

Test insert() exception (1/1)

Test min() exception (1/1)

Submitted Files

MinHeap.java Download

```java
/*
 * This Java source file was generated by the Gradle 'init' task.
 */
/*
author: Tyer Nguyen UCID: 30158563
date: October 4, 2023
description: MinHeap class that implements PriorityQueue and its methods
 */
package ca.ucalgary.cpsc331.a1;

public class MinHeap implements PriorityQueue{

	/**
	 * max of how much the heap can hold
	 */
	private int capacity;

	/**
	 * the current amount of keys that the heap is currently holding
	 */
	private int size;

	/**
	 * An array that represents the heaps data structure
	 */
	private int[] heap;

	/**
	 * constructor that intializes a new minHeap with the specified capacity value, as well as making it empty
	 * @param N
	 */
	public MinHeap (int N){
		capacity = N;
		size = 0;
		heap = new int[N];
	}

	/**
	 * checks if minHeap is empty
	 *
	 * @return true if the size is 0 which means mimHeap is empty else false
	 */
	@Override
	public boolean empty() {
		return size == 0;
	}

	/**
```

```
49          * checks if minHeap is full
50          *
51          * @return true if the size is capacity which means mimHeap is full else false
52          */
53         @Override
54         public boolean full() {
55             return size == capacity;
56         }
57
58         /**
59          * Inserts a new given key/value into the minHeap while perserving the minHeap property
60          * which is where for a given node, i the value must be less or equal to the values of
61          * its child nodes.
62          *
63          * @param key, which represents the key/value being inserted in the minHeap
64          * @throws RuntimeException when the heap is at capacity (full)
65          */
66         @Override
67         public void insert(int key) {
68             if(full()){ //checking if heap is at capacity (full)
69                 throw new RuntimeException("heap is at capacity");
70             }
71             heap[size] = key; //inserting the newly given key at the end of the heap
72             size++;
73             int curr = size - 1; //to get the current key that is being inserted
74             while(curr > 0){
75                 int parent = (curr-1)/2; //parent node calculation
76                 if(heap[curr]<heap[parent]){
77                     int temp = heap[curr];
78                     heap[curr] = heap[parent];  //swapping occurs
79                     heap[parent] = temp;
80                     curr = parent;
81                 }
82                 else{
83                     break;
84                 }
85             }
86             return;
87         }
88
89         /**
90          * extracts the minimum element from the minHeap while perserving the minHeap property
91          * which is where for a given node, i the value must be less or equal to the values of
92          * its child nodes.
93          *
94          * @return min, represents the minimum element that was extracted from the heap
95          * @throws RuntimeException when the heap is empty (0)
96          */
97         @Override
98         public int extractMin() {
99             if (empty()) { //checking if heap is empty (0)
100                 throw new RuntimeException("Heap is empty");
```

```
101             }
102             int min = heap[0]; //writing the minimum element into min which is at the root (heap[0])
103             heap[0] = heap[size-1]; //the root getting replaced by latest element in the heap
104             size--;
105             int curr = 0;
106             while (true) {
107                 int left = 2*curr+1; //calculating the left index +1 since java arrays start at 0
108                 int right = 2*curr+2; //calculating the right index + 2 since java arrays start at 0
109                 int smallest = curr;
110 
111                 //comparing with left child to determine if position correctly
112                 if (left < size && heap[left] < heap[smallest]) {
113                     smallest = left;
114                 }
115                 //comparing with right child to determine if position correctly
116                 if (right < size && heap[right] < heap[smallest]) {
117                     smallest = right;
118                 }
119                 if (smallest != curr) {
120                     int temp = heap[curr];
121                     heap[curr] = heap[smallest]; //swapping occurs
122                     heap[smallest] = temp;
123                     curr = smallest;
124                 }
125                 else{
126                     break;
127                 }
128             }
129             return min;
130         }
131 
132         /**
133          * gets minumum element from the minHeap without removing it
134          * @return min, represents the minimum element that was extracted from the heap
135          * @throws RuntimeException when the heap is empty (0)
136          */
137         @Override
138         public int min() {
139             if(empty()){ //checking if heap is empty (0)
140                 throw new RuntimeException("Heap is empty");
141             }
142             int min = heap[0];
143             return min;
144         }
145 
146         /**
147          * prints the full minHeap in a form of a string
148          *
149          * @return a representation of the whole minHeap
150          */
151         @Override
152         public String toString() {
```

```
153             //making a StringBuilder to build and represent the minHeap
154             StringBuilder sb = new StringBuilder();
155             sb.append("size = ").append(size).append("\n"); //printing the size of the heap
156             int levelSize = 1; //variable to keep track of the current level of the heap (tree)
157             int levelCount = 0;
158             for (int i = 0; i < size; i++) { //loop to iterate through the elements in the heap
159                 sb.append(heap[i]);
160                 levelCount++; //to track elements of the current level
161                 //adding spaces between elements that are the on same level
162                 if (levelCount < levelSize && i < size - 1) {
163                     sb.append(" ");
164                 }
165                 else{
166                     sb.append("\n");
167                     levelSize *= 2;
168                     levelCount = 0;
169                 }
170             }
171             return sb.toString();
172         }
173 }
```