Assignment 7: Part 2-Heap Sorting

Ryan Brinson

10/25/23

Output:

```
rbrinson2@DESKTOP-U8KJ4OP:~/Documents/CS3305/Assignments/A7$ java MinHeap.java

Randomized list:
[4, 53, -4, -5, 1, -3, 1, 2, -44, 0, 5, 3, 3]

Heap:
[[-44, -5, -3, -4, 0, 3, 1, 53, 2, 1, 5, 4, 3]]

Sorted List:
53 5 4 3 3 2 1 1 0 -3 -4 -5 -44
rbrinson2@DESKTOP-U8KJ4OP:~/Documents/CS3305/Assignments/A7$
```

Code:

```java
// Name: Ryan Brinson
// Class:CS 3305 W04
// Term: Fall 2023
//  Instructor:  Carla McManus
//  Assignment:  7-Part-2-Heaps

import java.util.*;


public class MinHeap {

    public static void main(String[] args) {
        // Create list to be sorted
        Integer[] list = {-44, -5, -3, 3, 3, 1, -4, 0, 1, 2, 4, 5, 53};
        Random rand = new Random();

        // RAndomize list
        for (int i = 0; i < list.length; i++) {
            int randomIndexToSwap = rand.nextInt(list.length);
            int temp = list[randomIndexToSwap];
            list[randomIndexToSwap] = list[i];
            list[i] = temp;
        }
        // Print the randomized list
        System.out.println("\nRandomized list:");
        System.out.println(Arrays.asList(list) + "\n");
```

```java
        // Sort the list again
        heapSort(list);

        // Print the final result
        System.out.println("Sorted List:");
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
        System.out.println();
    }

    // ----- Main Methods ----- //

    public static void heapSort(Integer[] list) {
        Heap heap = new Heap();

        // Add elements to the heap
        for (int i = 0; i < list.length; i++)
            heap.add(list[i]);

        // Print the current heap as an array
        System.out.println("Heap:");
        heap.printHeap();
        System.out.println();

        // Remove elements from the heap from the last element to the first
        for (int i = list.length - 1; i >= 0; i--)
            list[i] = heap.remove();
        }

}
// ----- Classes ----- //
class Heap {
    private ArrayList<Integer> list = new ArrayList<>();


    // ----- Constructors ----- //
    // Blank constructo
    public Heap(){}
    // Constructor that takes in list as argument
    public Heap(Integer[] objects){
        // Adds each element of the list using the add method
        for(int i = 0; i < objects.length; i++){
            add(objects[i]);
        }
    }
```

```java
// ----- Class Methods ----- //
// Print the current contents of list
public void printHeap(){
    System.out.println(Arrays.asList(list));
}

// Adds elements to the list
// Does preliminary sorting of the list
public void add(Integer newObject){
    list.add(newObject);
    int currInd = list.size() - 1;

    while (currInd > 0){
        // Find the parent to the current node
        int parentInd = (currInd - 1) / 2;

        // If the current node is smaller then the
        // parent node, swap them
        if (list.get(currInd) < list.get(parentInd)){
            Integer temp = list.get(currInd);
            list.set(currInd, list.get(parentInd));
            list.set(parentInd, temp);
        }
        else
            break;

        // Make your way up the heap
        currInd = parentInd;
    }
}

// Remove nodes from the heap
// This does the final sorting of the list
public Integer remove() {
    // Check if the list is empty
    if (list.size() == 0) return null;

    // Set removedObject to the parent node
    Integer removedObject = list.get(0);

    // Replace removedObject with the last element in the array
    list.set(0, list.get(list.size() - 1));
    // Remove the last element
    list.remove(list.size() - 1);
```

```
        int currentIndex = 0;
        while (currentIndex < list.size()) {
            // Get the two children nodes of the ith parent node
            int leftChildIndex = 2 * currentIndex + 1;
            int rightChildIndex = 2 * currentIndex + 2;

            // Check out of bounds
            if (leftChildIndex >= list.size()) break;

            // Check if the left or right child is a smaller value
            // whichever is smaller, set to minIndex
            int minIndex = leftChildIndex;
            if (rightChildIndex < list.size()) {
                if (list.get(rightChildIndex) < list.get(minIndex)) {
                    minIndex = rightChildIndex;
                }
            }

            // Swap if the current node is less than the min
            if (list.get(currentIndex) > list.get(minIndex)) {
                Integer temp = list.get(minIndex);
                list.set(minIndex, list.get(currentIndex));
                list.set(currentIndex, temp);
                currentIndex = minIndex;
            }
            else
                break;

        }

        return removedObject;
    }

    /** Get the number of nodes in the tree */
    public int getSize() {
        return list.size();
    }

    /** Return true if heap is empty */
    public boolean isEmpty() {
        return list.size() == 0;
    }
}
```