Homework Set 2

Chongyi Xu

April 12, 2017

Write Up

1. For each of the functions that you implemented given by the `PriorityQueue.java` interface, give the worst-case bigO notation for their runtime. Give the notation in terms of n, which should be the number of elements in the heap. This should be the tightest upper bound you can provide and should be a bigTheta bound if possible. Explain briefly how you came to each answer.

   - `isEmpty()` has $\Theta(1)$ runtime. Because `size` is stored and `isEmpty()` returns true if `size == 0`, returns false otherwise.
   - `size()` has $\Theta(1)$ runtime. Because `size` is stored and `size()` simply returns `size`.
   - `findMin()` has $\Theta(1)$ runtime. Because `findMin()` simply returns the first element in the heap array.
   - `insert()` has $O(logn)$ runtime. Because `insert()` **(1)** adds the element at the bottom and **(2)** percolates up to the correct position. In these two steps, **(1)** has $\Theta(1)$ and **(2)** has $O(logn)$. So `insert()` has $O(logn)$.
   - `deleteMin()` has $O(logn)$ runtime. Because `deleteMin()` **(1)** removes the top element, **(2)** moves the last element to the top and **(3)** percolates down. In these three steps, **(1)** has $\Theta(1)$, **(2)** has $O(logn)$ and **(3)** has $\Theta(1)$. So `deleteMin()` has $O(logn)$.
   - `changePriority()` has $O(n)$ runtime. Because it **(1)** finds the element($O(n)$), **(2)** changes the priority($\Theta(1)$) and **(3)** percolates up or down($O(logn)$). So `changePriority()` has $O(n)$.
   - `makeEmpty()` has $\Theta(1)$ runtime. Because it just simply recreate a new heap.

2. From your `testMany`, describe any helper test that you may have created. Additionally, identify the function you found most difficult to test. Explain why this function was most difficult. Finally, explain why BinaryHeap was easier or more difficult to test than the Queue from last week.

   I did not use any helper test. I found it most difficult to test `changePrioirty()` because the function itself returns *boolean* of whether the change is successful. But it can be hard to check if the heap property is still hold. I finally try `deleteMin()` and check if it equals string that a correct heap should have. And it works out.

   It is more difficult to test because BinaryHeap has to check if the resizing works as expected, while Queue does not need to resize.

3. Given the following segments of code, determine their asymptotic(bigTheta) runtimes in terms of n. For part c, only give the runtime of `mysteryThree(int n)`

```
(a) public void mysteryOne(int n) {
                int sum = 0;
                for (int i = n; i >= 0; i--) {
                        if ((i % 5) == 0) {
                                break;
                        } else {
                                for (int j = 1; j < n; j *= 2) {
                                        sum++;
                                }
                        }
                }
        }
```

   (a) has $O(n^2)$ runtime

```
(b)         public void mysteryTwo(int n) {
                int x = 0;
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < (n *(n + 1) / 3); j++) {
                                x += j;
                        }
```

```
                }
        }
```

(b) has $O(n^3)$ runtime

```
(c) public void mysteryThree(int n) {
                for (int i = 0; i < n; i++) {
                        printCats(n);
                }
        }

        public void printCats(int n) {
                for (int i = 0; i < n; i++) {
                        System.out.println("catsmoop");
                }
        }
```

(c) has $O(n^2)$ runtime