Analysis Document Assignment 3 Tim Dorny u0829896

1. Who is your programming partner? Which of you submitted the source code of your program?

My partner for this assignment was Tanner Marshall. I was the one to submit the source code for our assignment.

2. How often did you and your programming partner switch roles? Would you have preferred to switch less/more often? Why or why not?

We switched roles pretty sporadically. I think it worked out pretty well; following structured role patterns can be tiresome so we just went with the flow. I was happy with how it worked out.

3. Evaluate your programming partner. Do you plan to work with this person again?

Working with my partner was a great experience once again. We work really well together and efficiently completed the assignment without any issues. I would definitely want to work with Tanner again.

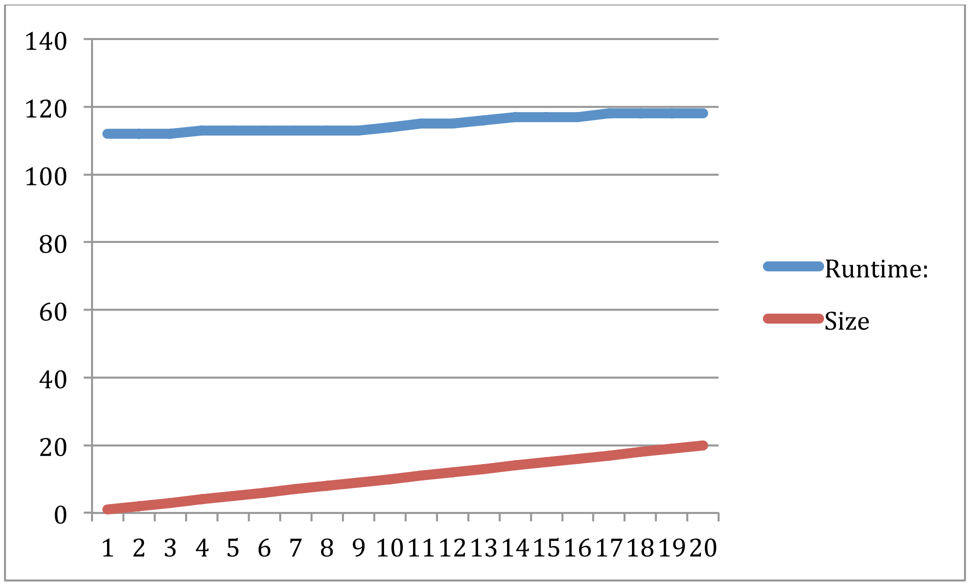
4. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development time.)

For starters, a Java list already implements many of the methods that we had to implement in this assignment. Working with a list would require considerably less coding on our end. It also allows the coveted ability to add an object directly to a specific index. This operation alone would have saved us a considerable amount of time and effort. I think using a List would be more efficient due to both the added methods and ease of use a List provides.

5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?

Since we are only looking for a single item and searching via binary search, I believe the Big-O behavior of this method would be log(N). It appears to be a pretty efficient method.

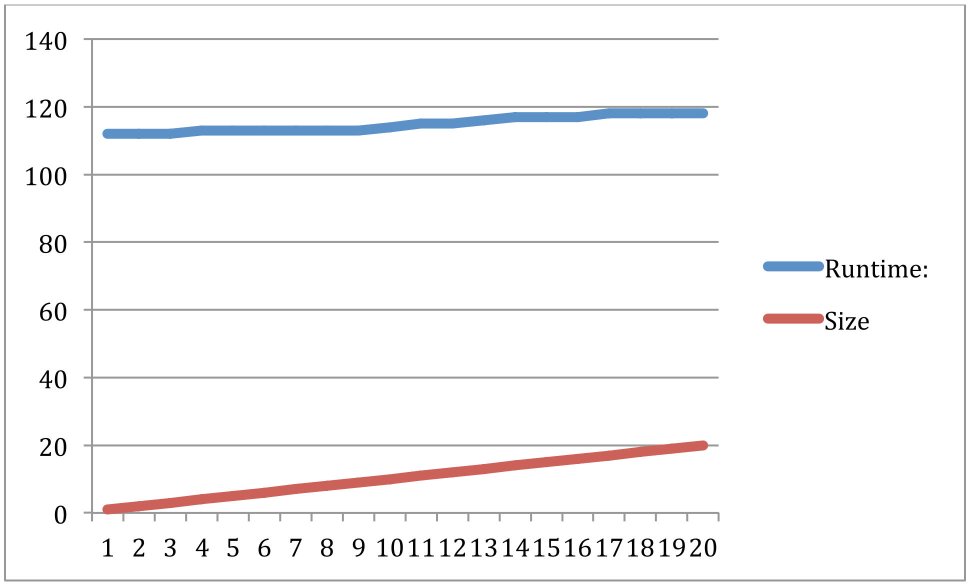
6. Plot the running time of MySortedSet's contains method for sets of sizes 100000 to 2000000 by steps of 100000. Use the timing techniques demonstrated in Lab 1. Be sure to choose a large enough value of timesToLoop to get a reasonable average of running times. Include your plot in your analysis document. Does the growth rate of these running times match the Big-oh behavior you predicted in question 5?



I used timesToLoop = 1000000. The timings were 11 ns for the first 12 tests, and 12 ns for the next 8. I know Log(N) should produce small increments, but I'm not sure if these are correct. I will paste the code for my timings at the end of the document.

`7. Consider your add method. For an element not already contained in the set, how long does it take to locate the correct position at which to insert the element? Create a plot of running times. Pay close attention to the problem size for which you are collecting running times. Beware that if you simply add N items, the size of the sorted set is always changing. A good strategy is to fill a sorted set with N items and time how long it takes to add one additional item. To do this repeatedly (i.e., timesToLoop), remove the item and add it again, being careful not to include the time required to call remove() in your total. In the worst-case, how much time does it take to locate the position to add an element (give your answer using Big-oh)?`

Our add method is almost identical to our contains method, so again our big-o is going to be Log(N).



This plot behaved very similarly to the previous one. Very small, almost negligible increases as the number of items in the set increased. I'm worried about the accuracy of these timings.

```
8. How many hours did you spend on this assignment?
```

We spent around 12 hours total on this assignment.

Here is the code I used for my timings on question 6:

```
public class Timing {

        public static void main(String[] args) {
          long startTime, midpointTime, stopTime;

          MySortedSet<String> hi = new MySortedSet<String>();
          int itemsAdded = 100000;
          for (int i = 0; i < itemsAdded; i++) {
                hi.add("hello");
          }

          // First, spin computing stuff until one second has gone by.
          // This allows this thread to stabilize.

          startTime = System.nanoTime();
          while (System.nanoTime() - startTime < 1000000000) { // empty block
          }

          // Now, run the test.

          //hi.add("Hi");
```

```
            long timesToLoop = 1000000;

            startTime = System.nanoTime();

            for (long i = 0; i < timesToLoop; i++) {
              hi.contains("hi");
            }

            midpointTime = System.nanoTime();

            // Run an empty loop to capture the cost of running the loop.

            for (long i = 0; i < timesToLoop; i++) { // empty block
            }

            stopTime = System.nanoTime();

            // Compute the time, subtract the cost of running the loop
            // from the cost of running the loop and computing square roots.
            // Average it over the number of runs.

            double averageTime = ((midpointTime - startTime) - (stopTime -
midpointTime))
                / timesToLoop;

            System.out.println("It takes exactly " + averageTime
                + " nanoseconds to compute the square roots of the "
                + " numbers 1..10.");
          }
        }
```