Natalie Cottrill-Yavornitzky

CS2420 – Assignment 3, MySortedSet

1 Programming Partner

Amit Athani is my programming partner.

I submitted the source code.

2 How Often Did Partners Switch?

We switched fairly often, but sometimes we were both so focused that we didn't pay attention to who was fulfilling which role. I'd estimate that overall, Amit was the driver about 70% of the time. This is his preferred role. When he has an idea to try, he finds it easier to type the logic instead of trying to explain it. I asked Amit to spend a bit more time telling me what he is thinking. This will allow him practice in getting his great ideas across and it will give me some more practice coding in the Java language. I do write many of the testing methods and I am helping Amit to learn more intensive testing with JUnit. Amit is helping me to better understand object oriented language. In the next project, I will suggest that we switch just once or twice per hour to see if we can help each other improve even more.

3 Evaluate Partner

Amit and I enjoy working together on the assignments. He likes to get the program done quickly, as do I. Yet, I like to ensure that we have covered as many of the code "breakage" cases as possible. We complement one another well as a team. We work fastest when I am a navigator and he is a driver. Amit knows Java language formatting very well, and I have several years' experience thinking through programming and poking holes in code from other languages. Amit is patient with my questioning our code to make sure it fulfilled all the requirements. We split the work well on this project. We plan on working together on the next project.

4 Java List vs Array

Yes, there would have been a difference in the efficiency of coding this project if we had used a Java List implementation, instead of an array, as the underlying data structure. In terms of coding, Lists are generally more flexible and faster to implement because there are more prewritten methods() in the List interface. Arrays do not have an interface and they have few methods associated directly with them, so manipulating arrays requires more lines of code.

Specifically, with regard to coding ease, Lists are easier and faster to implement than Arrays. For instance, Lists automatically track how many actual values they hold; Lists automatically expand when needed. They automatically grow bigger when an object is inserted and grow smaller when an object is removed. However, Arrays must be instructed (with additional lines of programming code) when it is time to get bigger. Furthermore, Arrays must be told how to get bigger and by how much. When using

Arrays, the programmer must manually create variable for the specific purpose of tracking last element in the Array that has a value. This variable must be manually incremented each time an element is added, and manually decremented each time an element is removed. Finally, when adding or removing an element to the inner elements of an Array, the programmer must write code to manually move each one of the values in the Array.

Finally, Lists objects have several very helpful methods that do common tasks, like toString() and equals(), which Array objects do not. Basically, working with Arrays requires more coding to do the same thing that Lists do with fewer lines of code and just as quickly.

```
Example:
System.out.println(list) = [list, of, object, values]          // toString Object<String>
System.out.println(array) = [Ljava.lang.String;@6f356496       // toString Array<String>
```

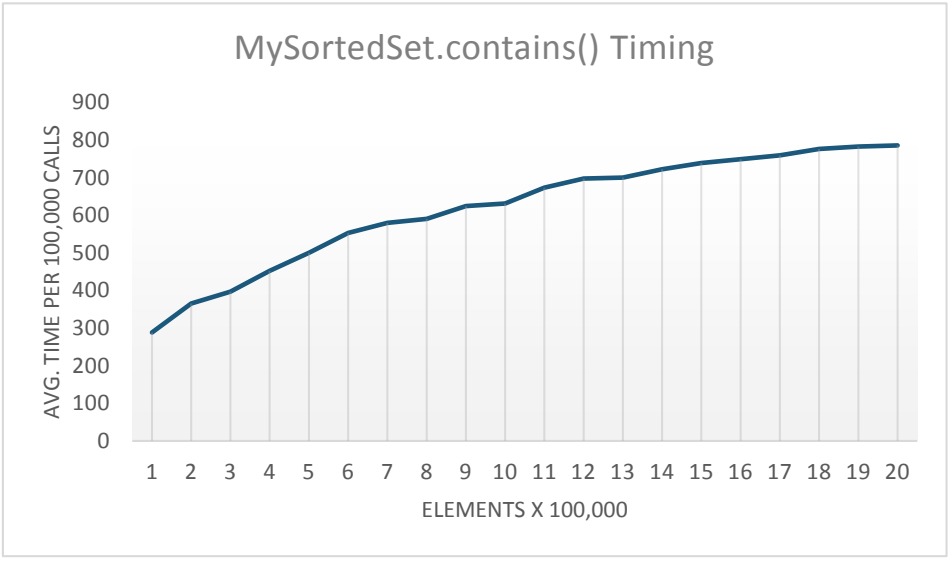
We have read several posts online and learned that there is not much performance cost in using a List over an Array, and that many times Lists are actually faster, e.g. http://eclipsesource.com/blogs/2014/04/11/3-good-reasons-to-avoid-arrays-in-java-interfaces/ and http://stackoverflow.com/questions/716597/array-or-list-in-java-which-is-faster

So, I think that using a Java List would have been more efficient, both in terms of development time and running time.

5 Big-O behavior of MySortedSet

Our MySortedSet.contains() method was driven by a binary sort. So, we expect it to be an O(log N) function and grow very slowly relative to the increasing number of elements.

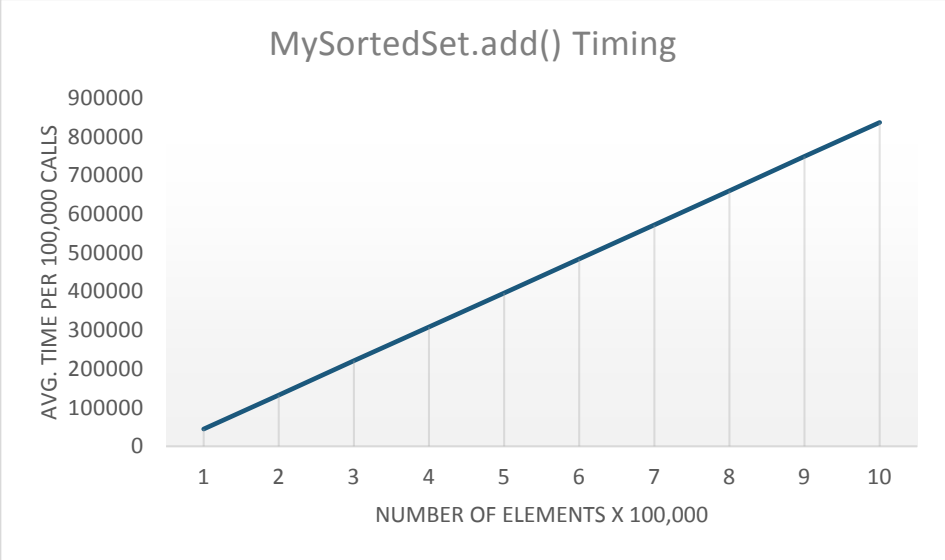
6 Plot the Contains Method



As expected, the contains() method grew slowly … even when the number of elements to search grew ever larger. This graph is not as flat as I expected However, it is notable that the average difference between 100,000 and 1,000,000 was 38 nanoseconds and the average difference between 1,100,000 and 2,000,000 was only 13 nanoseconds. So, I believe that it was the scaling of this chart makes the

difference look larger than expected. One of the limiting factors of the test was that we were timing on the school computers and they were busy. Sometimes we would obtain unusual readings, even though we were taking the average of 100,000 repetitions of the method call each time. A more controlled and dedicated environment may have helped to stabilize the timing.

7 Consider the Add Method



Our MySortedSet.add() method ended up being primarily an O(N) function with its timing. Every time we added 100,000 elements, the add() method took ~88,000 more nanoseconds to complete than it did in the previous iteration. In other words, the add() method timing was primarily dominated by the time it took to move the elements to the right each time. This linear "move" process ended up being the dominate function of the add() method.

8 How Many Hours?

We spent around 10 hours coding and a few hours writing up the analysis. An unusual amount of time (about 2 hours) was spent trying to interpret and implement the constructor instructions. The instructions for the first constructor said that it was to create a MySortedSet of Comparable objects. So, we constructed the base data array as a Comparable[] array. However, creating a base Comparable[] array clearly caused a ClassCastException() in the add() method when any non-Comparable class attempted to move through it. Initially, this is was confusing. After we obtained guidance from our very helpful TAs, we were able to proceed in good order. Approximately 6 hours was taken to implement JUnit tests and to test the timing.