Meng Jia U0878400

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

Brantly Walker, I submit the code.

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

2 Hours.

Yes, because it will make sure both of us familiar with code.

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

Yes. It's influenza season, everyone is sick…☹

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.)

Basic array would have more efficient (when consider very large elements).
Basic array can be accessed be memory. The ArrayList has been encapsulated, so check the memory address would be more complicated.
However, ArrayList implements the List Abstract Data Type using an array as its underlying implementation. Access speed is virtually identical to an array, with the additional advantages of being able to add and subtract elements to a List.
I would say it depends.

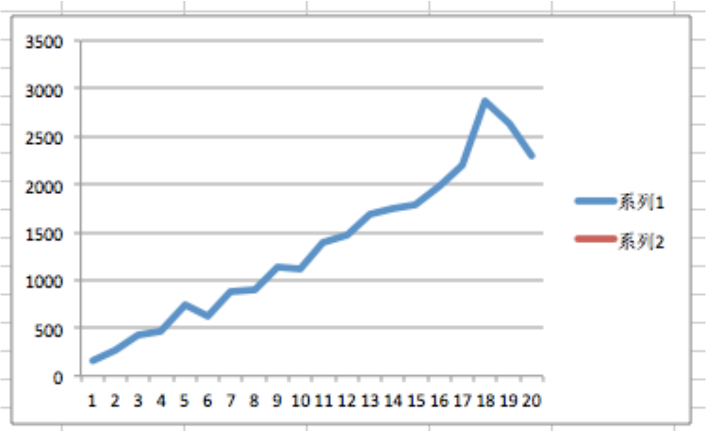
5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?

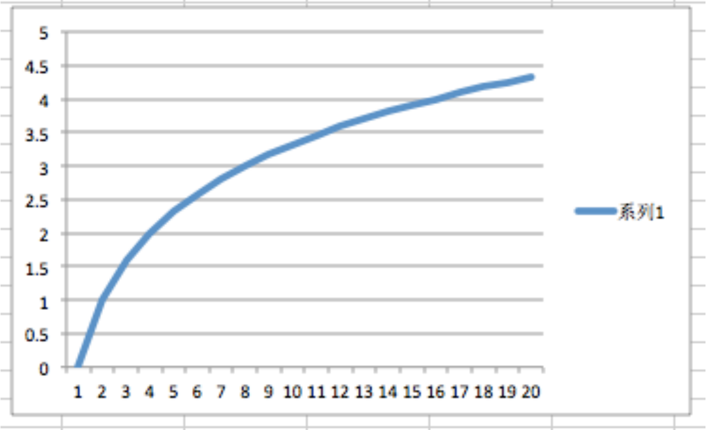
$O(\log n)$

Because the contain method is using binary search method to find if the item already include in it

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?

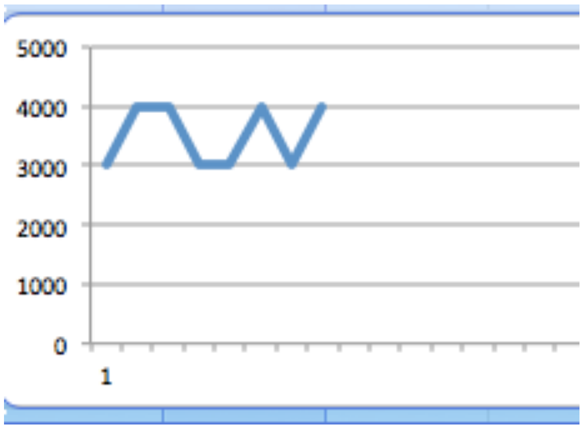
```
public void testContanRunTime() {

      MySortedSet<Integer> newSet1 = new MySortedSet<Integer>();

      for(int i=1;i<=20;i++){
         for(int j=0 ; j<100000*i; j++){
            test1.add(j);
         }
         System.out.println("------- test "+i+" begin -------");
         System.out.println("test size: "+test1.size());
         // start test contain function
         long startTime = System.nanoTime();
         for(int j=0;j<100000;j++){
            test1.contains(j);
         }
         long lastTime = System.nanoTime();
         System.out.println("time is: "+(lastTime-startTime));
             }
```





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

```
for(int i=0 ; i<2000000; i++){
         test1.add(i);
      }
      test1.remove(200000);
      test1.remove(200);
      test1.remove(499);
      test1.remove(100001);
      test1.remove(99876);
      test1.remove(998);
      test1.remove(199999);
      long startTime1 = System.nanoTime();
//    newSet1.add(200000);
//    newSet1.add(200);
//    newSet1.add(499);
//    newSet1.add(100001);
//    newSet1.add(99876);
//    newSet1.add(998);
      newSet1.add(199999);
      long lastTime1 = System.nanoTime();
      System.out.println(lastTime1-startTime1);
```



It take about 3500ns to locate the correct position at which to insert the element.
Complexity O(Logn+2n) where n = 200000

8. How many hours did you spend on this assignment?
About 18 hours