Jackson Murphy
u0647107
CS 2420
Assignment 3 Analysis
02/04/15

1. My partner is Jacob Luke. I submitted the source code.

2. We switched roles after every method or test method. (So about every 3 to 20 minutes, depending on how complicated the method was). This frequency of changing roles suited me just fine. Switching roles many times throughout the assignment helps me to stay engaged.

3. I enjoyed working with Jacob. At our first meeting, it was clear that he hadn't fully read over the assignment. But he picks things up really fast so it wasn't an issue. His coding style is a little different than mine (he took cs1410 at Utah St.) but this wasn't a problem either. I think we make a good pair. Our abilities are pretty evenly matched. I would like to work with him again.

4. Because a Java List's length is flexible but an array's length isn't, using a list would eliminate the need to manually grow our set. This means that we wouldn't have had to monitor the number of elements in our set versus the length of the array, nor double the array when it becomes full. Using a list would simplify our add() method (because this is where we grow our array) and would save us from having to increment or decrement the effective "size" of our set in the add() and remove() methods.

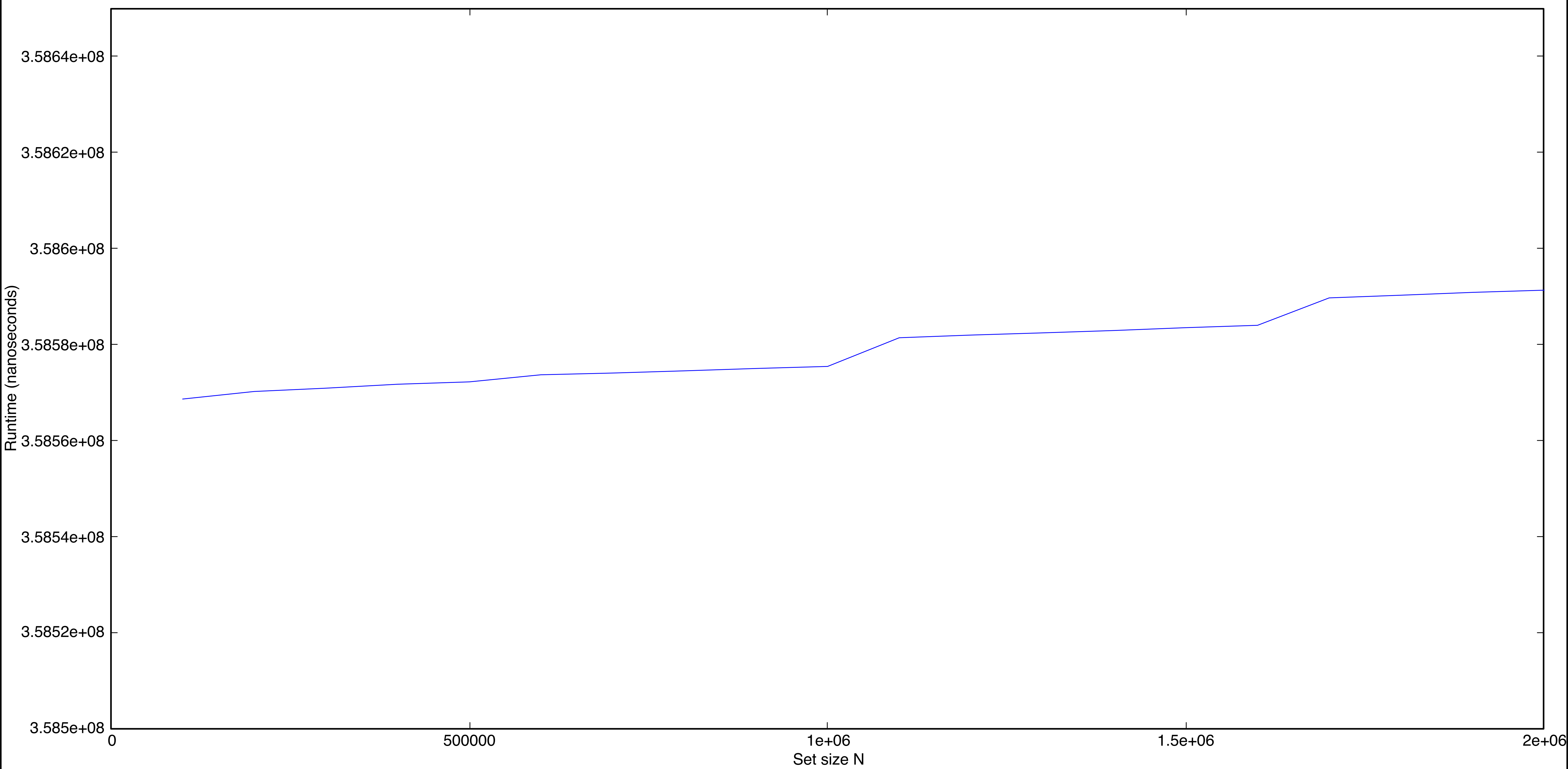
I suspect using a Java List would have been more efficient. It would have made our add() method easier to implement (thus decreasing our development time). And I think the difference in runtime would be negligible because under the hood the Java list probably uses the same strategy that we used for growing the array. The one downside is that a list would require a little more memory (object + its internal array). I suppose for a very, very large set size, this could become a consideration.

5. I'd expect the runtime for the contains() method to be O(logN). We implemented this method with the binary search algorithm, which uses the halving principle to achieve logarithmic complexity.

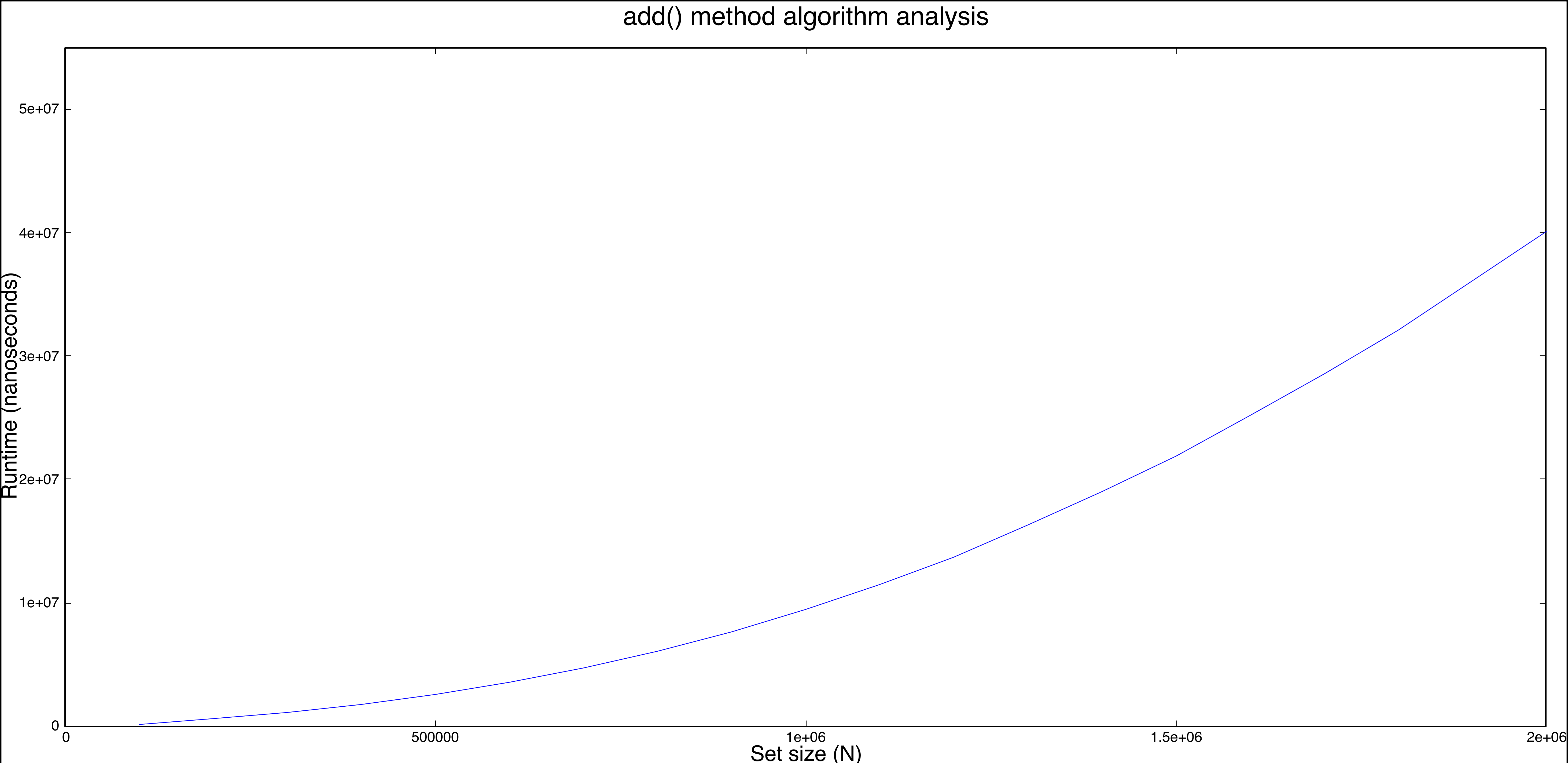
6. timesToLoop = 100,000. The plot is attached. Yes, the growth rate for these run times seem to match O(logN).

7. Our add() method uses binary search to find the position at which the new element should be inserted. So in the worst case, it takes O(logN) time to find the position. However, the growth rate for the add() method in its entirety is greater than O(logN)— it's more like O(N)— because it must also shift up to N items in order to make room for the new element. This is illustrated in the attached plot.

8. Time spent on this assignment: about 6 hours of pair programming, and 6 hours of preparation and analysis



contains() method algorithm analysis



add() method algorithm analysis

```java
package assignment3;

/**
 * Contains efficiency tests for the contains() and add() methods
 * of the MySortedSet<E> class
 *
 * @author Jackson Murphy
 */
public class Assgn3Analysis {

    public static void main(String[] args) {

        // / Analysis Question #6: Test whether the contains() method is O(logN)

        // Declare some variables
        MySortedSet<Integer> set = new MySortedSet<Integer>();
        long startTime, midpointTime, stopTime;
        long timesToLoop = 100_000;
        double avgTime;
        boolean answer = false;

        // Spin for a second to stabilize the thread
        startTime = System.nanoTime();
        while (System.nanoTime() - startTime < 1_000_000_000) {
        }

        // Run test for twenty different set sizes
        for (int i = 0; i < 20; i++) {

            // Add another 100,000 items to the set (or create an initial set of
            // size 100,000)
            for (int j = 1 + 100_000 * (i); j <= 100_000 * (i + 1); j++) {
                set.add(j); // The set is ordered. This shouldn't make a
                            // difference.
            }

            // System.out.println("Set size = " + set.size()); TEST

            // Set is now created. Next, time how long it takes for the
            // contains() operation to complete for an arbitrary integer

            // Choose a random integer to find in the set
            int someNumber = (int) (Math.random() * set.size());

            // Start the timer
            startTime = System.nanoTime();

            for (long k = 0; k < timesToLoop; k++) {
                // This is the method we're timing
                answer = set.contains(99999);
            }

            midpointTime = System.nanoTime();

            // The time just calculated above also include the time req'd to
            // initialize loop variables, etc. Let's account for that by
            // running and timing an empty loop of the same size

            for (long m = 0; m < timesToLoop; m++) {
            }

            stopTime = System.nanoTime();

            // Calculate/print the average time req'd for 1 contains() call
            avgTime = ((midpointTime = startTime) - (stopTime - midpointTime))
                    / timesToLoop;
            // System.out.println("Size step #" + i + ": " + avgTime +
            // " Number: " + someNumber);

            System.out.printf("%1.0f \n", avgTime);

        }

        // / Analysis Question #7: Find the growth rate of the add() method

        // Declare some variables
        MySortedSet<Integer> set2 = new MySortedSet<Integer>();
        long startTime2, stopTime2;
        long timesToLoop2 = 1_000; // add() takes longer than contains(), so
                                    // we'll only loop 1,000 times instead of
                                    // 100,000
        double avgTime2;
        long sumOfTimes = 0;

        // Spin for a second to stabilize the thread
        startTime2 = System.nanoTime();
        while (System.nanoTime() - startTime2 < 1_000_000_000) {
        }

        // Run test for twenty different set sizes
        for (int i = 0; i < 20; i++) {

            // Add another 100,000 items to the set (or create an initial set of
            // size 100,000)
            for (int j = 1 + 100_000 * (i); j <= 100_000 * (i + 1); j++) {
                set2.add(j); // The set is ordered. This shouldn't make a
                                // difference.
            }

            // System.out.println("Set size = " + set.size()); TEST

            // Set is now created. Next, time how long it takes for the
            // add() operation when given a new element

            // add a new element to the set many times to get a good average
            // time
            for (long k = 0; k < timesToLoop2; k++) {

                // Start the timer
                startTime2 = System.nanoTime();

                // Add a new element to the set
                set2.add(0);

                // Stop the timer and add the elapsed time to the running total
                stopTime2 = System.nanoTime();
                sumOfTimes += stopTime2 - startTime2;

                // remove the item from the set so that we can repeat the add()
                set2.remove(0);

            }

            // Calculate/print the average time req'd for 1 add() call
            avgTime2 = sumOfTimes / timesToLoop2;
            // System.out.println("Size step #" + i + ": " + avgTime +
            // " Number: " + someNumber);

            System.out.printf("%1.0f \n", avgTime2);

        }

    }
}

```