Gradey Culllins

1. Who is your programming partner? Which of you submitted the source code of your program? **Jet Vellinga, I submitted the source 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? **50/50 role switching. I always prefer to type because it is difficult to deeply understand code when I can't navigate exactly where I want, when I want, onscreen.**

3. Evaluate your programming partner. Do you plan to work with this person again? **My partner is satisfactory. I will probably work with this person 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.)
If using an array list instead of a basic array, the following points could be changed:
1) no need to dynamically grow our array when it is filled using the add method, array list does this automatically.
2) less conflict with casting e.g. object[] arr = (E[]) new Object[]();, instead could be : ArrayList<E> arr = new ArrayList<E>();.

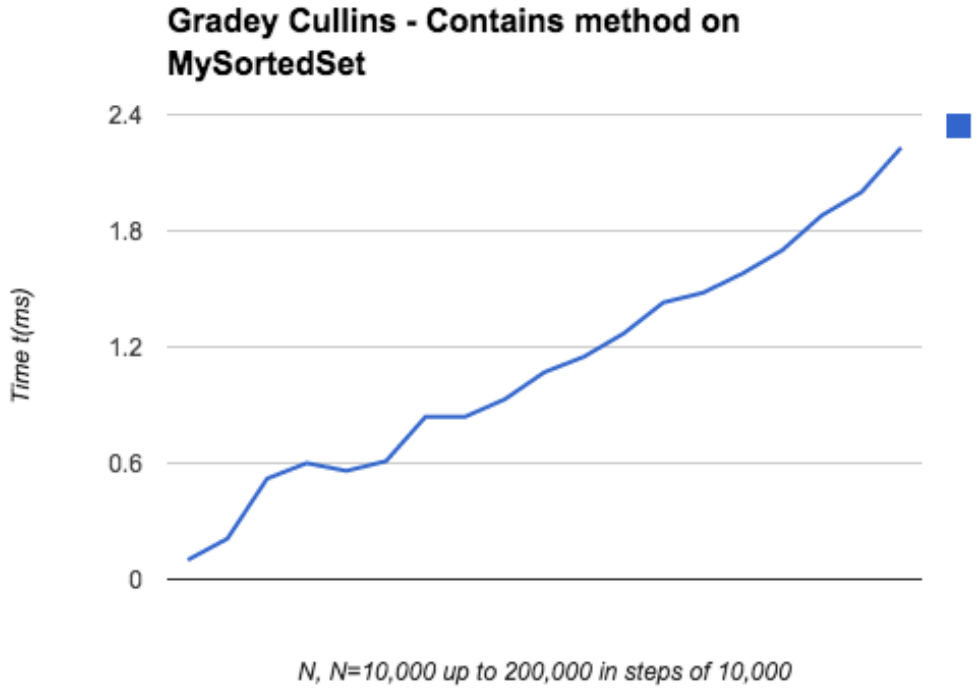
Backing the sorted set with Java's array list would have been highly beneficial in terms of program development time. Doing so would eliminate the need for my partner and I to create our own pseudo-dynamic array each time the array is filled to capacity; Array lists of the Collection interface take care of this automatically. Don't reinvent the wheel. Also, array lists already have add and remove methods which work in essentially the same fashion as the add and remove methods we implemented in our own class. So, again, this saves time on the development side assuming we were allowed to use such methods. Also, both array lists and our basic arrays are index-based data structures, meaning many of the algorithms we employ within our code would be similar if not the same because of the nature of index-based search, access, etc.

In terms of running time, many things would differ if using Java's array list and its api versus our custom class. For one,

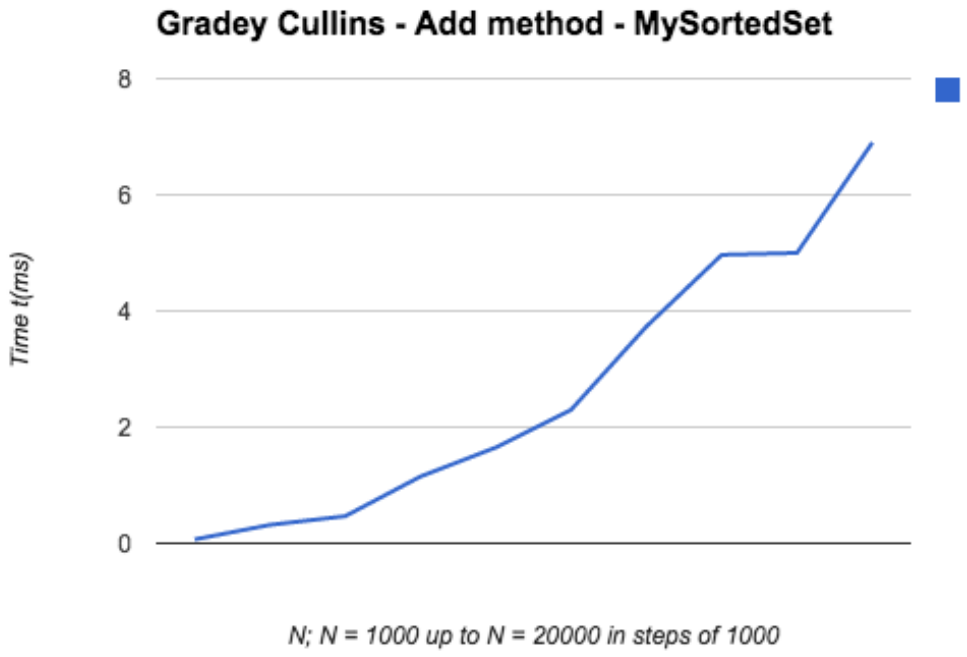
our class adds items in sorted order, array list adds items at the end of the list, meaning ours would be much slower. Assuming one is not using Java's array list add method and still using our custom add method, the running time should be similar, as, again, both data structures are index-based ones, and thus require searching through the list by means of an algorithm that hopefully avoids O(N) behavior e.g. binary search. The same explanation would hold for contains assuming one is using the method contained in our class and not Java's contains.

5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?
Because the contains method's only relevant body of code is that of the binary search implementation which has a O(logN) runtime, I would expect the big-O behavior of contains to be O(logN).

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? **Yes the growth rate matches behavior expected in question 5.**



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)? **It will take O(logN) to locate the correct position to insert an element not already contained in the set. In the worst case of this method, it takes O(logN) time to find the element. In this case, binary search must continue halving the sub array size until the sub array is of size 1. This operation is equivalent to continually dividing N size by 2 until only 1 item remains, or logN.**



8. How many hours did you spend on this assignment? **6 hours**