When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 30% of your Assignment 3 grade. Ensure that your analysis document addresses the following.

1. Who is your programming partner? Which of you submitted the source code of your program?
My partner is Charles Shoup. I will submit 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?
We work together at the CODE Lab from Tuesday through Thursday. However, we didn’t make any role to work, we just make method together, and figure out the error and fix them each works, and then, tell about it to the partner that I fix it.

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

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.)
I don’t need to create iterator, add, remove, contains method because the list already have those method itself. It will be more efficient because the list don’t need to check every single method about size or factors.

5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?
I expect that Big-O behavior would be N log N because the MySortedSet is used binary research method to calculate all position that should be stored.

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?

```
Time to perform 100000 adds is 7015609 nanoseconds
Time to perform 200000 adds is 13258693 nanoseconds
Time to perform 300000 adds is 20945829 nanoseconds
Time to perform 400000 adds is 28104399 nanoseconds
Time to perform 500000 adds is 36251701 nanoseconds
Time to perform 600000 adds is 43799371 nanoseconds
Time to perform 700000 adds is 51701315 nanoseconds
Time to perform 800000 adds is 58930461 nanoseconds
Time to perform 900000 adds is 67047553 nanoseconds
Time to perform 1000000 adds is 74783591 nanoseconds
Time to perform 1100000 adds is 83090157 nanoseconds
Time to perform 1200000 adds is 91903411 nanoseconds
Time to perform 1300000 adds is 99769420 nanoseconds
Time to perform 1400000 adds is 108392314 nanoseconds
Time to perform 1500000 adds is 116183125 nanoseconds
Time to perform 1600000 adds is 124643051 nanoseconds
Time to perform 1700000 adds is 132019563 nanoseconds
Time to perform 1800000 adds is 140219487 nanoseconds
Time to perform 1900000 adds is 148467289 nanoseconds
Time to perform 2000000 adds is 156097426 nanoseconds
```
It growths what I expected that is N log N.

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)?

```
Time to perform one ~random add on a 100000 member set is 13729110 nanoseconds
Time to perform one ~random add on a 200000 member set is 6570435 nanoseconds
Time to perform one ~random add on a 300000 member set is 4387630 nanoseconds
Time to perform one ~random add on a 400000 member set is 6040624 nanoseconds
Time to perform one ~random add on a 500000 member set is 4429089 nanoseconds
Time to perform one ~random add on a 600000 member set is 4325851 nanoseconds
Time to perform one ~random add on a 700000 member set is 6153364 nanoseconds
Time to perform one ~random add on a 800000 member set is 4159989 nanoseconds
Time to perform one ~random add on a 900000 member set is 4583681 nanoseconds
Time to perform one ~random add on a 1000000 member set is 4461361 nanoseconds
Time to perform one ~random add on a 1100000 member set is 4271400 nanoseconds
Time to perform one ~random add on a 1200000 member set is 4329329 nanoseconds
Time to perform one ~random add on a 1300000 member set is 4502064 nanoseconds
Time to perform one ~random add on a 1400000 member set is 4558277 nanoseconds
Time to perform one ~random add on a 1500000 member set is 4059114 nanoseconds
Time to perform one ~random add on a 1600000 member set is 4232706 nanoseconds
Time to perform one ~random add on a 1700000 member set is 4273207 nanoseconds
Time to perform one ~random add on a 1800000 member set is 4288265 nanoseconds
Time to perform one ~random add on a 1900000 member set is 4299634 nanoseconds
Time to perform one ~random add on a 2000000 member set is 4041719 nanoseconds
```
It shows that one random variable added has similar time even though as many items as added. Therefore, like what I expected for Big-O, when N increasing 2N, the Big-O increasing NlogN time.

8. How many hours did you spend on this assignment?
I spent 18 hours to do.

Programming partners are encouraged to collaborate on the answers to these questions. However, each partner must write and submit his/her own solutions.

Upload your document (.pdf only!) to the Assignment 3 page by 11:59pm on February 5.