1. Java Lists already take care of capacity, so it would negate any need for use to be aware of the fixed size array as we add items. However, it seems that we created a Java List-like structure since we take this into account. Adding and removing items to a Java List also might be easier since there are built in methods to do this. (There are more methods that you can use with a Java List in general). In terms of space, I would assume that a Java List would take up more space considering all these extra properties. In terms of run time, I assume that a Java List would take longer to run. In terms of program development time, it would take less time to implement. Using a basic array, the development time would be longer, but the memory/speed would potentially be less. As with most things we have talked about, I think both the running time and development time would depend on what kinds of lists we are using them for, and one might be better than the other in different scenarios.

2. I expect the Big-O notation of the BinarySearchSet's contains method to be O(logN). This is because the contains() method uses the binarySearch helper method, which has a Big-O notation of O(logN), so it will mirror this.

    - This is calculated because each iteration cuts the amount of the set to go through in half, from N until N/2x.

    - To solve for the relationship of N to X, N=2^x

    - Then, to find the value of x, we take the log base 2 of N, multiple by 2: x=log(base2)Nx2

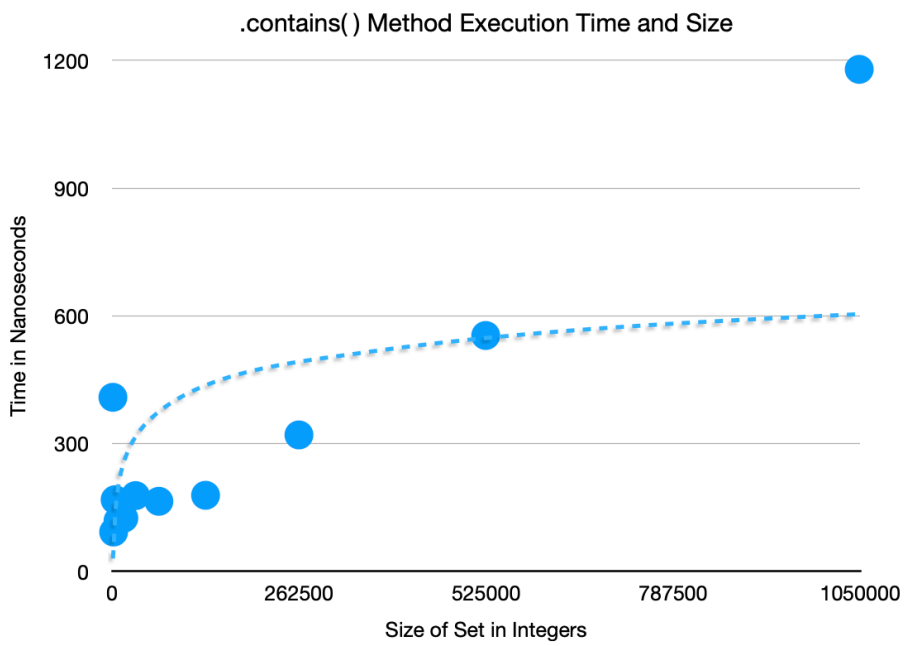
    - Expressed in Big-O notation, this is simplified to O(log N). The time complexity grows logarithmically with the size of the set, making it efficient in scenarios where

the set size can increase

3. The growth rate of this document increases more drastically over time than what I would have expected with a O(logN) formula.

contains_experiment

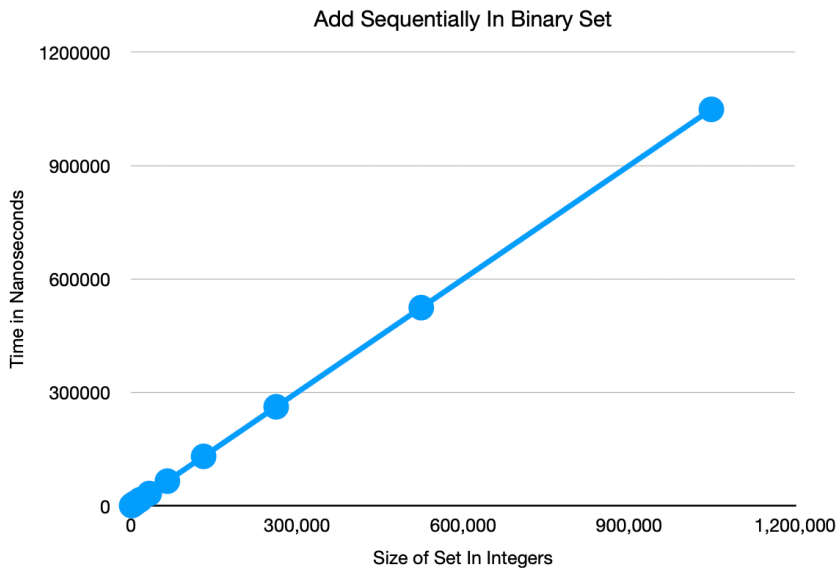
Size	Time
1024	408.76
2048	92.49
4096	168.39
8192	119.15
16384	125.0
32768	177.94
65536	164.59
131072	178.7
262144	320.41
524288	553.84
1048576	1178.74



4. The worst case scenario is O(N). The first chart is when you time .add() during the creation of the BinarySet with each sequential number, the second is when you time .add() with an element after removing it over and over, without timing the removal.

add_experiment, add all

Size	AvgTime
1024	49066.27
2048	39076.68
4096	111415.5
8192	182962.92
16384	377240.42
32768	785834.51
65536	1712166.68
131072	3585450.38
262144	9874084.64
524288	2.526203002E+07
1048576	5.722186417E+07



add_experiment2

Size	AvgTime
1024	269.16
2048	183.7
4096	288.37
8192	372.97
16384	643.34
32768	1335.01
65536	2743.73
131072	5630.02
262144	24448.34
524288	23704.2
1048576	47929.13