Hayden Walpole
U1210998
5/27/2024

Analysis Document — Assignment 3

1. My Partner for this assignment was Zijia (Harry) Xie, and they submitted our assignment on the GradeScope group.

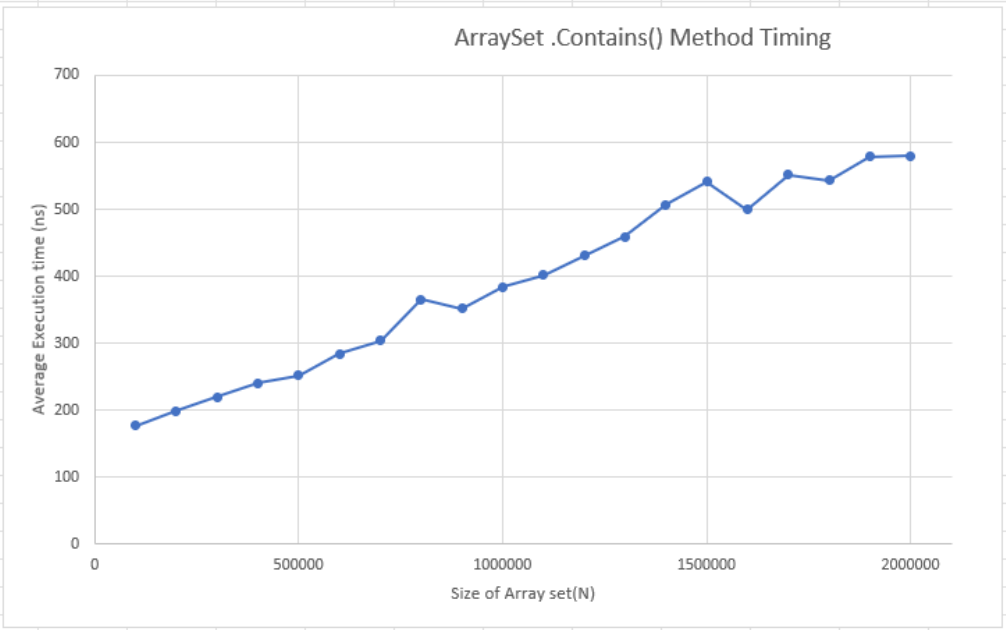
2. The pair programming experience went fairly well on this assignment. We wrote all the code together over the course of about 5 hours for the initial code and debugging, and 4 additional hours testing and timing. We would try to switch roles every 30 minutes, but occasionally lost track of time. I served as Navigator more often than Driver and want to do better at switching more in future assignments. For the next assignment I suggest using my computer instead of Harry's and hope this will make it more natural to drive more often. I enjoy working with Harry, but noticed at times we had some difficulties explaining our thought process when trying to debug the code together. We are going to do assignment 4 together, and I want focus on better two way communication while we are trying to solve issues with the code.

3. If we had backed the set with a Java ArrayList instead of a basic array, we would need to make a few implementation changes in order to make the code work properly. The first change would be to check for data types when adding items to the array, ArrayList cannot take primitive types like int, and would need to convert them to Integers instead. ArrayList also sizes automatically, in our code using a basic backing array we needed to check for indexes out of bounds and double the array size in our code before adding values to an already full array. ArrayList already has methods for many of the operations like add, remove, and checking for an empty array, while we needed to write them for a basic array in the assignment. ArrayList also adds each new item to the end of the list. For the purposes of this assignment without calling a sort method, we need to be able to insert new elements at specific indexes to keep the set in order. From what I understand using an ArrayList may have been faster during program development, but may have reduced runtime at execution due to the automatic changing size of the array when adding and deleting elements and would require us to sort the array before doing a binary search.

4. **contains method:** For this timing experiment we used a value of 1,000,000 times to loop with each N starting at N=100,000 to N=2,000,000 increasing N in steps of 100,000. After looking at the code I expect this method to have a Big-O = O(Log(N)) because it implements a binary search operation with no other loops. Shown below in Graph 1 is the data collected from timing analysis and the accompanying graph. Upon inspection using an analysis check, we see the data looks close to linear, but even closer to a logarithmic function. This is what we would expect, although we do see some

jumps of higher values in our data which continued to be present even as we increased our times to loop.

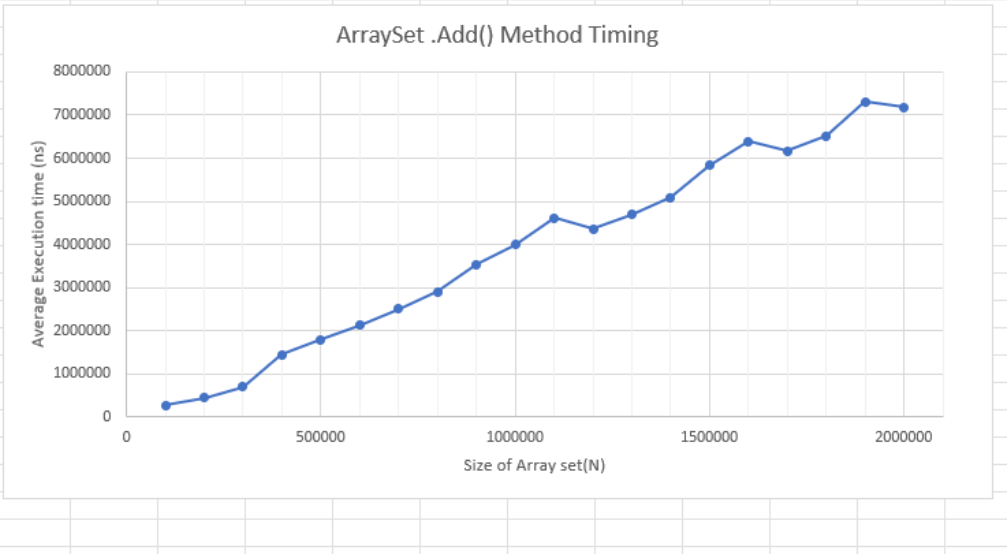
Size (N)	time(ns)
100000	177.4672
200000	199.035
300000	219.1784
400000	240.9578
500000	251.9565
600000	284.4304
700000	303.7524
800000	365.3814
900000	352.1588
1000000	384.5126
1100000	401.3507
1200000	431.4213
1300000	458.5956
1400000	506.7521
1500000	541.6679
1600000	499.497
1700000	551.2189
1800000	542.8902
1900000	578.8859
2000000	580.2314



Graph 1

5. **add Method:** For this timing experiment we used a value of 10,000 times to loop with each N starting at N=100,000 to N=2,000,000 increasing N in steps of 100,000. We expect the Big-O notation of this method to be O(N*log(N)). It first uses a binary search to find the location to insert the new element, and then must shift all the elements after the insertion point, which is an order O(N) operation. After analysis check, the growth rate of the add method seems to best fit O(N*log(N)) as we predicted.

Size (N)	time(ns)
100000	268271.6
200000	443753
300000	700957.9
400000	1449800
500000	1799187
600000	2118078
700000	2504413
800000	2905229
900000	3537637
1000000	3993146
1100000	4613605
1200000	4354100
1300000	4692937
1400000	5084508
1500000	5832962
1600000	6389148
1700000	6169104
1800000	6511035
1900000	7313403
2000000	7183623



Graph 2