Hayden Walpole
U1210998
6/18/2024

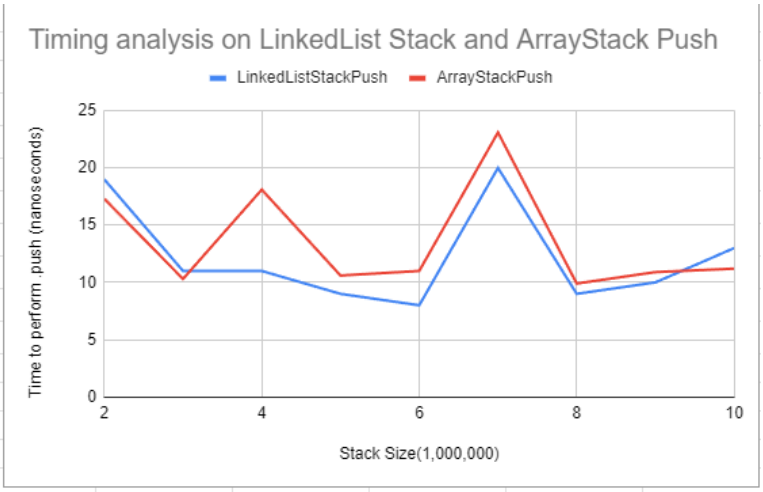
Analysis Document — Assignment 6

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 good for us. We wrote all the initial code together over the course of about 4 hours, and 9 additional hours debugging and writing our timing program. We would try to switch roles every 40 minutes during coding, but occasionally lost track of time. I was able to drive closer to 50% this time, and I noticed during debugging Harry was more willing to switch and try both our ideas to solve the problem. We both have different initial approaches to solving problems in the code, and it is beneficial to learn from each other, and find when the best time is to use each approach.

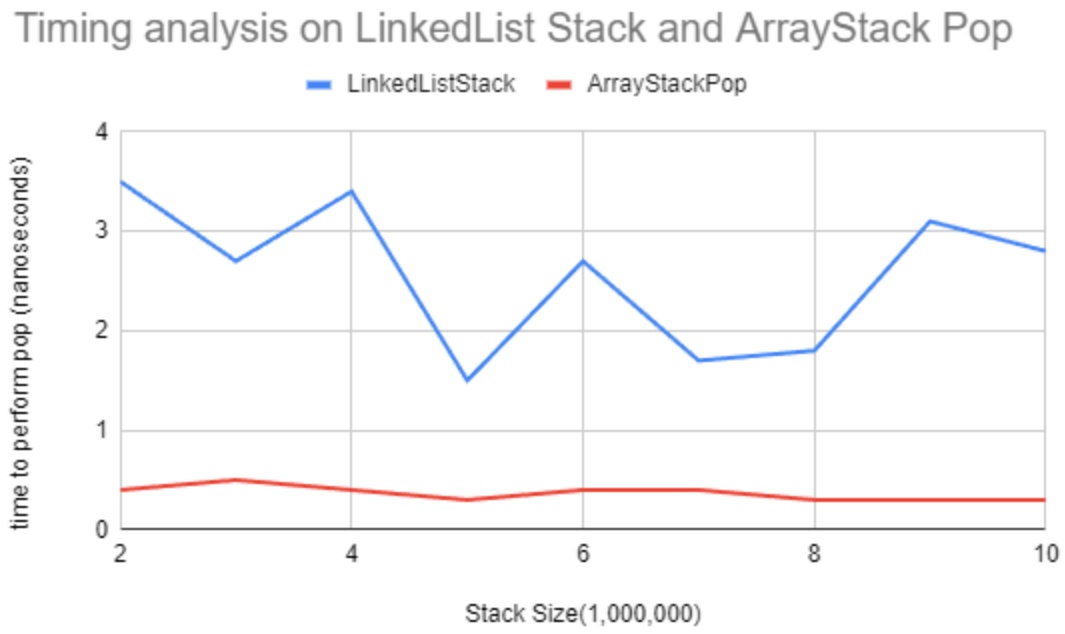
3a. To compare the Running time of the Push method on the ArrayStack and LinkedListStack classes we used a timing test using a stack size starting at 1,000,000 to 10,000,000 incrementing it by 1,000,000 each time. For each N value we first built up our stack then called the stack.push(i) method 1,000 times, and took the average. For this test we did not worry about accounting for the backing array/list growing a little each time we pushed a value into it because we ensured our size N was significantly larger than our times to loop number. Both Stacks have an expected behavior of O(1), which we see in Graph 1 below. This growth is because push is pushing to a stack it will push to the top of the stack and only update the head node, there are not indexes that need to be changed like if we add to the first element of an array. I also want to note that because the time is in tens of nanoseconds, deviations of only a couple nanoseconds cause large deviation. Average time to perform push for the linkedListStack was 12.22 nanoseconds, and ArrayStack had an average time of 13.6 nanoseconds. Making LinkedListStack faster at performing Push on a random index for our sample.

size(N)	LinkedListStac	ArrayStackPush
2000000	19	17.3
3000000	11	10.3
4000000	11	18.1
5000000	9	10.6
6000000	8	11
7000000	20	23.1
8000000	9	9.9
9000000	10	10.9
10000000	13	11.2
Average Time	12.22222222	13.6



Graph 1

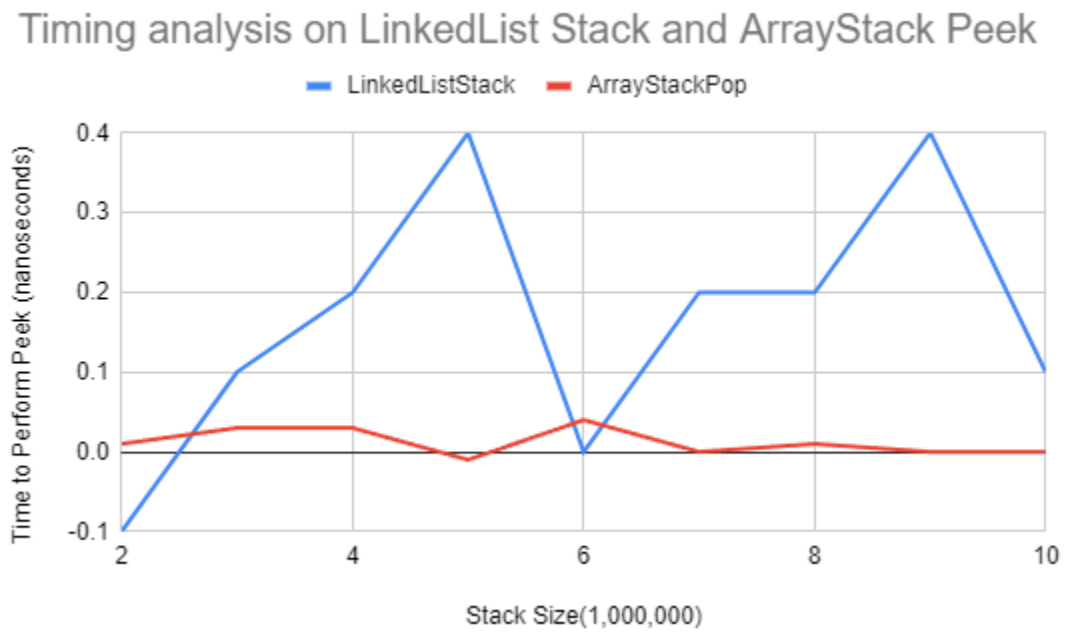
3b. Like above to compare the Running time of the "Pop" method on the ArrayStack and LinkedListStack classes we used a timing test with a stack size starting at 1,000,000 to 10,000,000 incrementing it by 1,000,000 each time. For each N value we first built up our stack then called the stack.pop() method 1,000 times, and took the average. Again, we see a lot of variation in our LinkedListStack since the timing is so small. We are expecting O(1) again for both classes, because our linked list will return the top node without needing to change any of the links on the other nodes regardless of size. For array list pop will return the element at the top index of the array, and increment the value for top by one, this will always take the same amount of time regardless of the size of the array/stack. In Graph 2 below our results are plotted. We can see from the graph and timing analysis below that ArrayStack will execute faster. Both classes appear to be O(1) as expected as well.



Graph 2

Stack Size(N)	LLStackPop	AStackPop		ArrayStack		
				O(1)	O(N)	O(logN)
2000000	3.5	0.4		0.4	2.00E-07	6.35E-02
3000000	2.7	0.5		0.5	1.67E-07	7.72E-02
4000000	3.4	0.4		0.4	1.00E-07	6.06E-02
5000000	1.5	0.3		0.3	6.00E-08	4.48E-02
6000000	2.7	0.4		0.4	6.67E-08	5.90E-02
7000000	1.7	0.4		0.4	5.71E-08	5.84E-02
8000000	1.8	0.3		0.3	3.75E-08	4.35E-02
9000000	3.1	0.3		0.3	3.33E-08	4.31E-02
10000000	2.8	0.3		0.3	3.00E-08	4.29E-02
Average Time	2.577777778	0.3666666667				

3c. Like above to compare the Running time of the “Peek” method on the ArrayStack and LinkedListStack classes we used a timing test with a stack size starting at 1,000,000 to 10,000,000 incrementing it by 1,000,000 each time. For each N value we first built up our stack then called the stack.peek() method 1,000 times, and took the average. Again, we see a lot of variation in our Stack class timing since the timing values are so small. We expect Peek to be O(1) for both our classes, since we are just returning the value on top for both methods, and not changing anything the operation will take the same time regardless of stack size. In Graph 3 below our results are plotted. We can see from the graph and timing analysis below that ArrayStack will execute faster. Both classes appear to be O(1) as expected as well. There is a lot of variation on the LinkedListstack, this was observed through running the test multiple times.



Graph 3

4. Based on our result from the three parts of Question 3, using the ArrayStack class in our TextEditer application would be just barely faster. One possible draw of using the Linked list is that our list will always be the minimum size of nodes needed, which we use a double backing array in ArrayStack , which could utilize more memory. This difference is small enough I think it can be considered negligible and would still recommend using ArrayStack for this TextEditor application.