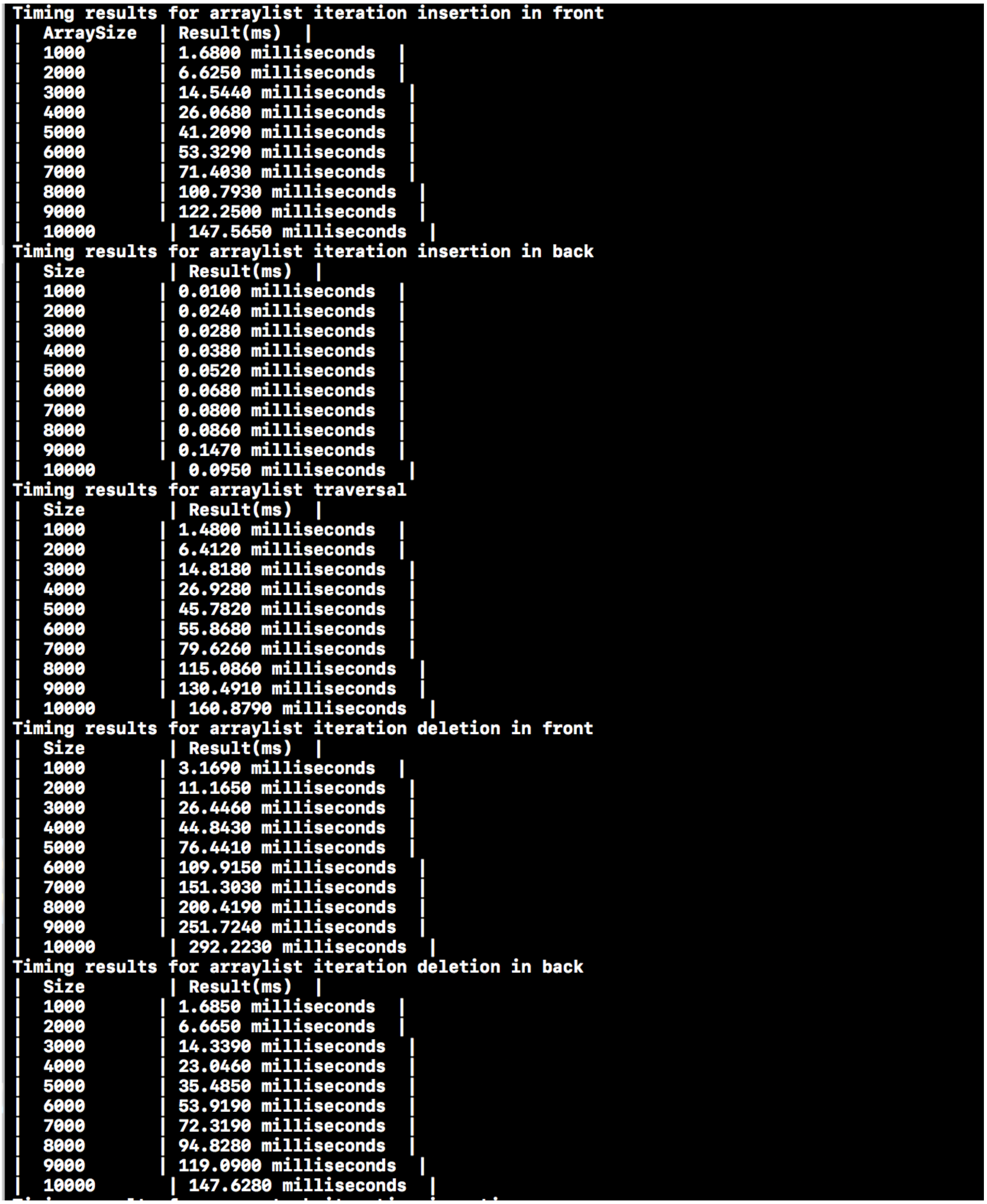
TIMING

ArrayList

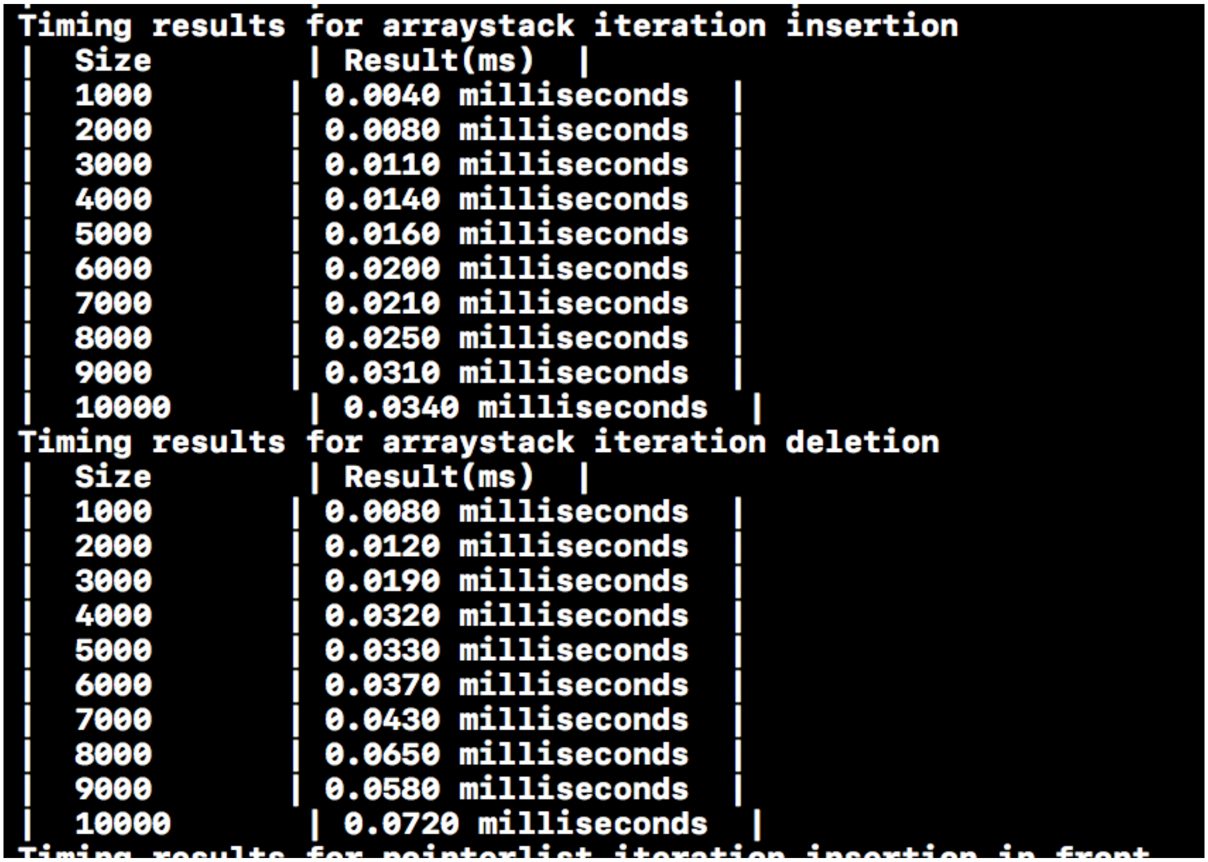


```
Timing results for arraylist iteration insertion in front
|  ArraySize  | Result(ms)  |
|  1000       | 1.6800 milliseconds  |
|  2000       | 6.6250 milliseconds  |
|  3000       | 14.5440 milliseconds  |
|  4000       | 26.0680 milliseconds  |
|  5000       | 41.2090 milliseconds  |
|  6000       | 53.3290 milliseconds  |
|  7000       | 71.4030 milliseconds  |
|  8000       | 100.7930 milliseconds  |
|  9000       | 122.2500 milliseconds  |
|  10000       | 147.5650 milliseconds  |
Timing results for arraylist iteration insertion in back
|  Size        | Result(ms)  |
|  1000       | 0.0100 milliseconds  |
|  2000       | 0.0240 milliseconds  |
|  3000       | 0.0280 milliseconds  |
|  4000       | 0.0380 milliseconds  |
|  5000       | 0.0520 milliseconds  |
|  6000       | 0.0680 milliseconds  |
|  7000       | 0.0800 milliseconds  |
|  8000       | 0.0860 milliseconds  |
|  9000       | 0.1470 milliseconds  |
|  10000       | 0.0950 milliseconds  |
Timing results for arraylist traversal
|  Size        | Result(ms)  |
|  1000       | 1.4800 milliseconds  |
|  2000       | 6.4120 milliseconds  |
|  3000       | 14.8180 milliseconds  |
|  4000       | 26.9280 milliseconds  |
|  5000       | 45.7820 milliseconds  |
|  6000       | 55.8680 milliseconds  |
|  7000       | 79.6260 milliseconds  |
|  8000       | 115.0860 milliseconds  |
|  9000       | 130.4910 milliseconds  |
|  10000       | 160.8790 milliseconds  |
Timing results for arraylist iteration deletion in front
|  Size        | Result(ms)  |
|  1000       | 3.1690 milliseconds  |
|  2000       | 11.1650 milliseconds  |
|  3000       | 26.4460 milliseconds  |
|  4000       | 44.8430 milliseconds  |
|  5000       | 76.4410 milliseconds  |
|  6000       | 109.9150 milliseconds  |
|  7000       | 151.3030 milliseconds  |
|  8000       | 200.4190 milliseconds  |
|  9000       | 251.7240 milliseconds  |
|  10000       | 292.2230 milliseconds  |
Timing results for arraylist iteration deletion in back
|  Size        | Result(ms)  |
|  1000       | 1.6850 milliseconds  |
|  2000       | 6.6650 milliseconds  |
|  3000       | 14.3390 milliseconds  |
|  4000       | 23.0460 milliseconds  |
|  5000       | 35.4850 milliseconds  |
|  6000       | 53.9190 milliseconds  |
|  7000       | 72.3190 milliseconds  |
|  8000       | 94.8280 milliseconds  |
|  9000       | 119.0900 milliseconds  |
|  10000       | 147.6280 milliseconds  |
```

- I believe there is a huge difference for inserting from back and front because for inserting from front you have to push every element by one, that’s why it takes so much more time to do it. Same with deleting, there is almost double the time difference between front and back.

  - Thus, for inserting and deleting from front it take O(n^2) and Ω(n^2) because it takes double to time to iterate every element and then do the operation on the other hand for in the front it takes only O(n) and Ω(n) because we only care about inserting and deleting.
  - Traverse function is more like back elements which takes only one set of functions and that's why it takes O(n) and Ω(n) time to accomplish.

<u>ArrayStack</u>



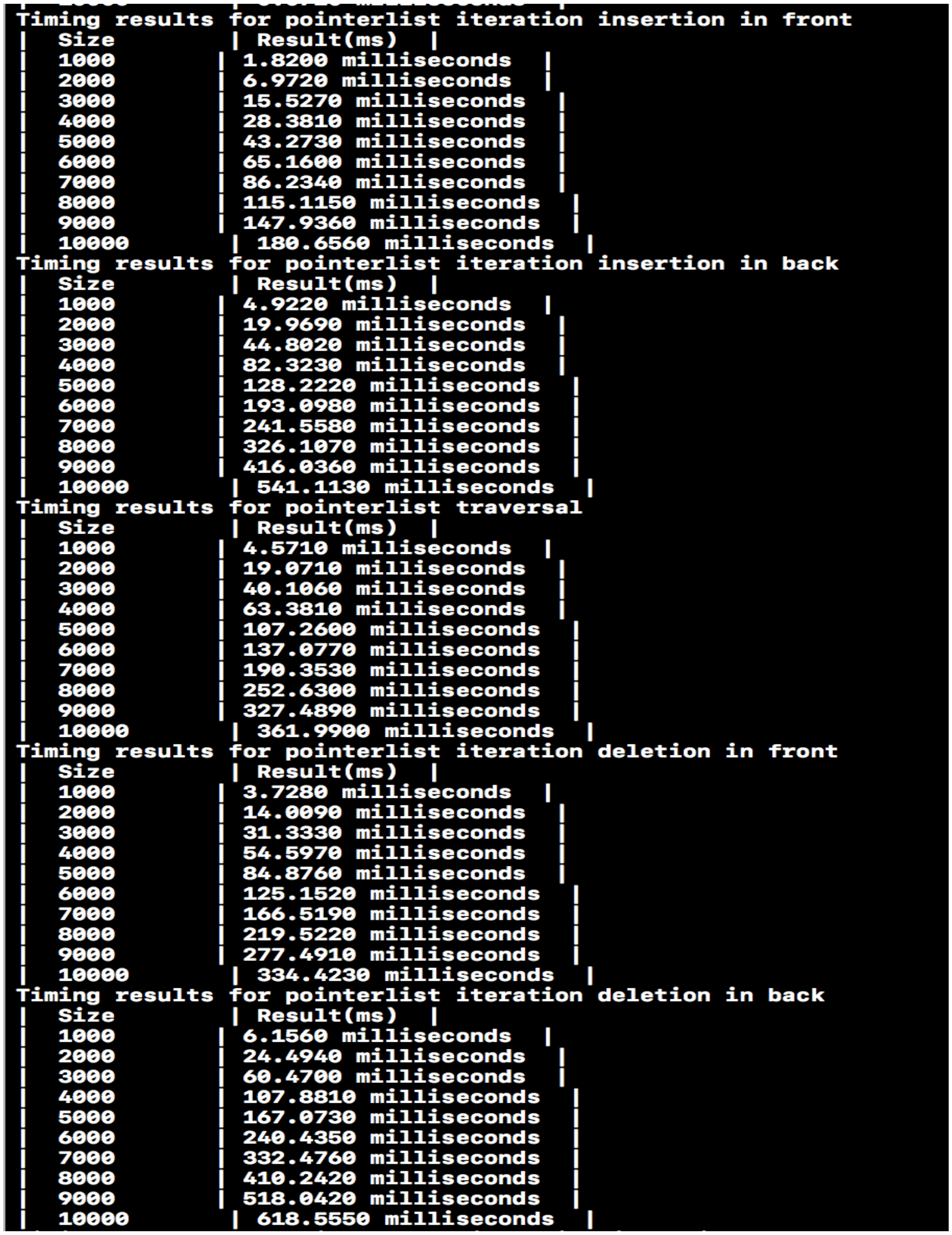
```
Timing results for arraystack iteration insertion
|   Size         | Result(ms)  |
|   1000        | 0.0040 milliseconds  |
|   2000        | 0.0080 milliseconds  |
|   3000        | 0.0110 milliseconds  |
|   4000        | 0.0140 milliseconds  |
|   5000        | 0.0160 milliseconds  |
|   6000        | 0.0200 milliseconds  |
|   7000        | 0.0210 milliseconds  |
|   8000        | 0.0250 milliseconds  |
|   9000        | 0.0310 milliseconds  |
|   10000        | 0.0340 milliseconds  |
Timing results for arraystack iteration deletion
|   Size         | Result(ms)  |
|   1000        | 0.0080 milliseconds  |
|   2000        | 0.0120 milliseconds  |
|   3000        | 0.0190 milliseconds  |
|   4000        | 0.0320 milliseconds  |
|   5000        | 0.0330 milliseconds  |
|   6000        | 0.0370 milliseconds  |
|   7000        | 0.0430 milliseconds  |
|   8000        | 0.0650 milliseconds  |
|   9000        | 0.0580 milliseconds  |
|   10000        | 0.0720 milliseconds  |
Timing results for pointerlist iteration insertion in front
```

- I believe stacks are very simple to move around because we do not need any looping, thus its pretty simple. The timing component will be O(n) and Ω(n) because we don't loop at all so all it takes is to look at the TOP() element and make computations with that.

<u>PointerList</u>

- I believe for pointerlist operations we have quadratic functions/orders of O(n^2) and Ω(n^2). Before we start it checks if we are inside of a list which takes n operations to do and then we actually start the action it will be n*n =N^2 thus giving us a O(n^2) and Ω(n^2) time complexity for this.
  - For inserting in the front it takes significantly less time than in back because we can just add a pointer to a list in the front but from the back we need to go through every element and then add thus it takes significantly more time, we can see this in the picture as well.

- For the traverse function since it needs to check and do n computations and for the traverse then it does n computations as well that’s why the traverse time is also O(n^2) and Ω(n^2).

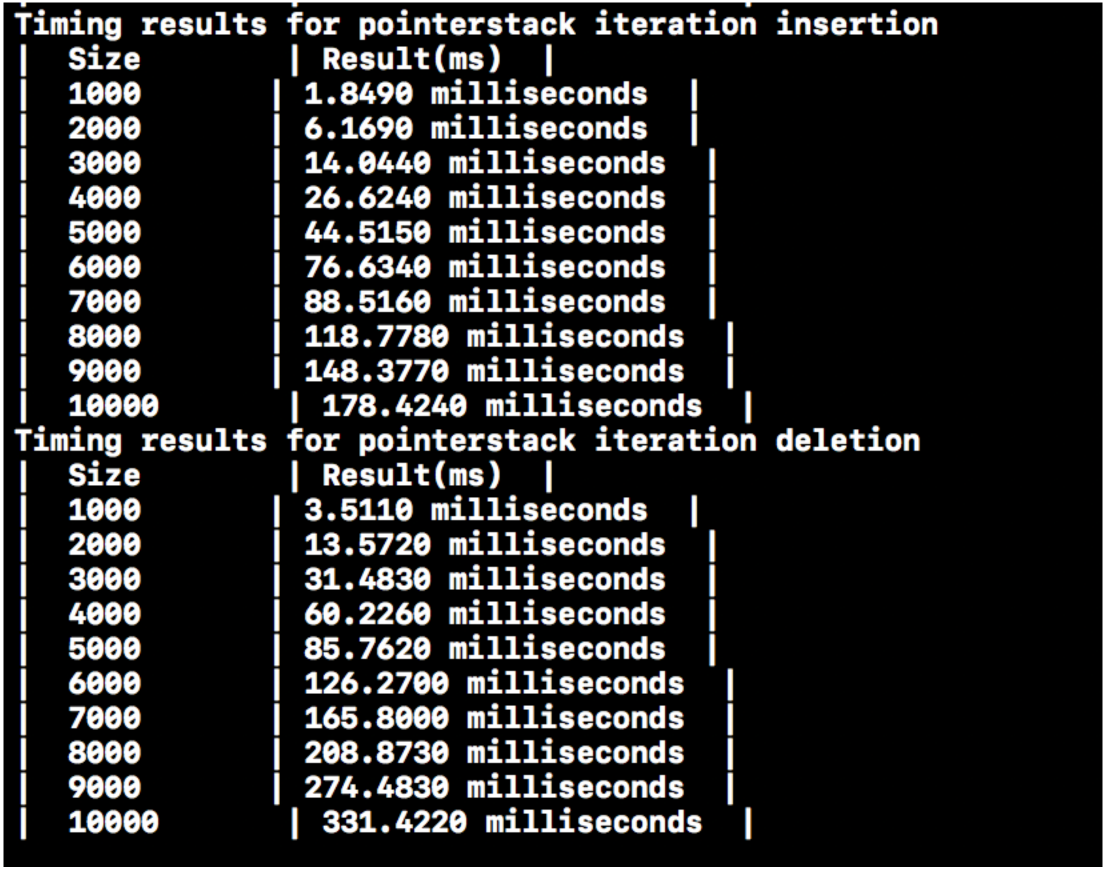


```
Timing results for pointerlist iteration insertion in front
|   Size         | Result(ms)  |
|   1000        | 1.8200 milliseconds  |
|   2000        | 6.9720 milliseconds  |
|   3000        | 15.5270 milliseconds  |
|   4000        | 28.3810 milliseconds  |
|   5000        | 43.2730 milliseconds  |
|   6000        | 65.1600 milliseconds  |
|   7000        | 86.2340 milliseconds  |
|   8000        | 115.1150 milliseconds  |
|   9000        | 147.9360 milliseconds  |
|   10000        | 180.6560 milliseconds  |
Timing results for pointerlist iteration insertion in back
|   Size         | Result(ms)  |
|   1000        | 4.9220 milliseconds  |
|   2000        | 19.9690 milliseconds  |
|   3000        | 44.8020 milliseconds  |
|   4000        | 82.3230 milliseconds  |
|   5000        | 128.2220 milliseconds  |
|   6000        | 193.0980 milliseconds  |
|   7000        | 241.5580 milliseconds  |
|   8000        | 326.1070 milliseconds  |
|   9000        | 416.0360 milliseconds  |
|   10000        | 541.1130 milliseconds  |
Timing results for pointerlist traversal
|   Size         | Result(ms)  |
|   1000        | 4.5710 milliseconds  |
|   2000        | 19.0710 milliseconds  |
|   3000        | 40.1060 milliseconds  |
|   4000        | 63.3810 milliseconds  |
|   5000        | 107.2600 milliseconds  |
|   6000        | 137.0770 milliseconds  |
|   7000        | 190.3530 milliseconds  |
|   8000        | 252.6300 milliseconds  |
|   9000        | 327.4890 milliseconds  |
|   10000        | 361.9900 milliseconds  |
Timing results for pointerlist iteration deletion in front
|   Size         | Result(ms)  |
|   1000        | 3.7280 milliseconds  |
|   2000        | 14.0090 milliseconds  |
|   3000        | 31.3330 milliseconds  |
|   4000        | 54.5970 milliseconds  |
|   5000        | 84.8760 milliseconds  |
|   6000        | 125.1520 milliseconds  |
|   7000        | 166.5190 milliseconds  |
|   8000        | 219.5220 milliseconds  |
|   9000        | 277.4910 milliseconds  |
|   10000        | 334.4230 milliseconds  |
Timing results for pointerlist iteration deletion in back
|   Size         | Result(ms)  |
|   1000        | 6.1560 milliseconds  |
|   2000        | 24.4940 milliseconds  |
|   3000        | 60.4700 milliseconds  |
|   4000        | 107.8810 milliseconds  |
|   5000        | 167.0730 milliseconds  |
|   6000        | 240.4350 milliseconds  |
|   7000        | 332.4760 milliseconds  |
|   8000        | 410.2420 milliseconds  |
|   9000        | 518.0420 milliseconds  |
|   10000        | 618.5550 milliseconds  |
```

PointerStack

- Normally stacks would take n time but since we are dealing with pointers, it takes more time to do any computations that’s why it takes O(n^2) and Ω(n^2) to finish insertion and deletion functionalities.

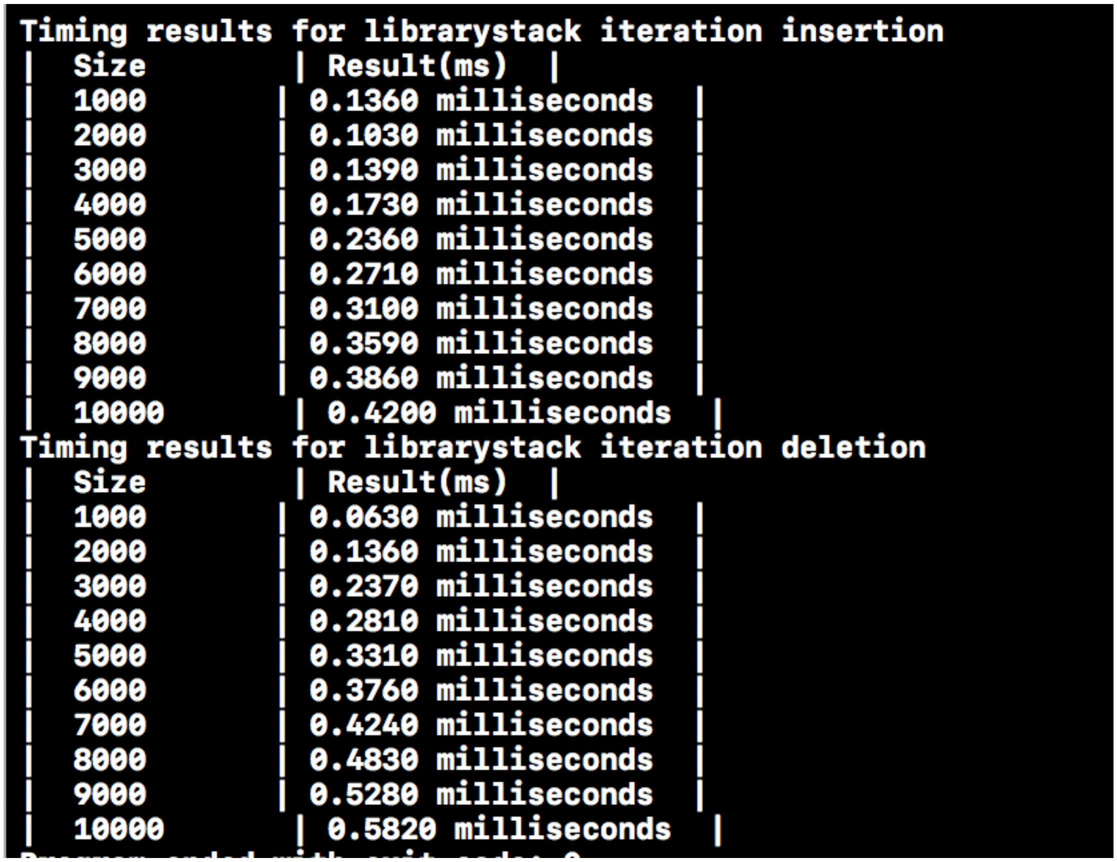
- This takes n^2 because the pointers, you have to loop through every element then we do the operations.



```
Timing results for pointerstack iteration insertion
|  Size         | Result(ms)  |
|  1000        | 1.8490 milliseconds  |
|  2000        | 6.1690 milliseconds  |
|  3000        | 14.0440 milliseconds  |
|  4000        | 26.6240 milliseconds  |
|  5000        | 44.5150 milliseconds  |
|  6000        | 76.6340 milliseconds  |
|  7000        | 88.5160 milliseconds  |
|  8000        | 118.7780 milliseconds  |
|  9000        | 148.3770 milliseconds  |
|  10000        | 178.4240 milliseconds  |
Timing results for pointerstack iteration deletion
|  Size         | Result(ms)  |
|  1000        | 3.5110 milliseconds  |
|  2000        | 13.5720 milliseconds  |
|  3000        | 31.4830 milliseconds  |
|  4000        | 60.2260 milliseconds  |
|  5000        | 85.7620 milliseconds  |
|  6000        | 126.2700 milliseconds  |
|  7000        | 165.8000 milliseconds  |
|  8000        | 208.8730 milliseconds  |
|  9000        | 274.4830 milliseconds  |
|  10000        | 331.4220 milliseconds  |
```

LibraryStack

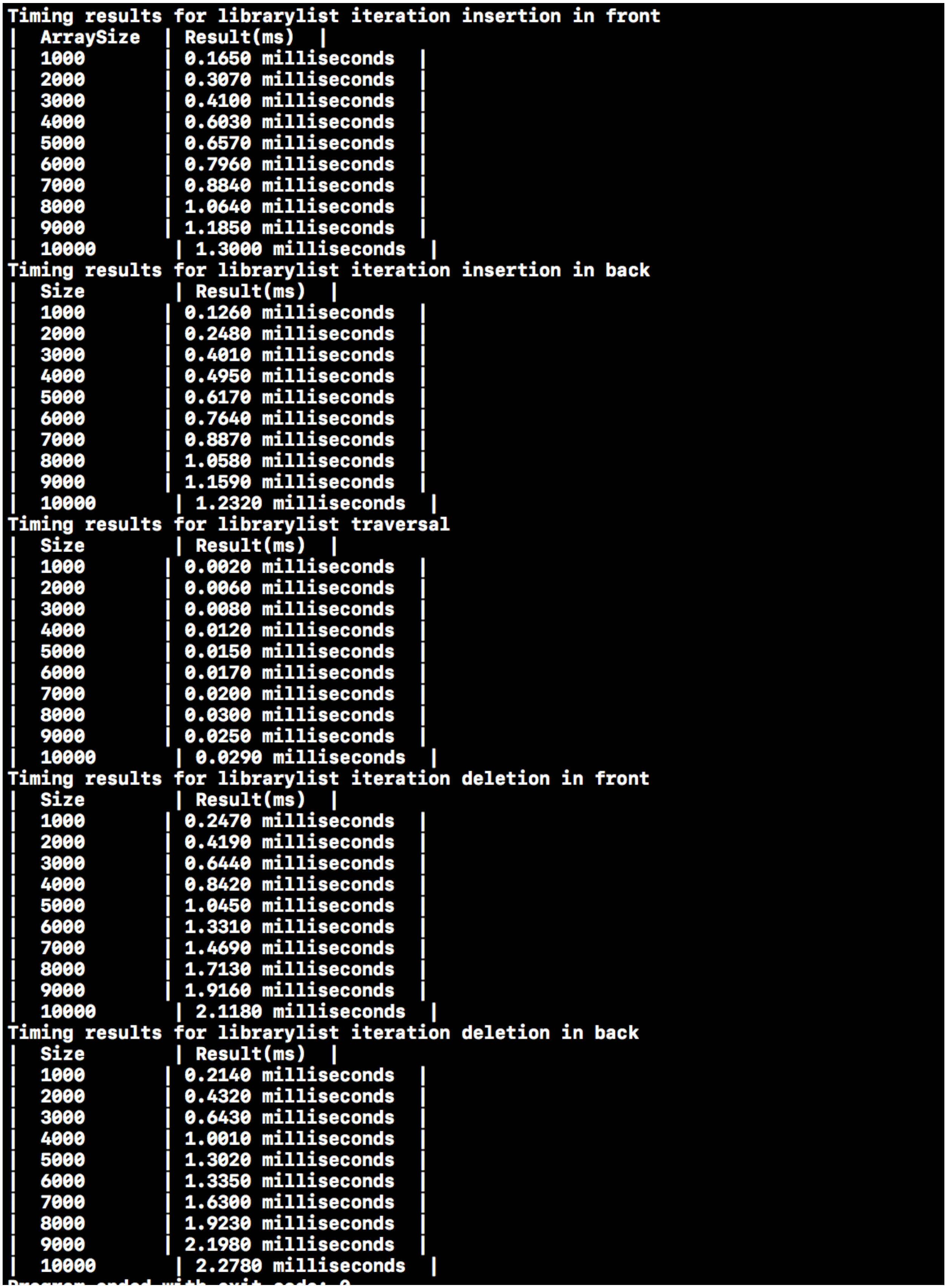
- Library implementation of the stack is similar to our because we don't use pointers and the usage of the stacks which only takes n operations to do the action we have O(n) and Ω(n) time complexity to finish a task.
    - For insertion and deletion for the library implementation we only care about the TOP(), thats why we have an relatively faster stack compare to other data types.



```
Timing results for librarystack iteration insertion
|  Size        | Result(ms)  |
|  1000       | 0.1360 milliseconds  |
|  2000       | 0.1030 milliseconds  |
|  3000       | 0.1390 milliseconds  |
|  4000       | 0.1730 milliseconds  |
|  5000       | 0.2360 milliseconds  |
|  6000       | 0.2710 milliseconds  |
|  7000       | 0.3100 milliseconds  |
|  8000       | 0.3590 milliseconds  |
|  9000       | 0.3860 milliseconds  |
|  10000       | 0.4200 milliseconds  |
Timing results for librarystack iteration deletion
|  Size        | Result(ms)  |
|  1000       | 0.0630 milliseconds  |
|  2000       | 0.1360 milliseconds  |
|  3000       | 0.2370 milliseconds  |
|  4000       | 0.2810 milliseconds  |
|  5000       | 0.3310 milliseconds  |
|  6000       | 0.3760 milliseconds  |
|  7000       | 0.4240 milliseconds  |
|  8000       | 0.4830 milliseconds  |
|  9000       | 0.5280 milliseconds  |
|  10000       | 0.5820 milliseconds  |
Program ended with exit code: 0
```

<u>LibraryList</u>

- For Library list it is fairly simple because since it's a library of lists it means we can read any element anytime or do anything with it, this gives us immense space for moving in between elements and doing computations thus making it a fast environment. That's why it has O(n) and Ω(n) time complexity to finish a task.
  - For the difference between front and the back there is not much difference because as I explained before we can go through elements fairly easily.
  - For the Traverse, there wasn't many data for me to come up with a resolution in this library but it is still has O(n) and Ω(n) time complexity to finish a task.



```
Timing results for librarylist iteration insertion in front
|  ArraySize  | Result(ms)  |
|  1000       | 0.1650 milliseconds  |
|  2000       | 0.3070 milliseconds  |
|  3000       | 0.4100 milliseconds  |
|  4000       | 0.6030 milliseconds  |
|  5000       | 0.6570 milliseconds  |
|  6000       | 0.7960 milliseconds  |
|  7000       | 0.8840 milliseconds  |
|  8000       | 1.0640 milliseconds  |
|  9000       | 1.1850 milliseconds  |
|  10000       | 1.3000 milliseconds  |
Timing results for librarylist iteration insertion in back
|  Size        | Result(ms)  |
|  1000       | 0.1260 milliseconds  |
|  2000       | 0.2480 milliseconds  |
|  3000       | 0.4010 milliseconds  |
|  4000       | 0.4950 milliseconds  |
|  5000       | 0.6170 milliseconds  |
|  6000       | 0.7640 milliseconds  |
|  7000       | 0.8870 milliseconds  |
|  8000       | 1.0580 milliseconds  |
|  9000       | 1.1590 milliseconds  |
|  10000       | 1.2320 milliseconds  |
Timing results for librarylist traversal
|  Size        | Result(ms)  |
|  1000       | 0.0020 milliseconds  |
|  2000       | 0.0060 milliseconds  |
|  3000       | 0.0080 milliseconds  |
|  4000       | 0.0120 milliseconds  |
|  5000       | 0.0150 milliseconds  |
|  6000       | 0.0170 milliseconds  |
|  7000       | 0.0200 milliseconds  |
|  8000       | 0.0300 milliseconds  |
|  9000       | 0.0250 milliseconds  |
|  10000       | 0.0290 milliseconds  |
Timing results for librarylist iteration deletion in front
|  Size        | Result(ms)  |
|  1000       | 0.2470 milliseconds  |
|  2000       | 0.4190 milliseconds  |
|  3000       | 0.6440 milliseconds  |
|  4000       | 0.8420 milliseconds  |
|  5000       | 1.0450 milliseconds  |
|  6000       | 1.3310 milliseconds  |
|  7000       | 1.4690 milliseconds  |
|  8000       | 1.7130 milliseconds  |
|  9000       | 1.9160 milliseconds  |
|  10000       | 2.1180 milliseconds  |
Timing results for librarylist iteration deletion in back
|  Size        | Result(ms)  |
|  1000       | 0.2140 milliseconds  |
|  2000       | 0.4320 milliseconds  |
|  3000       | 0.6430 milliseconds  |
|  4000       | 1.0010 milliseconds  |
|  5000       | 1.3020 milliseconds  |
|  6000       | 1.3350 milliseconds  |
|  7000       | 1.6300 milliseconds  |
|  8000       | 1.9230 milliseconds  |
|  9000       | 2.1980 milliseconds  |
|  10000       | 2.2780 milliseconds  |
Program ended with exit code: 0
```