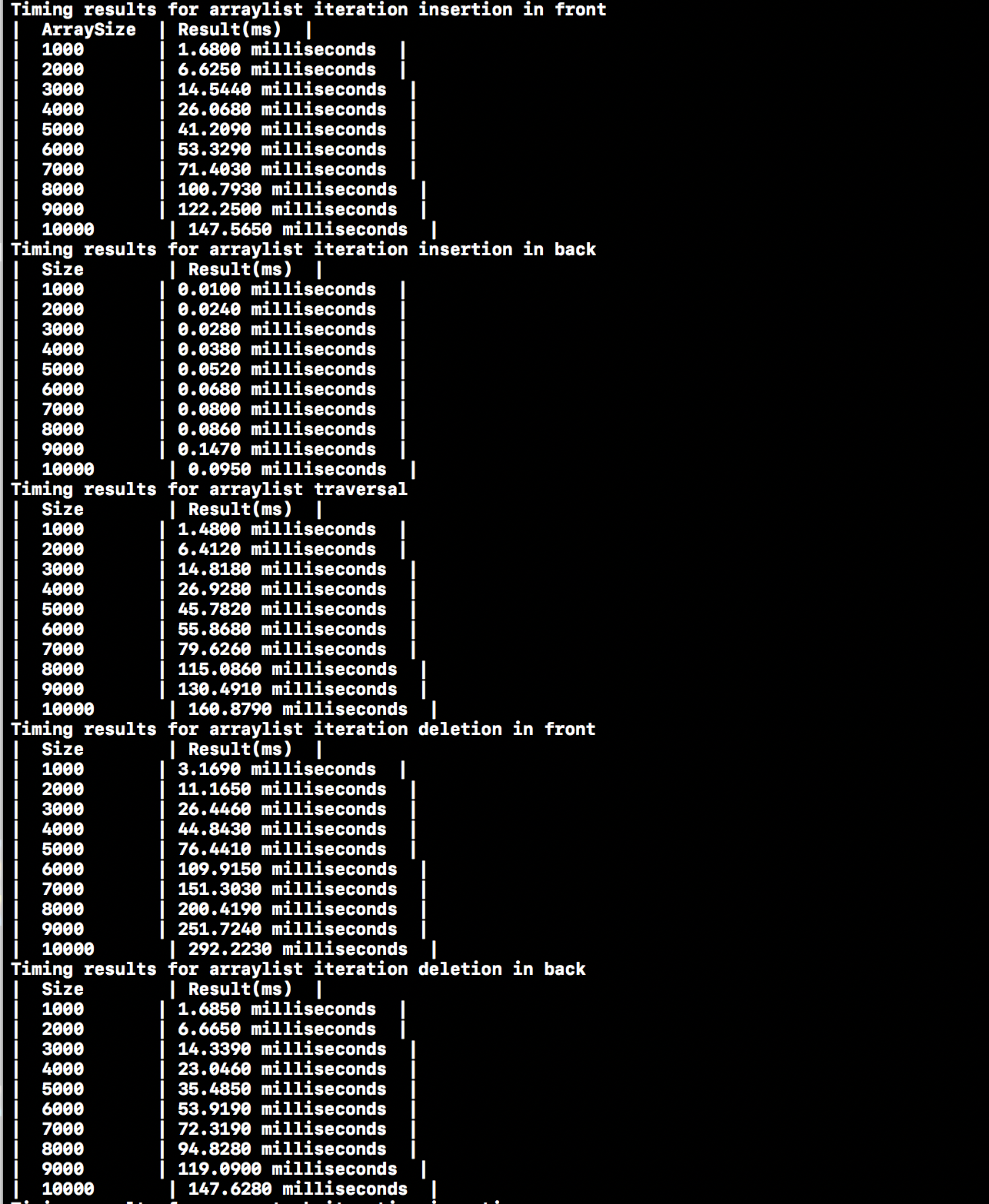
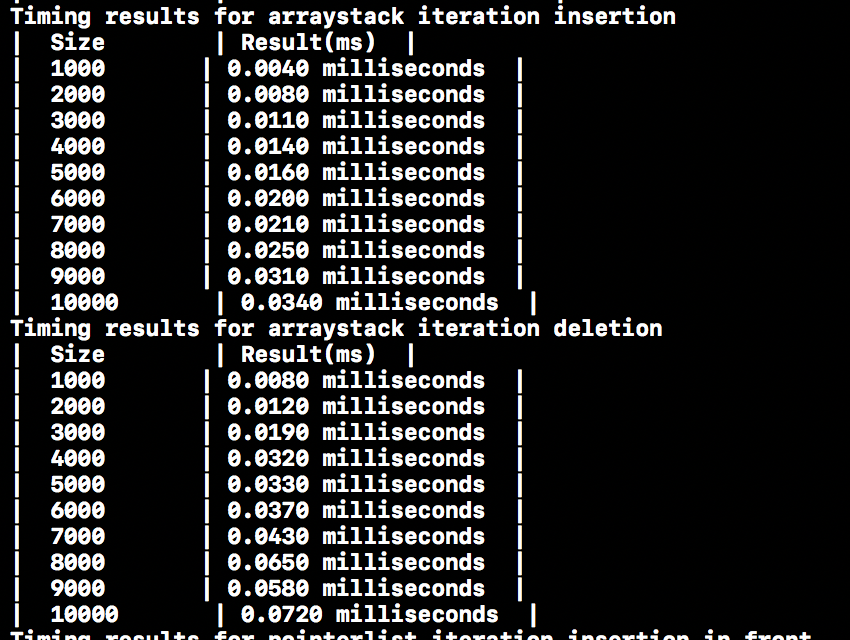
***TIMING***

***ArrayList***



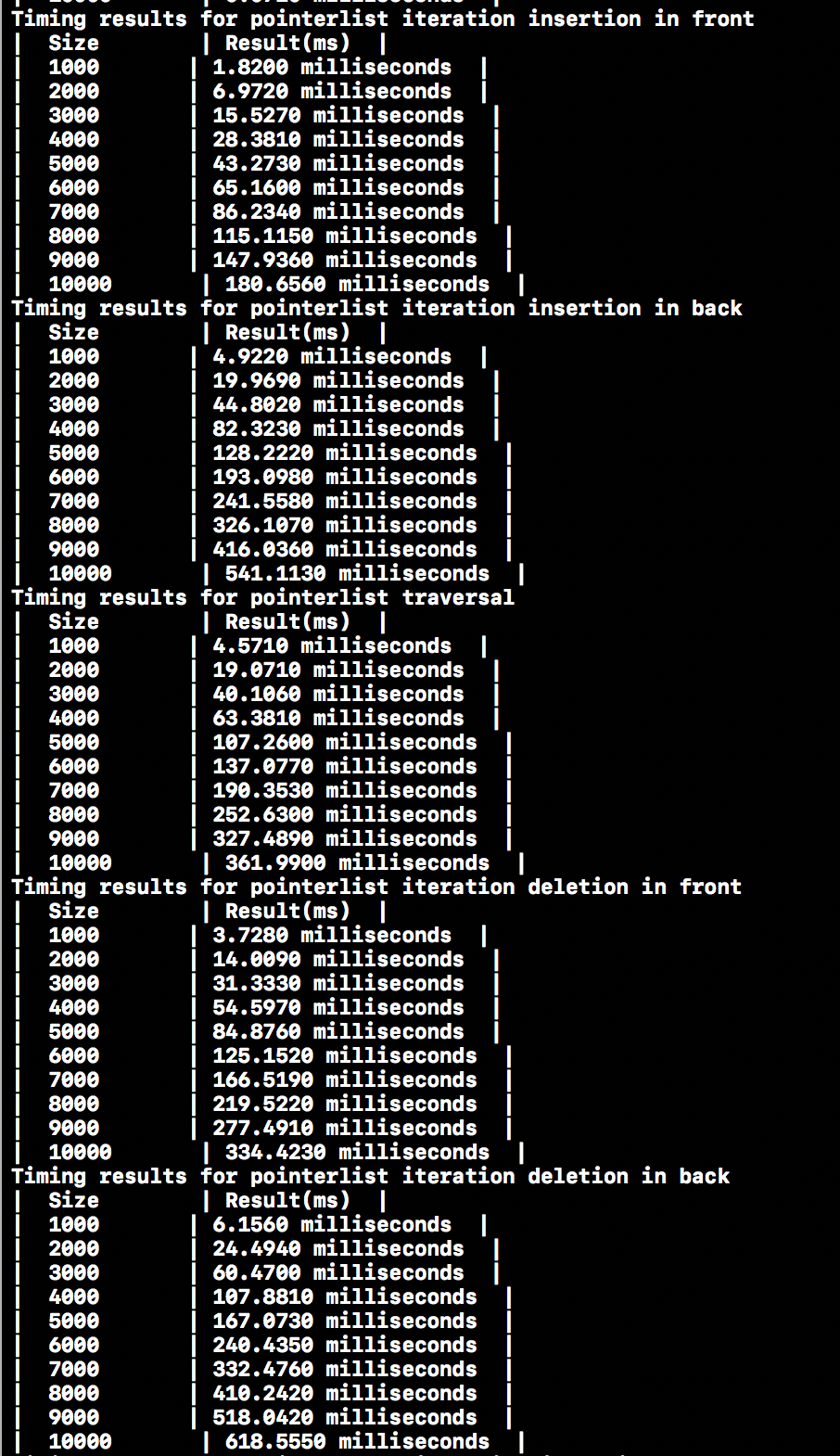
* I believe there is a huge difference for inserting from back and front because for inserting from front you have to push every element by one, that’s why it takes so much more time to do it. Same with deleting, there is almost double the time difference between front and back.
  + Thus, for inserting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

***ArrayStack***

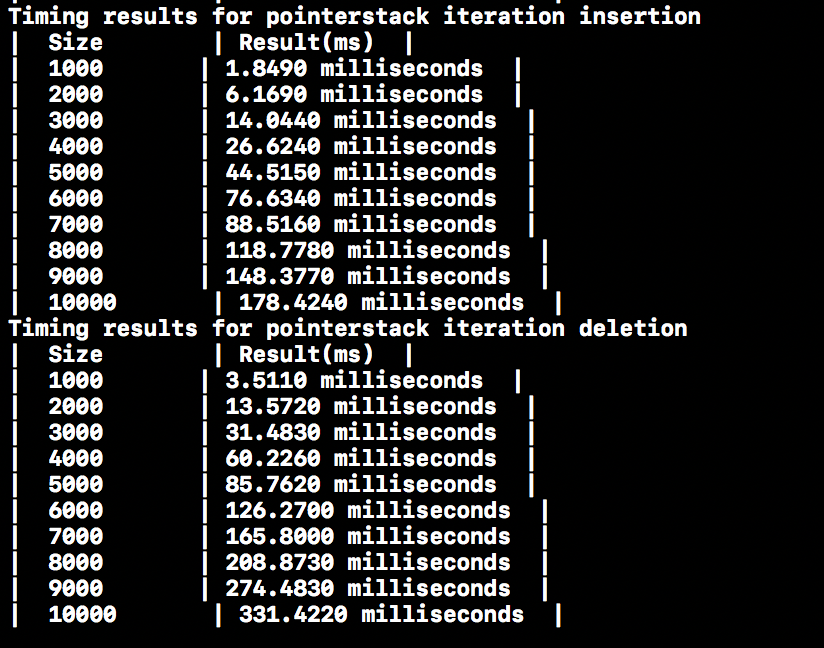
* I believe stacks are very simple to move around because we do not need any looping, thus its pretty simple. The timing component will be O(n) and Ω(n) because we don’t loop at all so all it takes is to look at the TOP() element and make computations with that.

***PointerList***

* 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).

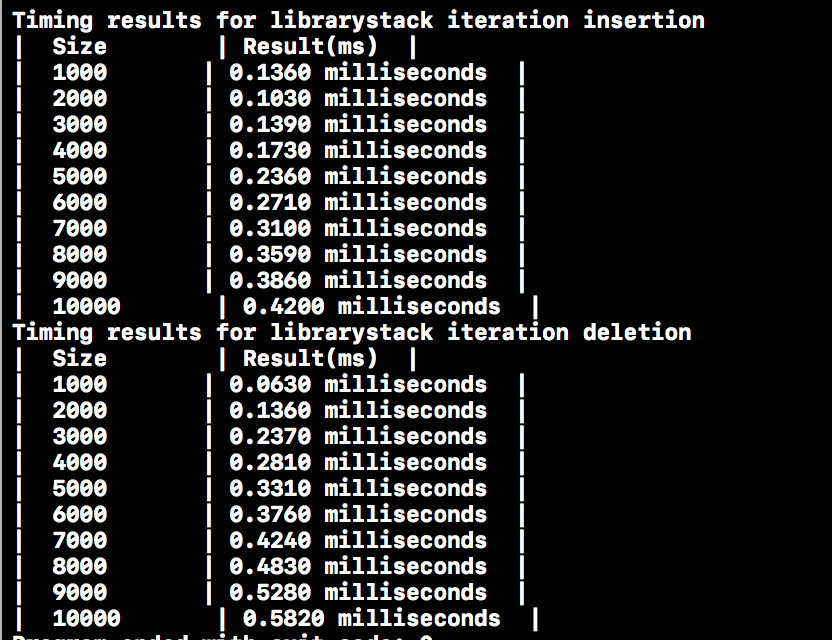
******

***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.

***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.



***LibraryList***

* 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.

