Study.pdf – Coby Hong – May 10, 2017

Specs:

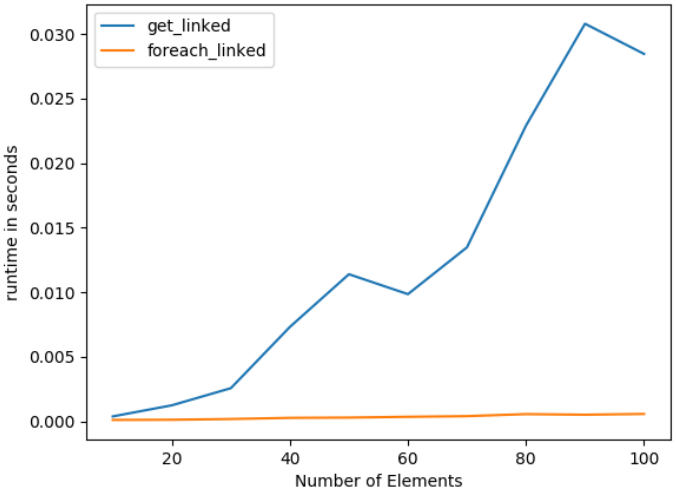
- Intel Core i7 6th Gen 6500U (2.50 GHz)
- 8 GB Memory 1 TB HDD
- Intel HD Graphics 520
- 1920 x 1080
- Windows 10 Home 64-Bit
- DL DVD+-RW/CD-RW

Elements start at 10 and are increased by 10 at a range of 10. Total of 100 elements.

Using matplotlib to create graphs.

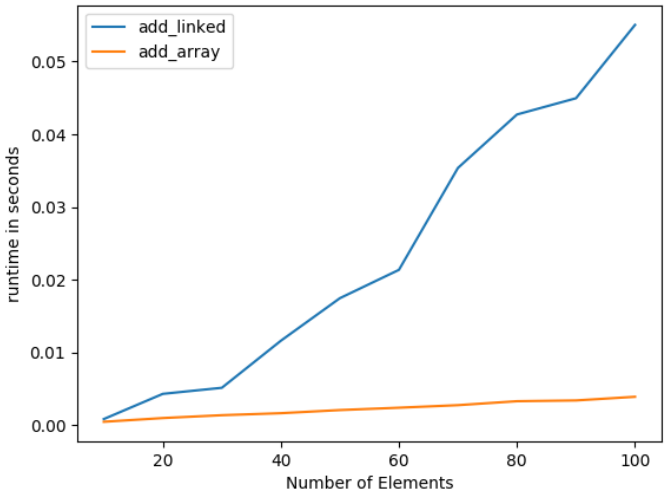
File here:

study.py



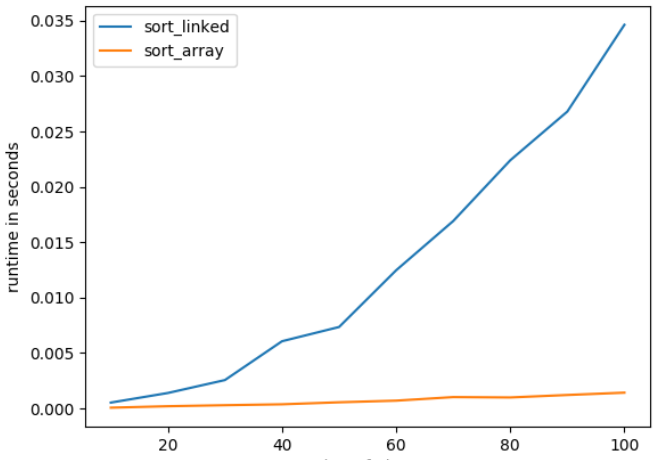
Get vs Foreach:
Linked List:

Get seems to have a o(n) or o(n^2). Foreach has a linear o(l). Get likely has this operation of 2 O(n) since it must increment position and traverse/ reiterate over until it finds a value matching at the lst.first; this time varying by the value's position in the list. Meanwhile foreach just applies to every element.



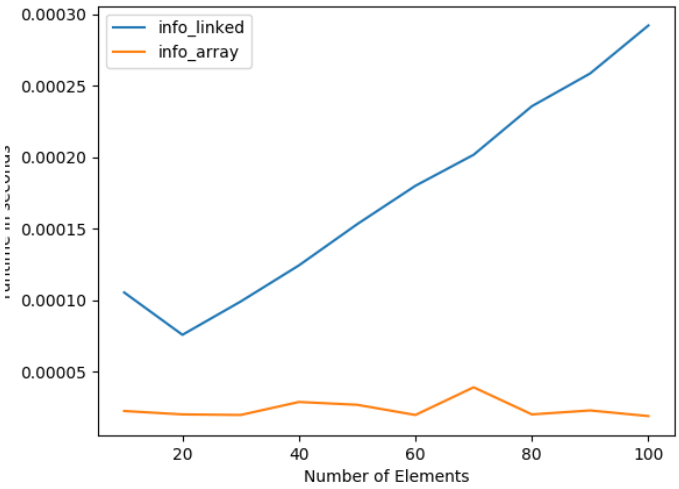
Load Time:
Linked List vs Array List:

We are comparing the time it takes to build lists. Linked list is o(n). This is because the add function must be called a multitude of times. Array list is o(n) as well since a shift in elements must occur multiple times



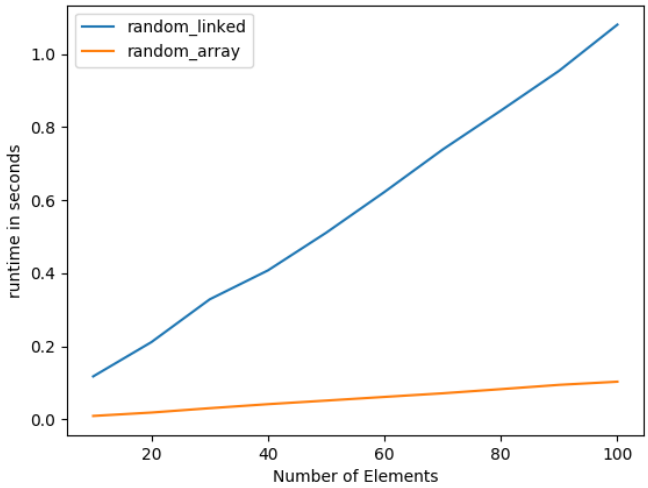
Sort:
Linked List vs Array List:

Here we are comparing sort times. Insertion sort used on Array List and Linked List. Linked List is o(n^2), caused by usage of an insert function from list_ops with a comparison. Array should be o(n) but came out as o(l). My best guess if because



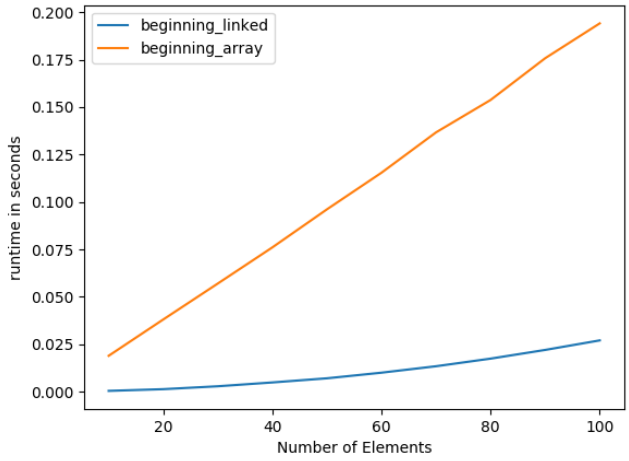
Song info:
Linked List vs Array List:

We are grabbing info from song list using get. For array, this is an o(l) since we can directly access the index. Linked List is a o(n) since recursion must be done to traverse the list.



Adding song s random/realistic:
Linked List vs Array List:

Adds songs to list but at random indexes. Though Linked List should be o(l), ever increasing runtime causes operation to take longer. Same reason as for Array List.

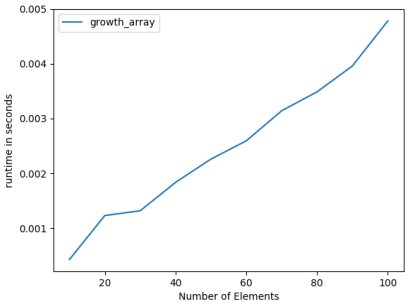


Adding songs contrived / beginning:

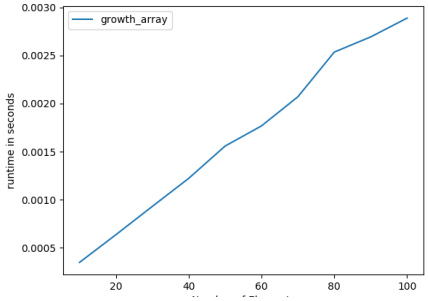
Linked List vs Array List:

Both have an o(n) operation. This due to the runtime with more elements being added. Basically, the same reasons as for "Adding songs randomly / realistic".

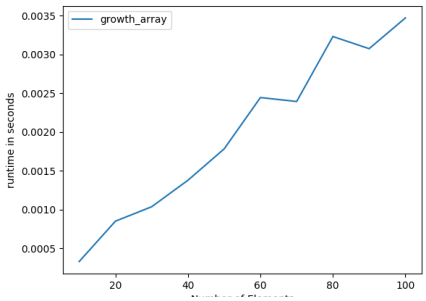
Growth Rate Strategies:



Growth by one element.



Growth by double.



Growth by five.

Growth by one element builds a list by adding one additional element to each call of add. Basically, minimal allocation to add to list being one. Growth by double builds list by doubling or two times current size of the list. Growth by five just builds list by allocating specific amount; this being an increase by 5 every time. All roughly follow an O(l) or linear behavior, but creation of new list and running of add returns a complexity of O(n).