EMPIRICAL STUDY

Justin Mo

Linked List (foreach vs. get)

Raw Data (foreach vs. get)

linked_list.foreach(): 10 elements 10 iterations in 0.000 seconds
linked_list.foreach(): 100 elements 10 iterations in 0.000 seconds
linked_list.foreach(): 1000 elements 10 iterations in 0.004 seconds
linked_list.foreach(): 5000 elements 10 iterations in 0.029 seconds
linked_list.foreach(): 10000 elements 10 iterations in 0.064 seconds

linked_list.get(): 10 elements 1 iterations in 0.000 seconds
linked_list.get(): 100 elements 1 iterations in 0.004 seconds
linked_list.get(): 1000 elements 1 iterations in 0.350 seconds
linked_list.get(): 5000 elements 1 iterations in 11.687 seconds
linked_list.get(): 10000 elements 1 iterations in 49.513 seconds

Linked List (foreach vs. get) Analysis

As we can see from the raw data, it is much more efficient to access each element in the linked list with the foreach function. foreach() is able to iterate 10,000 elements in just 0.064 seconds whereas it takes get() 49.5113 seconds to iterate through the same amount of elements.

Array List Growth Strategies

Increase capacity by 1 byte each time:
array_list.build_list(): 10 elements 10 iterations in 0.001 seconds
array_list.build_list(): 100 elements 10 iterations in 0.030 seconds
array_list.build_list(): 1000 elements 10 iterations in 2.544 seconds
array_list.build_list(): 5000 elements 10 iterations in 65.604 seconds
array_list.build_list(): 10000 elements 10 iterations in 284.148 seconds

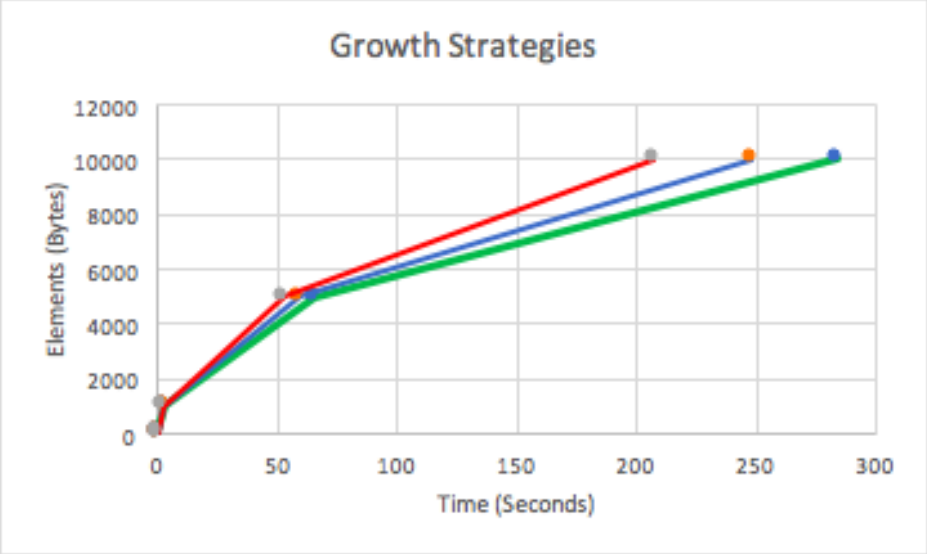
Increase capacity by 10 bytes each time:
array_list.build_list(): 10 elements 10 iterations in 0.001 seconds
array_list.build_list(): 100 elements 10 iterations in 0.029 seconds
array_list.build_list(): 1000 elements 10 iterations in 2.324 seconds
array_list.build_list(): 5000 elements 10 iterations in 58.924 seconds

array_list.build_list(): 10000 elements 10 iterations in 248.032 seconds

Increase capacity by 20 bytes each time:

array_list.build_list(): 10 elements 10 iterations in 0.001 seconds
array_list.build_list(): 100 elements 10 iterations in 0.020 seconds
array_list.build_list(): 1000 elements 10 iterations in 1.965 seconds
array_list.build_list(): 5000 elements 10 iterations in 52.648 seconds
array_list.build_list(): 10000 elements 10 iterations in 206.755 seconds

Growth Strategy Analysis



The first of the three different growth strategies that I utilized increases the capacity by 1 byte every time the allocated memory is filled up. This proved to be the most inefficient strategy because it took the most time (284.148 seconds) to iterate through 10,000 elements (shown by the green trend line).

The second of the three different growth strategies (blue trend line) that I utilized increases the capacity by 10 bytes every time the allocated memory is filled up. This strategy took slightly less time (248.032 seconds to iterate through 10,000 elements) than the first strategy but still took longer than my third strategy.

The final growth strategy that I utilized (red trend line) proved to be the most efficient. It increased the capacity by 20 bytes every time the allocated memory filled up. It took 206.755 seconds to iterate through 10,000 elements, which was faster than my first and second growth strategies.

In conclusion, we can induce from this data that the more memory allocated, the more efficiently our code runs over a large number of elements because it doesn't have to keep increasing capacity.

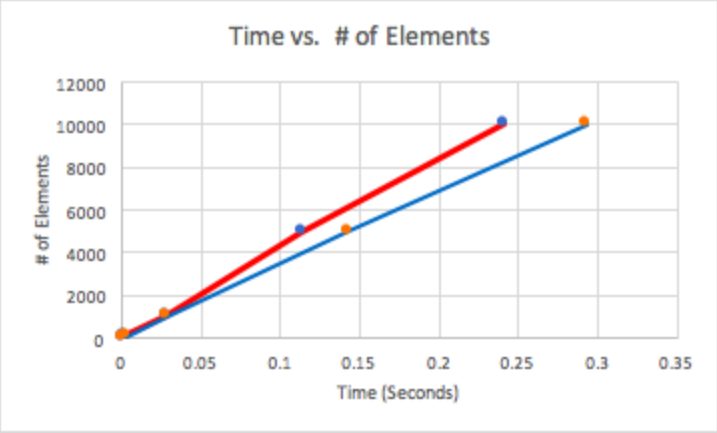
Array List vs. Linked List

Load Raw Data

linked_list.build_list():	10 elements	10 iterations in	0.000 seconds
linked_list.build_list():	100 elements	10 iterations in	0.002 seconds
linked_list.build_list():	1000 elements	10 iterations in	0.028 seconds
linked_list.build_list():	5000 elements	10 iterations in	0.114 seconds
linked_list.build_list():	10000 elements	10 iterations in	0.241 seconds

array_list.build_list():	10 elements	10 iterations in	0.000 seconds
array_list.build_list():	100 elements	10 iterations in	0.003 seconds
array_list.build_list():	1000 elements	10 iterations in	0.028 seconds
array_list.build_list():	5000 elements	10 iterations in	0.143 seconds
array_list.build_list():	10000 elements	10 iterations in	0.293 seconds

Load Analysis



As we can see from the raw data and graph, the linked list (red line) builds faster than the array list (blue line). The linked list can be built with 10,000 elements in a mere

0.241 second. On the other hand, the array list takes 0.293 seconds to be built with the same number of elements.

Sort Raw Data

array_list.build_list():	10 elements	10 iterations in	0.000 seconds
array_list.build_list():	100 elements	10 iterations in	0.002 seconds
array_list.build_list():	1000 elements	10 iterations in	0.031 seconds
array_list.build_list():	5000 elements	10 iterations in	0.163 seconds
array_list.build_list():	10000 elements	10 iterations in	0.299 seconds

linked_list.sort():	10 elements	10 iterations in	0.000 seconds
linked_list.sort():	100 elements	10 iterations in	0.003 seconds
linked_list.sort():	1000 elements	10 iterations in	0.021 seconds
linked_list.sort():	5000 elements	10 iterations in	0.125 seconds
linked_list.sort():	10000 elements	10 iterations in	0.240 seconds

Sort Analysis

As we can see from the raw data, sorting linked list is again more efficient than sorting the array list. It only takes 0.240 seconds to sort 10,000 elements in a linked list whereas it takes 0.299 seconds to sort an array list with the same number of elements.

Song Information Raw Data

array_list.get():	10 elements	10 iterations in	0.000 seconds
array_list.get():	100 elements	10 iterations in	0.001 seconds
array_list.get():	1000 elements	10 iterations in	0.005 seconds
array_list.get():	5000 elements	10 iterations in	0.028 seconds
array_list.get():	10000 elements	10 iterations in	0.059 seconds

linked_list.get():	10 elements	1 iterations in	0.000 seconds
linked_list.get():	100 elements	1 iterations in	0.004 seconds
linked_list.get():	1000 elements	1 iterations in	0.350 seconds
linked_list.get():	5000 elements	1 iterations in	11.687 seconds
linked_list.get():	10000 elements	1 iterations in	49.513 seconds

Song Information Analysis

As we can see from the raw data, array list is much more efficient when accessing random elements via the get function. It only takes 0.059 seconds to iterate through 10,000 elements for the array list but takes 49.513 seconds to iterate through the same number of elements for linked list. This is an expected result because array list can access elements much more quickly since, unlike the linked list, going through the entire list every time isn't necessary.

Add Songs Arbitrarily Raw Data

array_list.add_rando():	10 elements	10 iterations in 0.027 seconds
array_list.add_rando():	100 elements	10 iterations in 0.094 seconds
array_list.add_rando():	1000 elements	10 iterations in 2.781 seconds
array_list.add_rando():	5000 elements	10 iterations in 63.207 seconds
array_list.add_rando():	10000 elements	10 iterations in 274.316 seconds

linked_list.add_rando():	10 elements	10 iterations in 0.002 seconds
linked_list.add_rando():	100 elements	10 iterations in 0.004 seconds
linked_list.add_rando():	1000 elements	10 iterations in 0.032 seconds
linked_list.add_rando():	5000 elements	10 iterations in 0.124 seconds
linked_list.add_rando():	10000 elements	10 iterations in 0.248 seconds

Add Songs Arbitrarily Analysis

As we can see from our raw data, it is much more efficient to add songs randomly to a linked list. It only takes 0.248 seconds to iterate 10,000 elements for the linked list whereas it takes 274.317 seconds to iterate the same number of elements for the array list. This is an expected result because it takes much longer to insert number arbitrarily into an array list since everything must be shifted down after each add.

Add Songs to the Beginning Raw Data

array_list.add():	10 elements	10 iterations in 0.029 seconds
array_list.add():	100 elements	10 iterations in 0.099 seconds
array_list.add():	1000 elements	10 iterations in 2.752 seconds
array_list.add():	5000 elements	10 iterations in 61.625 seconds
array_list.add():	10000 elements	10 iterations in 237.136 seconds

linked_list.add():	10 elements	10 iterations in 0.002 seconds

linked_list.add(): 100 elements 10 iterations in 0.006 seconds
linked_list.add(): 1000 elements 10 iterations in 0.024 seconds
linked_list.add(): 5000 elements 10 iterations in 0.115 seconds
linked_list.add(): 10000 elements 10 iterations in 0.248 seconds

Add Songs to the Beginning Analysis

As we can see from the raw data, the results are similar to adding songs arbitrarily. It is much more efficient to add into a linked list as it only takes 0.248 seconds to iterate through 10,000 elements. However, it takes 237.136 seconds to iterate through the same number of elements for array list. This is the expected result because it should definitely be faster to add to the beginning of a linked list since, unlike the arraylist, not everything needs to be shifted down.