Kushi Jakati

CTIS 310

## **Project 7: Tables Comparing ArrayList and LinkedList**

### 

### **Random Get (1,000,000 times with n elements)**

**ArrayList**

| # of Elements | Operation | Approx. Time Average of 5Runs |
| --- | --- | --- |
| 1,000 | Random Get | 28.95 |
| 5,000 | Random Get | 29.2 |
| 10,000 | Random Get | 36.4 |

**LinkedList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Random Get | 1021.4 |
| 5,000 | Random Get | 12550.4 |
| 10,000 | Random Get | 28283.8 |

### **Sequential Get (1,000,000 times with n elements)**

**ArrayList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Sequential Get | 13.6 |
| 5,000 | Sequential Get | 12.8 |
| 10,000 | Sequential Get | 21.8 |

**LinkedList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Sequential Get | 641.4 |
| 5,000 | Sequential Get | 10673.6 |
| 10,000 | Sequential Get | 25641 |

### **Add (with n elements)**

**ArrayList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Add | .01 |
| 5,000 | Add | .02 |
| 10,000 | Add | .01 |

**LinkedList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Add | .042 |
| 5,000 | Add | .076 |
| 10,000 | Add | .212 |

### 

### **Sort (with n elements)**

**ArrayList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Sort | 1.3 |
| 5,000 | Sort | 4.03 |
| 10,000 | Sort | 7.5 |

**LinkedList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Sort | .97 |
| 5,000 | Sort | 3.51 |
| 10,000 | Sort | 5.6 |

### **Shuffle (with n elements)**

**ArrayList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Shuffle | .46 |
| 5,000 | Shuffle | 1.2 |
| 10,000 | Shuffle | 2.3 |

**LinkedList**

| # of Elements | Operation | Approx. Time Average of 5 Runs |
| --- | --- | --- |
| 1,000 | Shuffle | .5 |
| 5,000 | Shuffle | 1.9 |
| 10,000 | Shuffle | 2.7 |

In that same document write a paragraph or two that a) summarize which implementation appears to be better for each kind of operation; b) discuss how those results are consistent with how the list is implemented.

For the Random Get and Sequential Get operations, ArrayList was faster than Linked List, and that difference became even more exaggerated as the number of elements increased from 1,000 to 10,000. While the approximate time for ArrayList only increased by a few seconds, LinkedList increased by an estimated 10,000 seconds. The result from Random Get and LinkedList can be attributed to the fact that Linked Lists must get items in sequential order, compared to the contiguous ArrayList which can just pull the index of the item (unlike Linked List, which must go through the data location of the list and see if it is pointing to the desired object). Thus, even if LinkedList is at an advantage in Sequential Get in the very beginning because all the elements are at the start of the list when we have to per se, get the very end of the list, LinkedList still needs to go through all the elements of the list before it.

For the Add (insert in a random spot in the index) operation, ArrayList was slightly faster. Although ArrayList needs to shift elements when adding an element randomly, it is likely LinkedList took slightly longer than ArrayList because it needs to allocate new memory when adding a node/element. Finally, for Sort, LinkedList was slightly faster. Although I expected ArrayList to be faster because an ArrayList’s random access to data and cache locality would make sorting easier compared to LinkedList, the LinkedList being better was likely because the sample data used may not be large enough to produce significant differences. It may also be how we’re sorting (Collections.sort) that is causing this. With regard to Shuffle, ArrayList was just slightly faster, which is once again expected because ArrayList can randomly access and swap elements faster compared to a LinkedList that needs to go sequentially through each element to swap (rearranging pointers for blocks of memory as well).