CS2852 Lab 1

Name: Denise Malisa

Date: 3/15/2021

Instructor: Dr. Nowling

# Part 1

## What is the advantage of using the List interface and avoiding direct references to the ArrayList and LinkedList classes?

Your answer here

It can be used interchangeable like; in the case of the lab we were able to have both LinkedList and ArrayList used very easily.

## Why do you think we created the ItemRequest class?

Your answer here

I think it was created it to add the userId and itemId into a list(LinkedList or ArrayList) more easily.

## Why do you think we created the LibraryInventory class? If we wanted to make the simulation more realistic, how could we change this class?

Your answer here

N/A

# Part 2

## Create a table of your benchmark results.

Your answer here

|  |  |  |
| --- | --- | --- |
|  | ArrayList Waiting List (ms) | LinkedList Waiting List (ms) |
| benchmarkAdd | 9.221 | 7.717 |
| benchmark fulfillable requests | 9.374 | 20.68 |
| benchmark unfillable requests | 57.75 | 132.3 |

## The requestItem() method adds the object to the end or back of the list. Did you notice a significant difference in run time between the two types of lists?

Your answer here

Yes, there is a significant difference between the time it takes to add an object to the back of a LinkedList and ArrayList. It took a longer time to add to the back of the Arraylist than the linkedList.

## For the benchmark of the nextFulFillableRequest (true) method in which all items are fulfillable, the method will always remove and return the first item from the front of the list. Did you notice a significant difference in performance between the two types of lists?

Your answer here

Yes, in this case the ArrayList performs way better than the LinkedList in that it takes a noteworthy shorter amount of time.

## For the benchmark of the nextFulFillableRequest(false) method in which no items are fulfillable, the method will search through all items in the list, find no matches, and return null. Did you notice a significant difference in performance between the two types of lists? (Note that depending on how you implemented the method, your benchmark results may vary.)

Your answer here

Yes, I notice a significant difference in the performance between the two types of lists here again the Array performs more favorably than the LinkedList and takes a grossly shorter amount of time to perform the task.

## Thinking back to your startup, do you think it will be more common for item requests to be fulfillable or not? Based on that, which type of list would you choose and why?

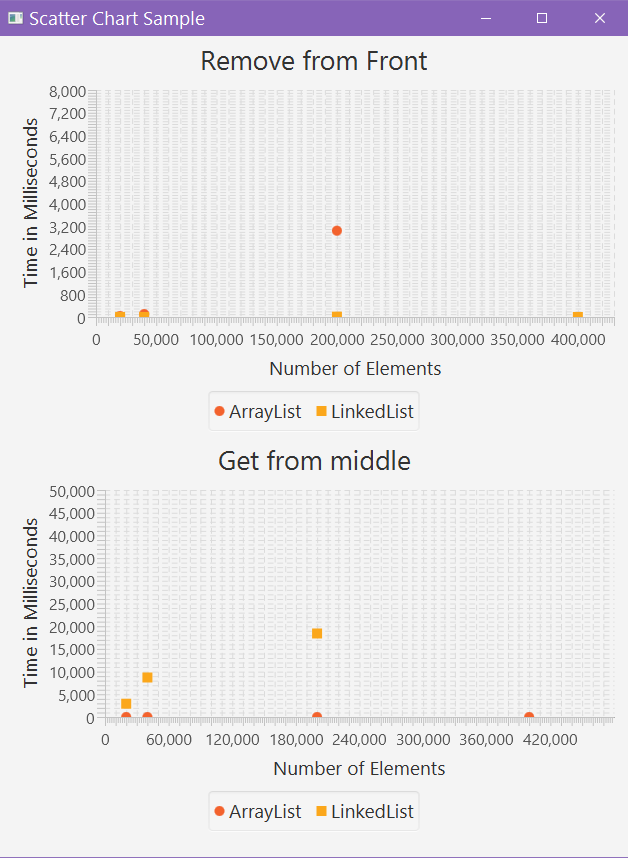
Your answer here

I think it will be more common for item requests fulfillable. Based on that I would choose the ArrayList because it performs relatively better than the linked list in both cases of being fulfillable or not.

# Part 3

## Paste screenshots of the two graphs created by your GUI program.

Your answer here



## Is the performance of removeFromFront() similar for the two lists?

Your answer here

I think so, because the values especially for the first two points are right on top of each other and it was quite difficult to find a good enough range for the graph to make a clear enough demarcation between the two.

## Is the performance of getMiddleBenchmark() similar for the two lists?

Your answer here

The performance is only slightly similar because there are some very noteworthy differences there compared to the performance similarities with the remove from front method.

## How do the differences in performance of the removeFrontBenchmark() and getMiddleBenchmark() methods for the two list types explain some of the performance differences of the WaitingList.nextRequest() method?

Your answer here

The differences in the performance of the removeFrontBenchmark() and getMiddleBenchmark() methods explains the time differences in the WaitingList.nextRequest() method because with those benchmarks it is evident that the ArrayList take more time to remove from front because of the shifts of positions as compared to the LinkedList in that case. It also explains the ArrayList takes less time than to get from middle than the LinkedList.