# **Lab 5\_1**

# **Looking at Java’s ArrayList and LinkedList classes**

# **Week beginning 9th October 2023**

# The following table is from Thinking In Java 4 by Bruce Eckel (<http://sd.blackball.lv/library/Thinking_in_Java_4th_edition.pdf>) in

# Choosing between Lists in Chapter 11.

# It gives the time in milliseconds for multiple runs of various methodsof ArrayList and LinkedList.

---------------------ArrayList ---------------------

size add get set iteradd insert remove

10 121 139 191 435 3952 446

100 72 141 191 247 3934 296

1000 98 141 194 839 2202 923

10000 122 144 190 6880 14042 7333

---------------------LinkedList ---------------------

size add get set iteradd insert remove

10 182 164 198 658 366 262

100 106 202 230 457 108 201

1000 133 1289 1353 430 136 239

10000 172 13648 13187 435 255 239

(Note: Example of application: using ArrayList/LinkedList to store a dictionary in spelling check)

In this lab we will write some code to compare ArrayList and LinkedList. We will compare the get() method of both List types.

What is the Big Oh value for the get() method of each List type?

In your code, populate an ArrayList and a LinkedList with data from a large dataset.

There are datasets available to download from github (and elsewhere) e.g.

<https://github.com/dwyl/english-words/blob/master/words_alpha.zip>

Download this file – Choose View Raw. Save the file to desktop and then extract it to the desktop. This is just a file with 370099 words that you could use as a dictionary. You might come up with a better file to use as a dictionary. Let us know if you do!

Put the file in the root folder of your project in IntelliJ.

In IntelliJ, create a new project and add Lab5.java. Also add the words\_alpha.txt.

Run the program in the usual way and check that it works.

It should output the time for multiple “gets” for ArrayList and LinkedList, initially of size 10. Run it for various size, 10, 100 etc. See table below.

Note: in the code there is a variable reps which you may need to experiment with for your system.

Whatever value you chose for it, note that the same value of reps must be used for all runs of the program.

It should be big enough to allow you to distinguish between the two types of List for all the different sizes, but not so big that the code takes forever to run!

The time for ArrayList may be very small if you run it for small values of reps.

Time the tests on the two types of list. Look at System class for a method that gives you can use to time method calls.

Fill in the following table:

Specify the value you used for reps 1,000,000

ArrayList get() method

|  |  |
| --- | --- |
| size | Time taken |
| 10  100  1000  10000  100000 | 4  3  4  4  7 |

LinkedList get() method

|  |  |
| --- | --- |
| size | Time taken |
| 10  100  1000  10000  100000 | 12  21  362  4271  66004 |

Obtained with Apple M1 processor, 3.2 GHz, Java Version 20.0.2, MacOS Sequoia 15.0.1.

Check if your results are comparable to Eckels. Based on the Big Oh values for the respective get() methods, are the values as you would expect?

Comparison with Eckel's results:

1. ArrayList:

My results show very fast and consistent times (3-7ms) for all sizes, even up to 100,000 elements. This matches Eckel's pattern, where times stayed almost the same (139-144ms) as the list got bigger. My times are much faster, likely because I'm using newer, faster hardware.

2. LinkedList:

My results show a clear increase in time as the list size grows, just like Eckel's. For small lists (10-100 elements), the times are quick, but they increase dramatically for larger lists. This follows the same trend as Eckel's data, though my absolute times differ due to different hardware.

Do the results match what we expect based on Big O values?

1. ArrayList (O(1) for get()):

The results match expectations perfectly. The time stays almost the same (3-7ms) no matter how big the list gets. This is exactly what we expect for O(1) - constant time operations.

2. LinkedList (O(n) for get()):

The results clearly show the expected O(n) behavior. As the list size increases, the time grows significantly. For example, it jumps from 21ms for 100 elements to 66,004ms for 100,000 elements. This matches the idea that in a LinkedList, we need to traverse through more elements to reach our target as the list gets bigger.

In conclusion, these results demonstrate why ArrayList is better for quick access to elements at any position, while LinkedList might be preferred for other operations like adding or removing elements at the beginning or end of the list.