Domonic Lee

Unit 2 Assignment 2

20240616

CS352

(4)Generate the same timing tables that you did for Unit 1. You should see different values. Record these in your Word document.

UNIT 1 Output  
  
Name: Contestant3, Age: 22 Contestant ID: C3, Average Position: 6.0 Name: Contestant4, Age: 23 Contestant ID: C4, Average Position: 5.333333333333333 Name: Contestant10, Age: 29 Contestant ID: C10, Average Position: 4.083333333333333 Name: Contestant7, Age: 26 Contestant ID: C7, Average Position: 4.666666666666667 Name: Contestant9, Age: 28 Contestant ID: C9, Average Position: 2.6666666666666665 Name: Contestant8, Age: 27 Contestant ID: C8, Average Position: 4.25 Name: Contestant1, Age: 20 Contestant ID: C1, Average Position: 5.0 Name: Contestant6, Age: 25 Contestant ID: C6, Average Position: 3.75 Name: Contestant2, Age: 21 Contestant ID: C2, Average Position: 4.5 Name: Contestant5, Age: 24 Contestant ID: C5, Average Position: 4.75 Num Contestants: 43, Num Rounds: 10, Time: 1 ms Num Contestants: 43, Num Rounds: 100, Time: 2 ms Num Contestants: 43, Num Rounds: 1000, Time: 9 ms Num Contestants: 43, Num Rounds: 10000, Time: 99 ms Num Contestants: 45000, Num Rounds: 10, Time: 720 ms

UNIT 2 Output  
  
 Name: Contestant6, Age: 25 Contestant ID: C6, Average Position: 4.5 Name: Contestant9, Age: 28 Contestant ID: C9, Average Position: 4.083333333333333 Name: Contestant7, Age: 26 Contestant ID: C7, Average Position: 3.9166666666666665 Name: Contestant3, Age: 22 Contestant ID: C3, Average Position: 3.9166666666666665 Name: Contestant4, Age: 23 Contestant ID: C4, Average Position: 4.333333333333333 Name: Contestant2, Age: 21 Contestant ID: C2, Average Position: 5.666666666666667 Name: Contestant10, Age: 29 Contestant ID: C10, Average Position: 4.583333333333333 Name: Contestant1, Age: 20 Contestant ID: C1, Average Position: 3.9166666666666665 Name: Contestant8, Age: 27 Contestant ID: C8, Average Position: 5.0 Name: Contestant5, Age: 24 Contestant ID: C5, Average Position: 5.083333333333333 Num Contestants: 43, Num Rounds: 10, Time: 1 ms Num Contestants: 43, Num Rounds: 100, Time: 2 ms Num Contestants: 43, Num Rounds: 1000, Time: 10 ms Num Contestants: 43, Num Rounds: 10000, Time: 108 ms Num Contestants: 45000, Num Rounds: 10, Time: 7976 ms

(5) Compare the timing for using an ArrayList with the same timing for using a LinkedList. Try to explain what you see and make any recommendations. Also, consider your proposed algorithm from Unit 1 that rearranges Contestants in the same list. Based on your timing estimates, explain whether you think it would take more or less time using a LinkedList rather than an ArrayList.

When comparing the performance of an ArrayList and a LinkedList in Java, we need to consider the different operational complexities of these data structures.

Timing Comparison: ArrayList vs. LinkedList

1. Access Time (get operation):

* ArrayList: O(1) - ArrayLists have fast random access because they are backed by arrays.
* LinkedList: O(n) - LinkedLists require traversal from the head to the desired index.

1. Insertion and Deletion (add/remove operation):

* ArrayList: O(n) - Insertions and deletions can be expensive due to the need to shift elements.
* LinkedList: O(1) - Insertions and deletions are generally cheaper if we already have a reference to the node.

1. Iteration:

* ArrayList: O(n) - Iteration is efficient due to cache locality.
* LinkedList: O(n) - Iteration is slower due to cache misses.

Timing Estimates

Given the performance characteristics above, let’s break down the timing provided in UNIT 1 and UNIT 2 in terms of what they might imply about the different data structures:

UNIT 1 (ArrayList timings assumed)

* 43 Contestants, 10 Rounds: 1 ms
* 43 Contestants, 100 Rounds: 2 ms
* 43 Contestants, 1000 Rounds: 9 ms
* 43 Contestants, 10000 Rounds: 99 ms
* 45000 Contestants, 10 Rounds: 720 ms

UNIT 2 (LinkedList timings assumed)

* 43 Contestants, 10 Rounds: 1 ms
* 43 Contestants, 100 Rounds: 2 ms
* 43 Contestants, 1000 Rounds: 10 ms
* 43 Contestants, 10000 Rounds: 108 ms
* 45000 Contestants, 10 Rounds: 7976 ms

Explanation and Recommendations

1. Small Data Sets (43 Contestants):

* For smaller data sets, both ArrayList and LinkedList perform similarly, as seen from the timings of 1 ms to 2 ms for 10 and 100 rounds. The cost of access operations is not significant enough to make a substantial difference.

1. Larger Data Sets (45000 Contestants):

* For larger data sets, the difference becomes significant. The ArrayList performs much better (720 ms vs. 7976 ms) because LinkedList has a higher overhead due to frequent node traversal and poor cache performance.

Conclusion

For the algorithm that rearranges contestants:

* ArrayList would be more efficient due to faster access times and better iteration performance.
* LinkedList would incur significant overhead due to its linear time complexity for accessing elements by index.

For operations involving frequent access and rearrangement of elements, such as sorting or complex rearrangement algorithms, ArrayList is generally the better choice due to its faster access time. Use LinkedList if the primary operations involve frequent insertions and deletions at the beginning or middle of the list, and the data set is not excessively large.

(6)

What did you learn?

I learned about the performance differences between ArrayList and LinkedList in Java, particularly how access times, insertion, and deletion operations compare, and how these differences impact the efficiency of rearranging elements within the list.

What did you like about this project?

I liked the practical application of comparing data structures using real-world data and timing metrics, which provided clear insights into the performance implications of using different list implementations in Java.

What did you find confusing or would like to see done differently regarding this project?

I found it a bit confusing that the dataset was not explicitly labeled as being used with either ArrayList or LinkedList. It would be helpful to clearly specify which data structure is being referred to in each unit to avoid any ambiguity.

If you had another hour or two, what would you like to add to the project or how would you do things differently?

If I had another hour or two, I would add a detailed benchmarking analysis that includes a wider range of operations (e.g., sorting, shuffling) and a comparison of memory usage between ArrayList and LinkedList. Additionally, I would implement and test the proposed rearrangement algorithm on both data structures to provide empirical evidence supporting the theoretical performance differences.