Homework 5

D. Ivan Ochoa

Kerby

CS 2235

As the data shows for a Linked List: adding to the front of the linked list is roughly equivalent to O(1), and adding to the last, as well as accessing the middle, is roughly equivalent to O(n). However, removing from the front was extremely quick (roughly O(1) while removing from the back was O(n).

The data for an Array List shows accessing the Middle and the Back was slightly quicker than a Linked List, although roughly O(n), while adding from the front was far slower than a Linked List and removing from any position was the slowest for all three of the lists. This is presumably due to the fact that accessing an index within an array (add(int, object)) is on the order of O(n) and removing from the list takes far longer since the array must shift all the elements in order to compensate and correct the list size (presumably, the index accessing is O(1)[?] but the shifting is what causes the O(n)).

The data for the Deque shows that it was far quicker than either of the other two lists in both insertion and deletion, in either the front or the back and thus, was O(1). This even held true when clearing the entire list, all together. However, the downside being that one cannot access the middle of the list or any other location that isn’t the front/back, which can be a huge disadvantage.

Given the data, a deque would excel at simply adding and removing elements from the list, in an ordered manner, which would possibly apply to larger, more static[?] data structures. However, if a dynamic list is needed, hence adding, removing, and accessing from anywhere other than the front or back, then an Array List or Linked List are better suited.

According to the data, an Array List, although just as capable as a Linked List, might be better for only accessing and utilizing data – similar to the Deque but much more capable since the index not only allows access to the middle but also access to any random location within the array, given the index location is known. However, a great drawback to an Array List is its use in modifying data, i.e., insertion/deletion. The data shows that modifying the front of an Array List is slower than a Linked List, modifying the middle and end of the list is similar to a Linked List, and removing from any position is far slower than a Linked List.

Therefore, an Array List is better suited to work with larger static structures that seldom require modifications. However, unlike a Deque, the Array List is capable of not only accessing the middle but also random locations within the list and is therefore, better suited than a Deque would be.

A Linked List has the ability to add from the front, and remove from the front or back, on the order O(1). However, accessing the middle and adding from the back is on the order O(n) and slower than the equivalent method for an array. Because of this, a Linked List is better suited for more dynamic structures that will require heavy modification. However, without a pointer, the list is privy to modifying (i.e., insertion/deletion), exclusively, the front and back. Even with a pointer, accessing the middle and random locations within the structure is potentially slower than an Array List, although with the capability if being faster when the location is known.

Even then, if the dynamic structure, one is working with, will only be accessed from the front, back, and middle, exclusively, then a Linked List will be the better suited choice.