David S. Anderson

Md Ashiqur Rahman

CSc 210 – Summer 2020

16 July 2020

PA6 – Written Portion

1. ArrayList vs. LinkedList

1a. Having instance variables to track both the front and the back of the list would be helpful because Stacks and Queues perform quite a few operations on either the front (top) or back (bottom), respectively. Tracking which element is at the top of the stack improves the pop() and peek() methods for the Stack classes. Alternatively, tracking the element at the bottom of a Queue (the first element in and hence the first element that will be removed) makes it easier to perform pop() and peek() in those classes. Finally, it would be helpful to check the front and back of our list when iterating over all of the items because we would know when to stop; for example, if the two instance variables are equal, then we know it is the last element in the list.

1b.

Method	ArrayList	LinkedList
add(int value)	O(1)	O(1)
add(int index, int value)	O(n/2)	O(n/4)
clear()	O(1)	O(1)
contains(int value)	O(n)	O(n)
get(int index	O(1)	O(n)

isEmpty()	O(1)	O(1)
remove(int index)	O(n/2)	O(n/4)
toString()	O(n)	O(n)
equals(Object O)	O(1)	O(1)

1c. Using the above table, it is clear that ArrayList and LinkedList have certain features that make one advantageous over the other depending on the situation. One of the biggest benefits of ArrayList is the run-time efficiency of its get() method; with a Big-O of O(1), it is far more efficient than LinkedList in this regard and would therefore be preferable in situations where a user would be repeatedly retrieving data from the List. This was a big reason why ArrayLists were so useful during our Spotify replica programming assignment.

On the other hand, the LinkedList implementation is more efficient in general when inserting an item at a particular index and removing an item at a specific index. As a doubly linked list, it allows it to simply use the pointer to switch and point to the next object, rather than shifting every item in the list.

1d. If I knew that I would need to be inserting multiple elements into positions other than the front or the back of my list, I would use a LinkedList instead of an ArrayList. As was mentioned in the previous segment, the LinkedList implementation allows elements to use their pointer to simply point to the next item rather than shifting all elements. For example, if I were managing user information in a list of profiles, but I knew I would regularly need to remove members or update their information and re-insert them into the list, A LinkedList would make more sense due to its remove and insert efficiency.

2. ArrayStack vs. ListStack

2a

Method	ArrayStack	ListStack
push(int value)	O(1)	O(n)
pop()	O(n)	O(n)
peek()	O(1)	O(n)
isEmpty()	O(1)	O(1)
size()	O(1)	O(1)
clear()	O(1)	O(1)
toString()	O(n)	O(n)
equals(Object O)	O(n)	O(n)

2b Using the above table and my knowledge about how Stacks are used (as well as how I implemented both in this assignment), I would implement a Stack with an array as its backing data structure. Since I can use the growArray() method to increase the size of my array, an array makes it more efficient to "push" items onto the stack and to peek at an element without removing it. These are two of the most important features of the Stack data structure.

3. ArrayQueue vs. ListQueue

3a

Method	ArrayQueue	ListQueue
enqueue(int value)	O(n)	O(n)
dequeue()	O(n)	O(1)
peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
size()	O(1)	O(1)
clear()	O(1)	O(1)
toString()	O(n)	O(n)
equals(Object O)	O(n)	O(n)

3b If I were creating a new programming language and writing a Queue class, I would choose to use a linked list as the backing data structure for my queue. When using my own linked list with individual nodes, it is quite efficient to track and retrieve the first item in the list, because that element always stays with the "first" instance variable and successive elements are attached after it. This makes it much easier to dequeue the correct element while maintaining the order and not needing to "grow" the list each time an item is removed. It becomes far simpler to implement the Queue's First In, First Out (FIFO) structure.