Xin Li
October 21, 2020

1a. This is helpful because if we want to access the back of our list and only have to track the front of our list, the time complexity for going through the list is O(n). But if we have an instance variable tracking the back of our list, we can return it and the time complexity will be O(1).

1b.

Method	ArrayList	LinkedList
add(int value)	O(1)	O(1)
add(int index, int value)	O(index)	O(index)
clear()	O(n)	O(1)
contains(int value)	O(n)	O(n)
get(int index)	O(1)	O(index)
isEmpty()	O(1)	O(1)
remove(int index)	O(n)	O(index)
toString()	O(n)	O(n)
equals(Object o)	O(n)	O(n)

1c. According to the table above, I would use ArrayList to get the element with the index since its time complexity of the get function is O(1) and the LinkedList is O(n). This is because LinkedList doesn't keep track of the index, it just links things together in a straight form.

1d. I would use LinkedList if I want to add or remove the front element or the back element in the list. This is because the time complexity of this is O(1) since we have an instance variable tracking the front and the back list so we don't have to go through the whole list to find it. Also, after removing the front element from the ArrayList, we have to shift the elements left.

2a.

Method	ArrayStack	ListStack
push(int value)	O(1)	O(1)
pop()	O(1)	O(1)

peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
size()	O(1)	O(1)
clear()	O(n)	O(1)
toString()	O(n)	O(n)
equals(Object o)	O(n)	O(n)

2b. According to the table above, I will choose a linked list as its backing data structure because the time complexity of pop and clear function is O(1), which are better than ArrayStack.

3a.

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

3b. According to the table above, I will also choose a linked list as its backing data structure for the queue, because of the same reason as above. For the ArrayQueue, dequeue() is O(n) because it has to shift left after removing the front element. The clear() function is also O(n) because it has to initialize a new array, instead just set the head to null like the linked list.