**ArrayList and LinkedList Exercises**

1. What is the time complexity of the following code – best case, worst case and amortized complexity. Compare the 2.

**public** **boolean** **add**(E element) {  
**if** (size >= array.length) {  
// make a bigger array and copy over the elements  
E[] bigger = (E[]) **new** Object[array.length \* 2];  
System.arraycopy(array, 0, bigger, 0, array.length);  
array = bigger;  
}  
array[size] = element;  
size++;  
**return** **true**;  
}

O(1) the size does not need to get bigger hence why o1

Worst case: O(n) we don’t know how many times we need to incess our array to insert it in

**public** **boolean** **add**(E element) {  
**if** (size >= array.length) {  
// make a bigger array and copy over the elements  
E[] bigger = (E[]) **new** Object[array.length + 10];  
System.arraycopy(array, 0, bigger, 0, array.length);  
array = bigger;  
}  
array[size] = element;  
size++;  
**return** **true**;  
}

Worst case: O(n) we don’t know how many times we need to incess our array to insert it in

1. What is the best case and worst case complexity of following code. Explain in your own words why?

MyArrayList:  
**public** **boolean** **removeAll**(Collection<?> collection) {  
**boolean** flag = **true**;  
**for** (Object obj: collection) {  
flag &= remove(obj);  
}  
**return** flag;  
}

Best case: O(1) if the size is null

Worst case o(n\*m) , because O(n) to remve the object then it needs to shift the array. M being the element

1. Look at the add function in MyLinkedList and explain in your own words what it is doing? Compare it with add function in MyArrayList (from last week exercise).

In the ljnked list we have to incress the size manually, and add node if the index is 0. Also link the nodes.

In the array list , we ofc check the size if its out of bounds, then we shift the size and add the element.

1. Compare the complexity of the functions written by you (or solutions from author) with following implementations of the functions. What do you observe explain? Are these functions efficient?

@Override  
public int indexOf(Object target) {  
 //*TODO: FILL THIS IN!* for(int i = 0; i < size; i++) {  
 if(equals(target, get(i))) {  
 return i;  
 }  
 }  
 return -1;  
}

@Override **public** **void** **add**(**int** index, E element) {  
 //**TODO:** FILL THIS IN!  
 E old;  
 **for** (**int** i=index; i<size; i++) {  
 old = set(i, element);  
 element = old;  
 }  
 add(element);  
}

@Override **public** E **remove**(**int** index) {  
 //**TODO:** FILL THIS IN!  
 E element = get(size-1);  
 **for**(**int** i = size-2; i>=index; i--) {  
 element = set(i, element);  
 }  
 size--;  
 **return** element;  
}

1. Choose appropriate data structure: arraylist or linkedlist for following use cases:
   1. Storing student records

arraylist

* 1. Browser history where new pages visited are saved

Doublelinkedlisted

* 1. Leaderboard in a game

arraylist

* 1. Printer job queue

linkedlist

* 1. Playlist

arraylist

* 1. Undo/redo feature in text editor

doublelinkedlist

1. Why does doubly linkedlist have constant time for both adding and removing elements. Explain?  
   because we know the reference it has to the last and fitst. So its always constant
2. Extend the code in MyLinkedList with tail pointer. What does this mean?