# COMP 2402 Midterm Review

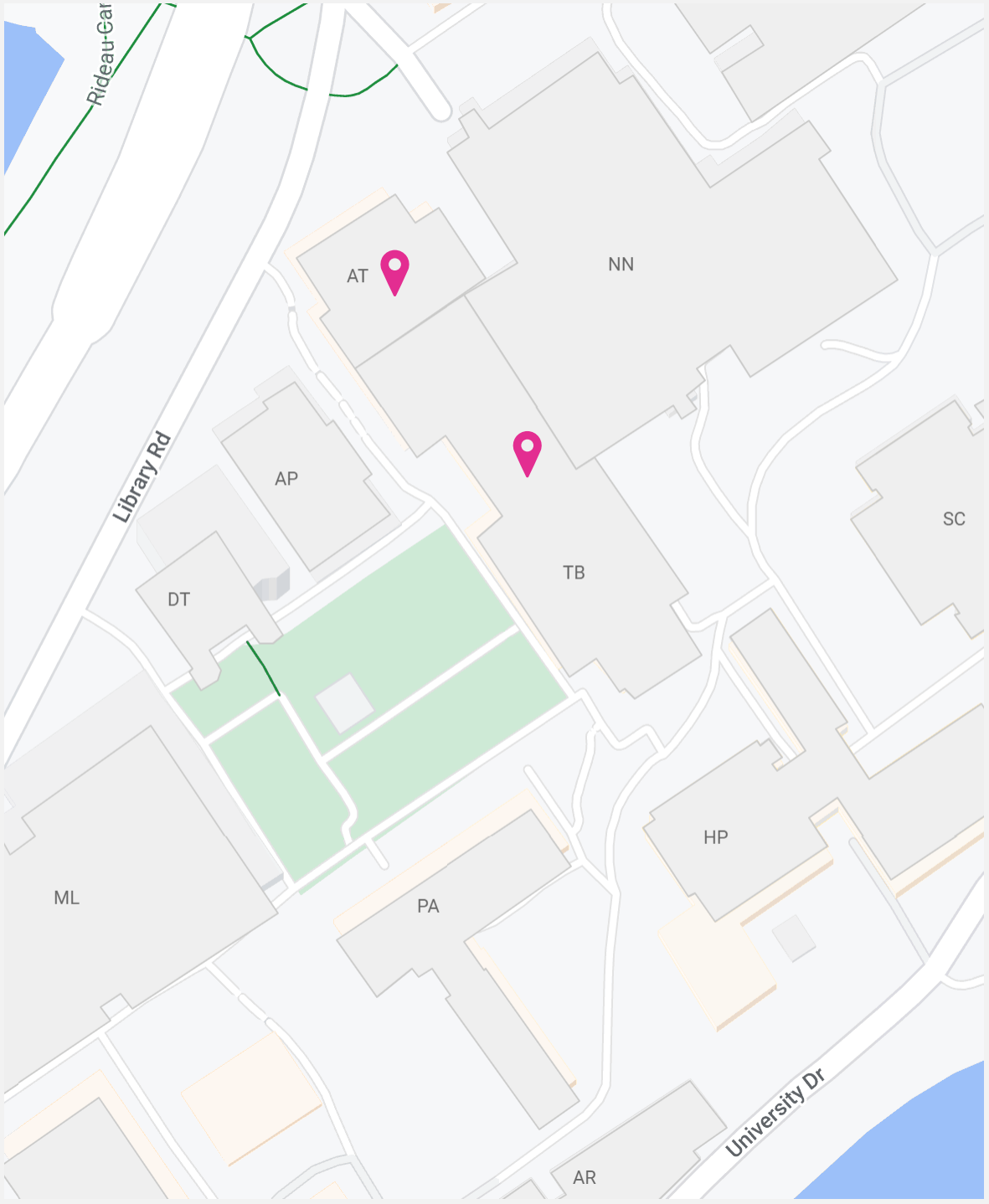
Carleton University
Alina Shaikhet

# Midterm

- October 21,  9:00 – 10:20 am
- In person – Azrieli Theater, Tory Building
- Make sure you sit in “your” room
- Midterm is **15%** of your final grade

Bring:

- **Student id**
- Pencil HB, sharpener, eraser
- Water Bottle



# Midterm

- 30 Multiple Choice Questions

- Questions document
- Scantron – write your answers there

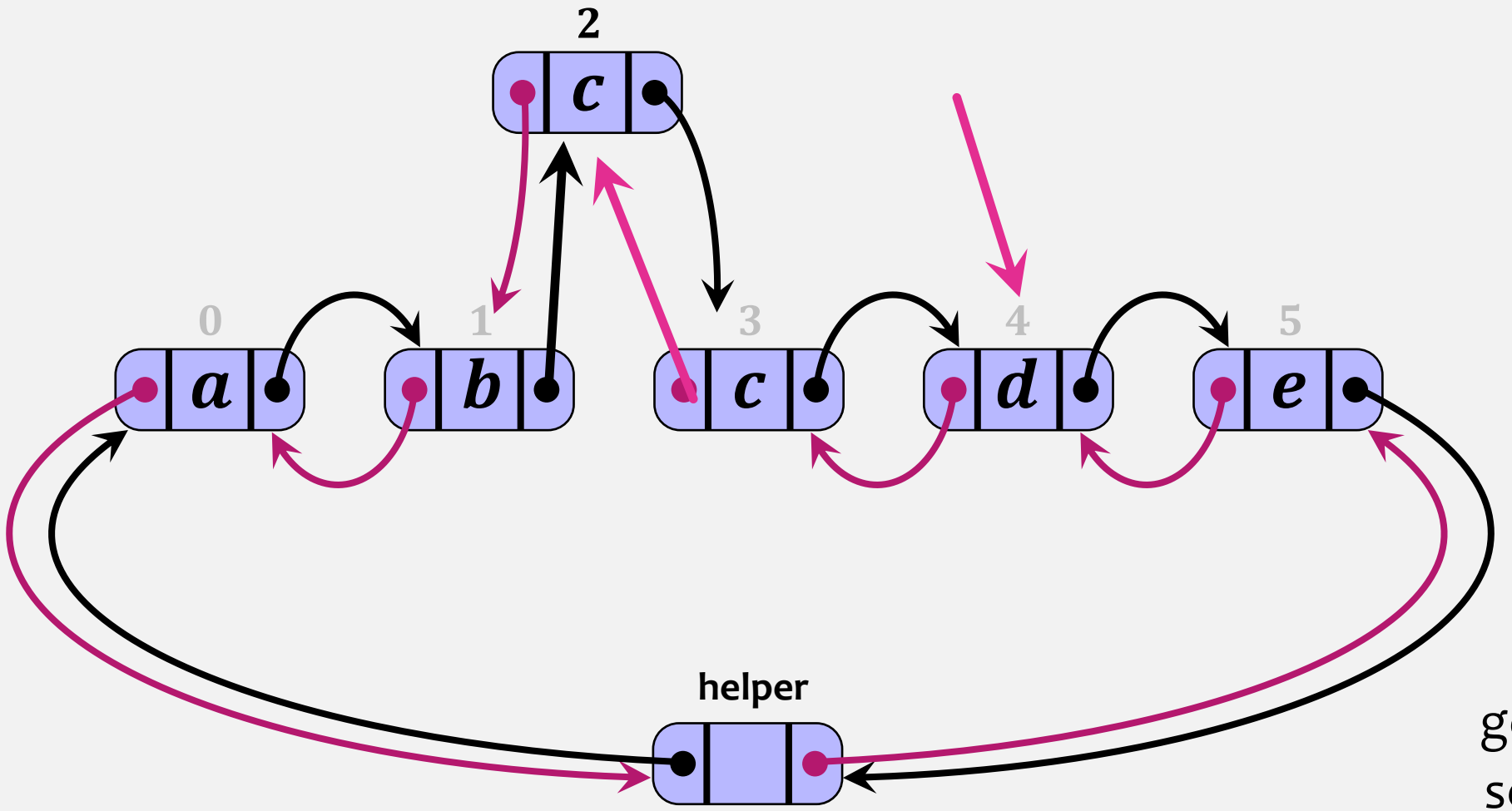
Put your name and id on both documents.
You will have to submit them before you leave.

# Topics

Everything we studied in this course up to and including “Binary Trees”.

This also includes assignments and weekly quizzes.

Some of the questions for the midterm were taken directly from your quizzes




get($i$)
set($i$,$x$)
add($i$,$x$)
remove($i$)

$$O\left(1+\min(i, n-i)\right)$$

# Java Collections Framework Reference

| List | | | | |
|---|---|---|---|---|
| | `get(i)` | `set(i,x)` | `add(i,x)` | `remove(i)` |
| `ArrayList` | $O(1)$ | $O(1)$ | $O(1+n-i)$ | $O(n-i)$ |
| `LinkedList` | $O(1+\min\{i,n-i\})$ | | | |

| Deque | | | | |
|---|---|---|---|---|
| | `addFirst(x)` | `removeFirst()` | `addLast(x)` | `removeLast(x)` |
| `ArrayDeque` | $O(1)$ | | | |
| `LinkedList` | $O(1)$ | | | |

| Queue | | | |
|---|---|---|---|
| | `add(x)` | `remove()` | `element()` |
| `ArrayDeque` | $O(1)$ | | |
| `LinkedList` | $O(1)$ | | |
| `PriorityQueue` | $O(\log n)$ | $O(\log n)$ | $O(1)$ |

# Java Collections Framework Reference

| Set | | | |
|---|---|---|---|
| | `add(x)` | `remove(x)` | `contains(x)` |
| `HashSet` | $O(1)$ | | |
| `TreeSet` | $O(\log n)$ | | |

| SortedSet | | | |
|---|---|---|---|
| | `headSet(y)` | `tailSet(x)` | `subSet(x,y)` |
| `TreeSet` | $O(\log n)$ | | |

| Map | | | |
|---|---|---|---|
| | `get(k)` | `put(k,v)` | `containsKey(k)` |
| `HashMap` | $O(1)$ | | |
| `TreeMap` | $O(\log n)$ | | |

# Java Collections Framework Reference

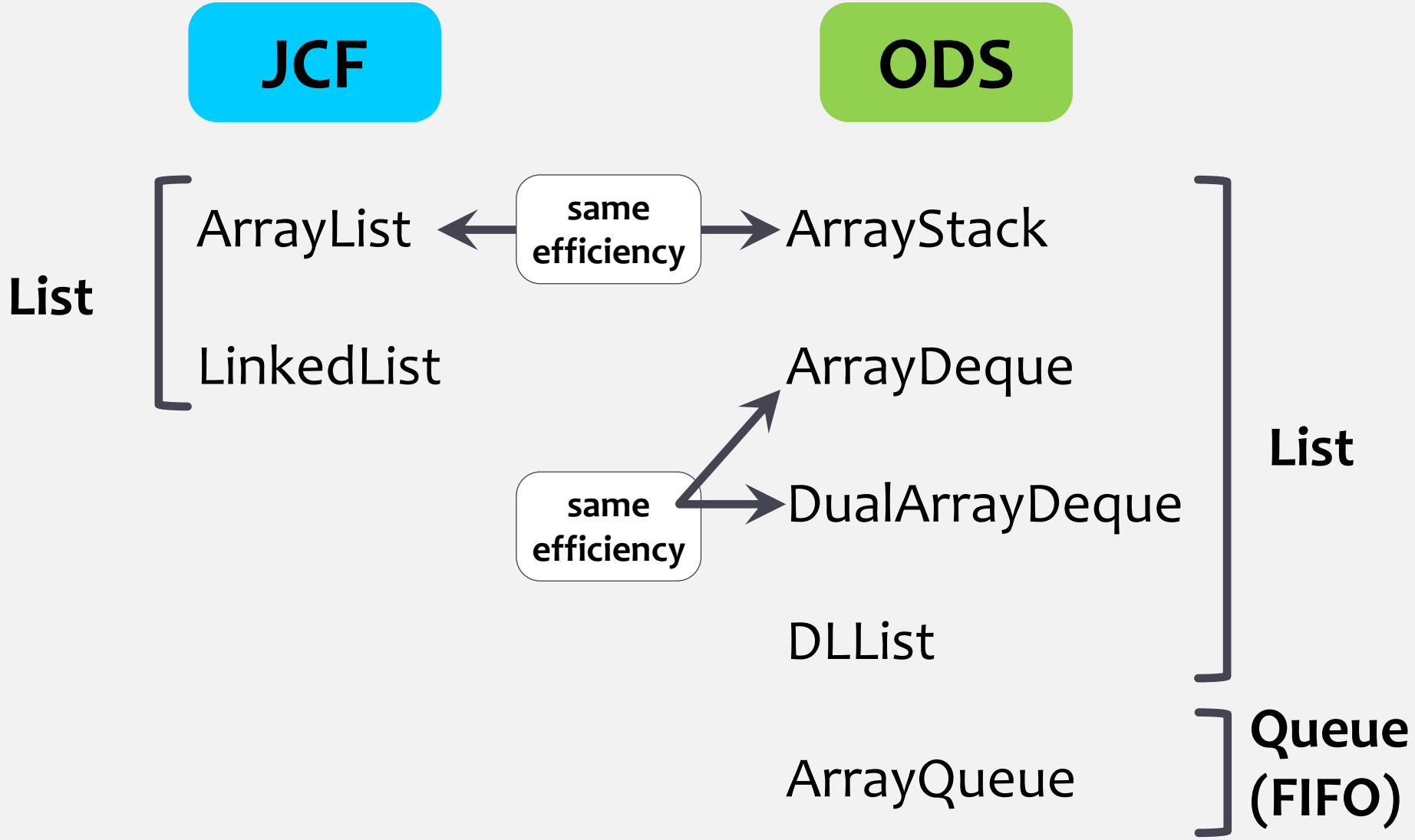
class

| Collections | | | |
|---|---|---|---|
| `sort(list)` | `min(c)/max(c)` | `reverse(list)` | `binarySearch(list,x)` |
| $O(n \log n)$ | $O(n)$ | $O(n)$ | $O(\log n)$ |

Collections.sort(mylist); ← Sorts the specified list into ascending order

https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html

# Overview


JCF
ODS
List
ArrayList
LinkedList
same efficiency
ArrayStack
ArrayDeque
same efficiency
DualArrayDeque
DLList
List
ArrayQueue
Queue (FIFO)

# Q1

Suppose that the ArrayStack implementation were modified in the following way:

`resize()` resizes the backing array so that its size is $\boldsymbol{n + 9}$

Which of the following statements is true?

a) This has zero impact on the amortized runtime of `add()` and `remove()`

b) This drastically (asymptotically) worsens the amortized runtime of `add()` and `remove()`

c) This drastically (asymptotically) improves the amortized runtime of `add()` and `remove()`

d) This mildly (by a constant) worsens the amortized runtime of `add()` and `remove()`

e) This mildly (by a constant) improves the amortized runtime of `add()` and `remove()`

# Q2

```
public static void reverse(List<Integer> l1) {
   List<Integer> l2 = new List<>();
   int n = l1.size();
   for( int i=0; i < n; i++ ) {
      int next = l1.remove(l1.size()-1);
      l2.add(next);
   }
}
```

The above method is

a) much faster when `l1` and `l2` are `LinkedLists`
b) much faster when `l1` and `l2` are `ArrayLists`
c) about the same speed independent of whether `l1` and `l2` are both `ArrayLists` or `LinkedList`

# Q3

```
public static void reverse2(List<Integer> l1) {
   List<Integer> l2 = new List<>();
   int n = l1.size();
   for( int i=0; i < n; i++ ) {
      int next = l1.remove(0);
      l2.add(0,next);
   }
}
```

The above method is

a) much faster when `l1` and `l2` are `LinkedLists`
b) much faster when `l1` and `l2` are `ArrayLists`
c) about the same speed independent of whether `l1` and `l2` are both `ArrayLists` or `LinkedList`

# List implementations

| | get(i) / set(i,x) | add(i,x) / remove(i) |
|---|---|---|
| ArrayList (JCF) | $O(1)$ | $O(1+n-i)$ |
| ArrayStack (ODS) | | |
| RootishArrayStack (ODS) | | |
| ArrayDeque (ODS) | $O(1)$ | $O(1+\min\{i,n-i\})$ |
| DualArrayDeque (ODS) | | |
| LinkedList (JCF) | $O(1+\min\{i,n-i\})$ | |
| DLList (ODS) | | |

fast at one end

fast at both ends

dynamic

# Q4

Worse runtime means asymptotically worse.

Recall that a `DualArrayDeque` implements the `List` interface using two `ArrayStacks`. Suppose we replace the `ArrayStacks` with `DLLists` so that the `Stack`, `Deque`, and `List` implementation of `front` and `back` is pointer-based (using doubly linked lists) instead of array-based:

```
public class DualArrayDeque<T> extends AbstractList<T> {
   DLList<T> front;
   DLList<T> back;
        ...
}
```

In the choices below, `Deque` operations are `addFirst`, `addLast`, `removeFirst`, and `removeLast`; `List` operations are `get`, `set`, `add`, and `remove`. Which of the following statements is true?

a) This worsens the asymptotic runtimes of some of the DualArrayDeque's Deque operations, and some of the List operations.
b) This has no impact on the asymptotic runtimes of the DualArrayDeque's Deque operations nor on its List operations.
c) This has no impact on the asymptotic runtimes of the DualArrayDeque's Deque operations, but it worsens the runtimes of some of its List operations.
d) This worsens the asymptotic runtimes of some of the DualArrayDeque's Deque operations, but has no impact on the runtimes of the other List operations.

# Q5

Both the RootishArrayStack and the ArrayDeque implement the List interface.

If you choose to use a RootishArrayStack over an ArrayDeque as your implementation, you have:

a) improved runtime for add(i,x) and remove(i), no change on space usage

b) no change on runtime for add(i,x) and remove(i), improved space usage

c) worse runtime for add(i,x) and remove(i), no change on space usage

d) no change on runtime for add(i,x) and remove(i), no change on space usage

e) worse runtime for add(i,x) and remove(i), improved space usage

# Q6

For the get(i), set(i,x), add(i,x), and remove(i) List operations, both the array- and pointer-based implementations

1. first find position ***i*** of the list, either in the array or the linked list, then
2. perform the appropriate operation at position ***i*** and anywhere else necessary.

Which of the following statements is true when comparing the two implementations?

a) Arrays spend more time than linked lists in part (1), and less time in part (2)

b) Linked lists spend more time than arrays in part (1), and less time in part (2)

c) Linked lists and arrays spend about the same amount of time in both part (1) and part (2)

d) Linked lists spend more time than arrays in both part (1) and part (2)

e) Arrays spend more time than linked lists in both part (1) and part (2)

# Q7



When comparing the DLList and DualArrayDeque implementations of the Deque interface, which of the following is true?

a) To store $\boldsymbol{n}$ elements, the DLList and DualArrayDeque implementations use $\boldsymbol{O(1)}$ and $\boldsymbol{O(n)}$ extra space, respectively
b) To store $\boldsymbol{n}$ elements, the DLList and DualArrayDeque implementations use $\boldsymbol{O(n)}$ and $\boldsymbol{O(1)}$ extra space, respectively
c) To store $\boldsymbol{n}$ elements, both the DLList and DualArrayDeque implementations use $\boldsymbol{O(n)}$ extra space
d) To store $\boldsymbol{n}$ elements, both the DLList and DualArrayDeque implementations use $\boldsymbol{O(1)}$ extra space

# Q8

Suppose you are given an input file. Your goal is to read the input one line at a time and output each line except if the current line is the same as the previous one. That is, if there are consecutive duplicate elements, you only print out the first of the (consecutive) sequence.

Choose the best (most time- and space-efficient) data structure to solve this problem.

a) A single variable (in addition to loop iteration variables) suffices for data storage
b) A stack (or, deque) is the best choice for data storage
c) No data storage at all is necessary (other than to store loop iteration variables)
d) A deque (but not a stack or a queue) is the best choice for data storage (this is the deque as we did in class, that implements the List interface)
e) A sorted set is the best choice for data storage

# Q9

Suppose you are given an input file and a parameter $x$. Your goal is to read the input one line at a time and output the first $x$ distinct lines in sorted order.

Choose the best (most time- and space-efficient) data structure to solve this problem.

a) No data storage at all is necessary (other than to store loop iteration variables)

b) A single variable (in addition to loop iteration variables) suffices for data storage

c) A stack (or, deque) is the best choice for data storage

d) A deque (but not a stack or a queue) is the best choice for data storage (this is the deque as we did in class, that implements the List interface)

e) A sorted set is the best choice for data storage

# Q10

Recall that the length of a path in a tree is measured in the number of edges.

What is the maximum possible length of a simple path in a binary tree on $n$ nodes?

a) $2n$

b) $n$

c) $n - 1$

d) $2 \log_2 n$

e) $\log_2 n$

# Quizzes

Suppose we start with an empty list and then repeatedly add elements. Let $\boldsymbol{r}$ be the number of arrays used in our list implementation. Consider the following 5 statements. Which of them are true when $\boldsymbol{r}$ is fixed, and our goal is to have $\boldsymbol{O(n)}$ wasted space and $\boldsymbol{O(1)}$ amortized `resize`?

1. Repeated adds will sooner or later fill up the $\boldsymbol{r}$ arrays.
2. To make room for the adds, we must resize $\geq 1$ of our $\boldsymbol{r}$ arrays to a bigger size.
3. Resizing an array on add operation requires making a new (bigger) array, copying the $\boldsymbol{n}$ array elements from the original to the new array, and takes $\boldsymbol{O(n)}$ time.
4. We need at least $\boldsymbol{O(n)}$ adds between consecutive resizes.
5. We need our resize to increase the array size by at least $\boldsymbol{O(n)}$ spaces, which wastes $\boldsymbol{O(n)}$ space.

a) They are all true.

b) All but (2) are true.

c) All but (2), and (4) are true.

d) All but (2), (4) and (5) are true.

e) All but (4) and (5) are true.

# Quizzes

Suppose we start with an empty list and then repeatedly add elements. Let $\boldsymbol{r}$ be the number of arrays used in our list implementation. Consider the following 5 statements. Which of them are true when $\boldsymbol{r}$ is fixed, and our goal is to have $\boldsymbol{O(n^2)}$ wasted space and $\boldsymbol{O(1)}$ amortized `resize`?

1. Repeated adds will sooner or later fill up the $\boldsymbol{r}$ arrays.
2. To make room for the adds, we must resize $\geq 1$ of our $\boldsymbol{r}$ arrays to a bigger size.
3. Resizing an array on add operation requires making a new (bigger) array, copying the $\boldsymbol{n}$ array elements from the original to the new array, and takes $\boldsymbol{O(n)}$ time.
4. We need at least $\boldsymbol{O(n)}$ adds between consecutive resizes.
5. We need our resize to increase the array size by at least $\boldsymbol{O(n)}$ spaces, which wastes $\boldsymbol{O(n)}$ space.

a) They are all true.

b) All but (2) are true.

c) All but (2), and (4) are true.

d) All but (2), (4) and (5) are true.

e) All but (4) and (5) are true.

# Quizzes

Suppose we start with an empty list and then repeatedly add elements. Let $r$ be the number of arrays used in our list implementation. Consider the following 5 statements. Which of them are true when $r$ is fixed, and our goal is to have $O(n)$ wasted space and $O(n)$ amortized `resize`?

1. Repeated adds will sooner or later fill up the $r$ arrays.
2. To make room for the adds, we must resize $\geq 1$ of our $r$ arrays to a bigger size.
3. Resizing an array on add operation requires making a new (bigger) array, copying the $n$ array elements from the original to the new array, and takes $O(n)$ time.
4. We need at least $O(n)$ adds between consecutive resizes.
5. We need our resize to increase the array size by at least $O(n)$ spaces, which wastes $O(n)$ space.

a) They are all true.

b) All but (2) are true.

c) All but (2), and (4) are true.

d) All but (2), (4) and (5) are true.

e) All but (4) and (5) are true.


Good Luck!