COMP 2402A Midterm

May 29, 2025

- **ALL ANSWERS SHOULD BE WRITTEN IN THE DESIGNATED SPACE.** The last two pages are for rough work. If you answer any questions on these pages, be sure to clearly indicate it.
- This is a closed book test. Basic, non-graphing calculators are permitted but smartphones, laptops, or other aids are not.

Marking Scheme: This midterm is out of 25 marks.

- Question 1 is worth 10 marks.
- Question 2 is worth 5 marks.
- Question 3 is worth 4 marks.
- Question 4 is worth 6 marks.

1. We define a *fundamental operation* to be either:

   - copying a single piece of data from one location to another (as one would do with an array), or
   - updating a reference to traverse from one node to another (as one would do with a linked list).

   With the assumption that all of the structures mentioned below implement the *List* interface and contain 1000 data items, specify the number of fundamental operations you would need (rounded to the nearest 100) for each of the following method calls as they have been presented in class (**Note**: 49 or less rounded would be rounded to 0, 50-99 would be rounded to 100). **DO NOT GIVE BIG-O NOTATION**.

   (a) Calling add(800, 'x') on an **Singly-Linked List**.

   (b) Calling remove(700) on a **Doubly-Linked List**.

   (c) Calling get(100) on an **ArrayStack**.

   (d) Calling add(400, 'x') on an **ArrayDeque**.

   (e) Calling add(700, 'x') on an **DualArrayDeque** where front.size() = 400 and back.size() = 600.

(f) Calling set(300, 'x') on an **DualArrayDeque** where front.size() = 600 and back.size() = 400.

(g) Calling add(700, x) on a **SEList** (space efficient list) with block size `b = 400`.

(h) Calling add(100, x) on a **BlockedList** (from Assignment 2) with block size `b = 200`, i.e., this **BlockedList** would have 5 blocks. **Note:** adding or removing a block is not a fundamental operation and would not be counted.

2. Using the code provided below (adapted from our in-class examples and from the official textbook), be prepared to implement the any of the methods in the Deque or List interfaces, such as `remove(i)` method. (n.b., When you see '...' in the body of a method, you can assume that the method has been completely and correctly implemented. You may **ONLY** call the methods shown here.)

```
public class ArrayDeque<T> {
    // The circular array used to store the elements
    protected T[] array;
    protected int j; // The index of the front element
    protected int n; // The number of elements in the deque
    // The values of j and n are initialized to 0
    public ArrayDeque(Class<T> t) { ... }
    public int size() { return n; }
    protected void resize() { ... } // Resizes the array to 2*n
}
```

`public T remove(i)`

remove(i) continued...

3. If you were trying to specify the location of an element with index i in a **RootishArrayStack** created to hold n elements, you would need to specify both the block number b and the corresponding index j (into that block b). Recalling that b is the smallest integer such that...

$$\frac{(b+1)(b+2)}{2} \geq i+1$$

...and that if index i is in block b, then the number of elements in blocks $0, 1, \ldots, b-1$ is...

$$\frac{b(b+1)}{2}$$

...answer the following question:

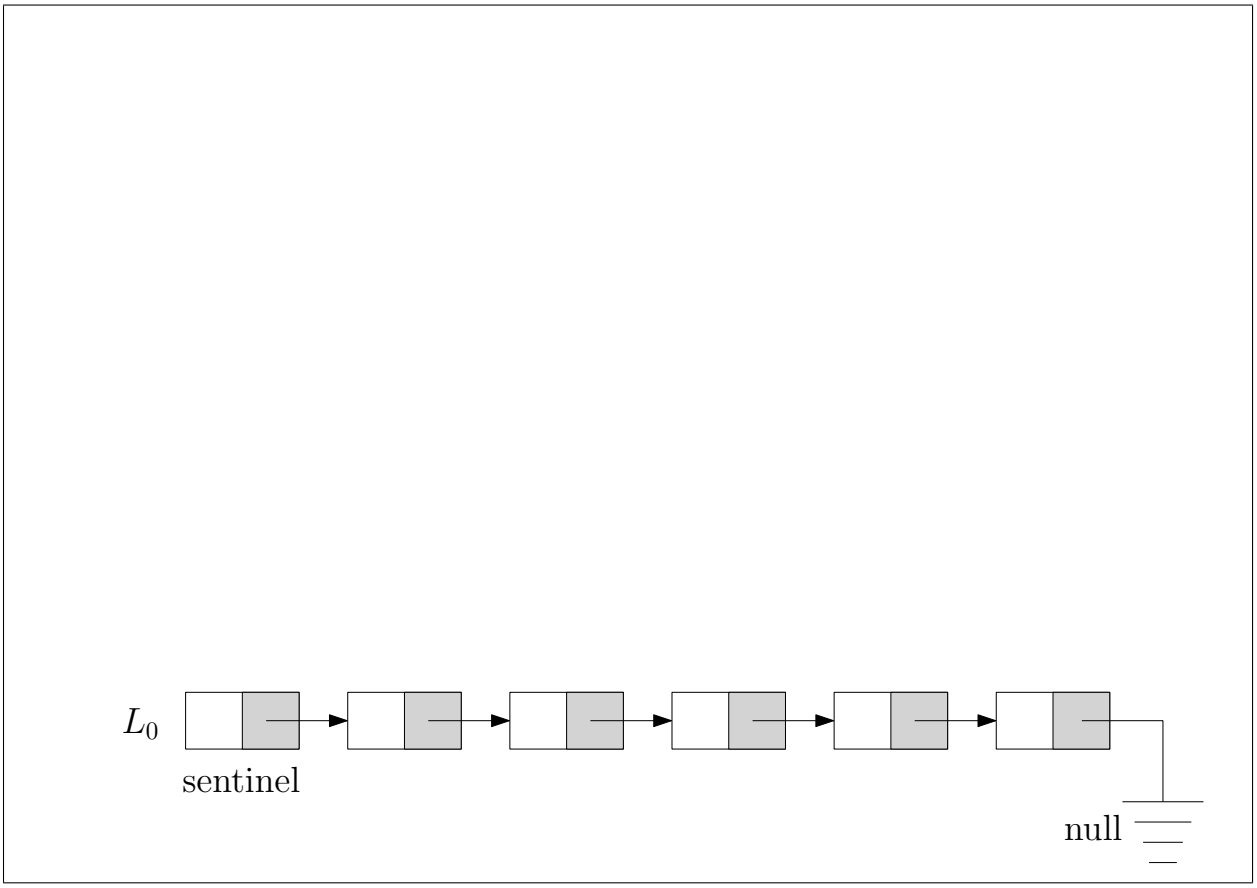
If n = **1000**, what is the location (b, j) of the element that is located at index **746**? Be careful to show all of your work in determining both b and j, as a correct answer presented without valid justification will receive no marks.

4. A **Skiplist** is a sequence of singly-linked lists denoted L_0 to L_H, with H representing the final "height" of the **Skiplist**. This **Skiplist** will store the contents (characters) in alphabetical order. Recall that each element added is included in L_0. To determine if it is included in subsequent lists, we "flip a coin". If the comes up **heads**, we include the element in the next list and repeat the process. If we flip **tails**, the process ends.

   Starting from an empty **Skiplist**, we add the following characters in the order given along with the sequence of coin flips that accompanies them. Draw the resulting **Skiplist**.

   1) **b - heads, heads, tails**
   2) **c - tails**
   3) **f - heads, heads, tails**
   4) **a - heads, heads, heads, tails**
   5) **d - tails**

   (a) Carefully draw the **Skiplist** that would be created by adding to the image below. Include arrows to indicate pointers from one node to another.



(b) You are to add **e** to the **Skiplist** above in a **Node** with height 2. This requires a "find" operation. Describe each step taken during the "find" operation as follows (you may not require every Step listed below):

i. If you move "right" in L_3 from a node containing x to a node containing y, write

$$L_3 \quad x \to y$$

ii. If you move "down" one list on a node containing x from L_3 to L_2, write

$$x \quad L_3 \to L_2$$

Step 1: _____

Step 2: _____

Step 3: _____

Step 4: _____

Step 5: _____

Step 6: _____

Step 7: _____

Step 8: _____

(c) To insert **e**, the Node containing **e** must be added to multiple lists. Write each list this Node was added to, and the letters it was inserted between. For full marks, list them in the order they would occur in the **Skiplist** code from the **ods** library (and seen in class). For example, if the Node containing **e** was added to L_3 between Nodes containing **x** and **z**, you would write:

$$L_3, x, z$$

You may not require every line listed below.

Insert: _____

Insert: _____

Insert: _____

Insert: _____

Insert: _____

Insert: _____

rough work - don't write answers here ... unless you absolutely have to

rough work - don't write answers here ... unless you absolutely have to