CSCI 1933 Lab 12

Midterm II Review

Rules

There is nothing to submit via Moodle for this lab. You have **until the last office hours on Monday** to have this lab checked off by ANY TA if you were not present in today's lab.

New rule: You must work **alone** on this lab. In this lab, you will solve a set of problems in preparation for the second Midterm Exam later this week. You **cannot** use your computer/tablets/smartwatches or the CSE machines to solve these problems. You must solve them on a piece of paper and hand in your solutions by the end of lab.

Evaluations

Mandatory Survey: Please submit feedback regarding the labs' contents, questions, including feedback on structure, TA, and any thing that you think can help us for future labs and TAs. Note: This survey is primarily meant to help Professor and TAs to design future labs for 1933 including how they can be effective in their approach to assure the required readiness and learning that students should have by end of this course.

Responses are anonymous.

Requirements:

1. Keep 10 minutes aside to take a Survey,
2. Evaluating each of your lab TA is mandatory, and
3. Need to show the submitted screen to your TA before leaving the lab.

Use the following link to access the survey: `https://goo.gl/forms/Kxw2M3TUeGK37w8v2`.

Lab Overview

As you all are hopefully already aware at this point, you have your second midterm this week. But if this is news, don't panic! This review lab is here to help you out (and even if you did know, you should still stick around - if for no other reason than that all the steps in this lab are required). The problems are sectioned off as follows:

1. Array List
2. Linked List
3. Queues
4. Stacks
5. Exceptions
6. Complexity Analysis
7. Other Concepts

You **do not** have to complete all of the problems in this lab to get it checked off. Instead, you will get points for this lab if either:

1. you stay the entire lab and hand in your work showing sufficient effort was put into reviewing, or
2. you finish everything before lab starts/ends.

1 Array List

These problems will require you to recall your `ArrayList` implementation in project 3. Remember that you will not have electronics by your side during the midterm, so it is best to remember how you implemented your `ArrayList` in project 3.

1.1 Binary Search, Binary, ary, ry, y I found the y!

Binary search is a pretty neat searching algorithm that runs in `O(lgn)` time. The only problem is that it only works on a sorted collection, here's why. Let's say you are looking for 34 in the following list: $\langle -12, -11, -3, 5, 10, 11, 34, 64, 100, 1933\rangle$. Binary search looks at the element in the middle of the list, in this case 11. Since $34 > 11$, we can rule out the lower half of the list since it is sorted. Binary search then shifts its focus to the upper half: $\langle 34, 64, 100, 1933\rangle$. Again, looking at the middle element, $34 < 64$. Focusing on the lower half $\langle 34\rangle$ and looking at the element in the middle, we see that it is what we are looking for!

Notice how we only had to do 3 comparisons to determine whether or not 34 was in our list. If we were to do a linear search, we'd have to make 7 comparisons!

For this problem, implement a `boolean binarySearch(T element)` method in your `ArrayList` class and explain why binarySearch runs in `O(lgn)` time. The method should return true if the element is in your list and false if not. You may assume that your elements are sorted in increasing order.

1.2 If I may intersect for a moment

The intersection of two collections A and B is defined to be the collection that contains all elements of in A that also belong to B (and vice versa).

For this problem, implement `public void intersect(List<T> other)` method in your `ArrayList` class. Instead of returning the intersected `List`, it should modify the data in your list to be the result of the intersection.

Example: Suppose `mixture` contains: $\langle 1, 2, 3, 4\rangle$ and `other` contains $\langle 2, 4, 6\rangle$. Then `mixture .intersect(other)` should modify `mixture` such that it only contains $\langle 2, 4\rangle$.

2 Linked List

These problems will require you to recall your `LinkedList` implementation in project 3. Remember that you will not have electronics by your side during the midterm, so it is best to remember how `Node`'s work and how you implemented your `LinkedList` in project 3.

2.1 You get removed! You get removed! Every-two gets removed!

For this problem, implement `public void removeEvery(int n)` in your `LinkedList` class. This method should remove every n^{th} element from your list. You may assume that $n \geq 0$; if $n = 0$ or $n > size()$, do nothing.

Example: Suppose we have the following list: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$ and we call `removeEvery(2)`. We should end up with the following list: $A \rightarrow C \rightarrow E$.

2.2 Alright, break it up!

For this problem, implement `public List<List<T>> extractGroupsOf(int n)` in your `LinkedList` class. This method will return a list of lists, each of $size \leq n$. You may assume that $n \geq 0$; if $n = 0$ return an empty `List`.

Example: Suppose we have the following list: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$ and we call `extractGroupsOf(2)`. The following list is returned: $\langle A \rightarrow B,\ C \rightarrow D,\ E\rangle$.

Constraint: You **cannot** use your `get(int index)` method or any method that gets a specific node. You must do this problem by traversing the links between the nodes in your list

3 Queues

These problems will require you to recall your `Queue` implementation from lecture. Remember that you will not have electronics by your side during the midterm, so it is best to remember how `Queues` can be implemented using an `array` and `Nodes`

3.1 A Stacked Queue

You know how you've seen a `Queue` being implemented using an `array` and `Nodes`? Well, now you have to implement it using a `Stack`. Create a `StackedQueue` class and implement the following methods: `enqueue(T element)`, `dequeue()`, `peek()`, `isEmpty()`. You can use any of the `Stack` implementations from the lecture examples or use Java's built in `Stack`.

Constraint: If you are using an `array`-based `Stack`, you **cannot** touch the underlying `array`. Similarly, if you are using a `Node`-based `Stack`, you **cannot** touch the underlying `Nodes`

3.2 A Biased Queue

Budgers are terrible people, especially during Black Friday. Fortunately, we have a biased queue that prevents budgers from leaving the queue before non-budgers. For simplicity sake, our biased queue will be a queue of Strings. Each budger will be represented as the String `"Budger"` and each non-budger as some other String. Create a `BiasedQueue` class and implement the following methods:

- `public void enqueue(String person)` – This will put the person into the queue. If the person is a budger, allow the person budge the person at the end of the queue.

  Example: Suppose this is the state of our queue: `Andy, Leslie, Ron, Tom` and we call `enqueue("Budger")`. Then the queue becomes: `Andy, Leslie, Ron, Budger, Tom`.

- `public void dequeue()` – This will remove the person at the front of the line. If the person at the front of the line is a budger, then you should remove the first non-budger in line. If there are only budgers in line, then remove the first budger.

  Example: Suppose this is the state of our queue: `Budger, Budger, Andy, Leslie, Ron` and we call `dequeue()`. Then the queue becomes: `Budger, Budger, Leslie, Ron`.

Example: Suppose this is the state of our queue: `Budger, Budger` and we call `dequeue()`. Then the queue becomes: `Budger`.

- `public String peek()` – This returns the person at the front of the line.
- `public boolean isEmpty()` – Return `true` if empty and `false` otherwise.

You may use **any** data-structure you've learned so far as the underlying data structure for your `BiasedQueue`.

4 Stacks

These problems will require you to recall your `Stack` implementation from lecture. Remember that you will not have electronics by your side during the midterm, so it is best to remember how `Stacks` can be implemented using an `array` and `Nodes`

4.1 A Queued Stack

You know how you've seen a `Stack` being implemented using an `array` and `Nodes`? Well, now you have to implement it using a `Queue`. Create a `QueuedStack` class and implement the following methods:
`push(T element), pop(), top(), isEmpty()`. You can use any of the `Queue` implementations from the lecture examples or use Java's built in `Queue`.

Constraint: If you are using an `array`-based `Queue`, you **cannot** touch the underlying `array`. Similarly, if you are using a `Node`-based `Queue`, you **cannot** touch the underlying `Nodes`

4.2 A Hack Stack

You know how some times you just want stuff from the middle of a pile but you can't take it because it will cause absolute mayhem, e.g. playing Jenga? Fortunately, we can take something from the middle of a pile with our handy-dandy `HackStack`. Create a `HackStack` class and implement the following methods:

- `public void push(T element)` – This just pushes the element onto the stack.
- `public T pop()` – This just pops the top element from the stack and returns it; if empty, return `null`.
- `public T pop(int n)` – This pops the n^{th} element from the top of the stack. You may assume that $n \geq 0$ where 0 corresponds to the top of the stack. If $n \geq$ the size of the stack, return `null`.
- `public T peek()` – This returns the item at the top of the stack.

Constraint: Your **MUST** use a `Stack` as the underlying data structure. You can use Java's built-in `Stack`, one that you implemented yourself, or any of the ones in the lecture examples.

Constraint: If you are using an `array`-based `Stack`, you **cannot** touch the underlying `array`. Similarly, if you are using a `Node`-based `Stack`, you **cannot** touch the underlying `Nodes`

5 Exceptions

5.1 From Bad to Worse

What is the difference between returning a value to signify bad input versus throwing an exception?

5.2 To Check Or Not to Check?

What is the difference between a checked and unchecked Java exception?

5.3 Exception Types

Name two subclasses of an `Exception` and give a situation in which they would be thrown.

5.4 Are You the Exception?

Given the following Java code, what exceptions could be produced? Explain your reasoning.

```java
public class MatchingStrings {
    public static boolean containsMatch(String[] array, String match) {
            boolean containsMatch = false;
                for (int i = 0; i < 10; i++) {
                    if (array[i].equals(match)) {
                            containsMatch = true;
                        }
                }
                return containsMatch;
        }

        public static void main(String[] args) {
                //...
        }
}
```

5.5 Covering Your Bases (And Your Edges)

Suppose you have implemented your own version of `LinkedList`, as you did in project 3. You have all the required methods in place, your code is not producing any compile time errors, and given inputs that you expected, your code runs beautifully. Great! But you are not finished quite yet; it is time to test your code for edge cases. Identify at least three edge cases you should test for to ensure your code is functioning. (In particular, consider the add(int index, T element) and remove(T element) methods for LinkedList. What edge cases should you be certain to test?)

6 Complexity Analysis

6.1 Sorting Basics

Complete the following table (assume the data to be sorted is stored in an array):

Runtime Complexity	Best-Case	Average-Case	Worst-Case
Merge-Sort			
Bubble-Sort			
Selection-Sort			

Provide explanations for your answers.

6.2 Big-O

What is Big-O for the method `containsMatch(int[] array, int n)` in the following Java code? Explain your answer.

```java
public class MatchingSubsequences {
    //checks int array for subsequences of n matching ints
    public static int containsMatch(int[] array, int n) {
        int countMatches = 0;
        for (int i = 0; i < array.length - (n - 1); i++) {
            boolean containsMatch = true;
            for (int j = i + 1; j < i + n; j++) {
                if (array[i] != array[j]) {
                    containsMatch = false;
                }
            }
            if (containsMatch) {
                countMatches++;
            }
        }
        return countMatches;
    }

    public static void main(String[] args) {
        int[] array = {2, 2, 2, 3, 4, 4, 4, 4, 4, 5, 5};
        int n = 5;
        System.out.println("The array contains " + containsMatch(array, n) +
                           " subsequence(s) of " + n + " matching ints.");
    }
}
```

6.3 More Big-O

What is Big-O for the method `add(T element)` in the following Java code? Explain your answer.

```java
public class ArrayList<T extends Comparable<T>> implements List<T> {
    private T[] list;
    private int size;

    public ArrayList() {
            list = (T[]) new Comparable[2];
            size = 0;
    }

    private void doubleListLength() {
            T[] newList = (T[]) new Comparable[list.length * 2];
            for (int i = 0; i < size; i++) {
                    newList[i] = list[i];
            }
            list = newList;
    }

    public boolean add(T element) {
            if (element == null) {
                    return false;
            }
            if (size == list.length) {
                    doubleListLength();
            }
            list[size] = element;
            size++;
            return true;
        }

        //...

    public static void main(String[] args) {
            //...
    }
}
```

6.4 Sort This Out

Given the input list `[-11, -9, -8, -7, 7, 8, 9, 11, 22]`, which of the three sorting algorithms listed in section 6.1 would you use to sort the list (in increasing order)? What about decreasing order?

6.5 Trade-Offs

When might you want to use an `ArrayList` over a `LinkedList`? What about a `LinkedList` over an `ArrayList`?

6.6 Show Some Improvement

How can you make `add()` in `LinkedList` operate in constant time `O(1)`?

6.7 More Binary Search

Can the binary search algorithm (Section 1.1) be applied on a `LinkedList` in `O(lgn)` time? If so, code it up and show it to a TA. If not, explain why to a TA.

7 Other Concepts

7.1 Definitions

What does a T indicate in Java? When would you want to use it?

7.2 Apples and Oranges, Abstracts and Inheritance

What is the difference between an abstract class and and inheritance? When might you want to use one over the other? What about the difference between an abstract class and an interface?

7.3 Making Comparisons

What is Comparable, what is Comparator, and how are they different? When might you want to use one over the other?

7.4 Equality

What is the difference between "==" and ".equals()"? When would you want to use one over the other?