



University of Maryland College Park

Department of Computer Science

CMSC132 Spring 2026

Exam #2

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 100 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

Grader Use Only

#1	Part #1 (Short Answer)	22	
#2	Part #2 (Short answer)	12	
#3	Part #3 (Code 1)	32	
#4	Part #4 (Code 2)	34	
Total	Total	100	

Part #1 (Short answer – 2 points each)

1. If an algorithm performs the same number of operations regardless of the input size, its time complexity is _____.
2. A(n) _____ expression can be used only with an interface that has exactly one abstract method, called a functional interface.
3. A named class defined inside a method body is called a(n) _____ class.
4. Big-O notation describes the _____ growth rate of an algorithm as the input size becomes very large.
5. _____ allows a method in Java to accept zero or more arguments of the same type.
6. (12 pts @ 2 each) Assume the class **Guitar** extends **Instrument** which extends **Item**. Further, assume that each of the 3 classes has a default constructor that takes no argument. For part a to f, assume the only code that appears in the main method is the given code in the question. Simply circle C if it will compile, CE if it will compile but throw an exception, and NC if it will not even compile.

a.	<pre>ArrayList list = new ArrayList(); list.add(new Guitar()); Instrument i = (Instrument) list.get(0);</pre>	C	CE	NC
b.	<pre>ArrayList<Instrument> list = new ArrayList<>(); list.add(new Guitar()); list.add((Instrument) new Item());</pre>	C	CE	NC
c.	<pre>ArrayList<?> list = new ArrayList<Guitar>(); Object o = list.get(0);</pre>	C	CE	NC
d.	<pre>ArrayList<? extends Instrument> list = new ArrayList<Guitar>(); list.add(new Guitar());</pre>	C	CE	NC
e.	<pre>ArrayList<Instrument> list = new ArrayList<>(); ArrayList<? super Guitar> other = list; other.add(new Guitar());</pre>	C	CE	NC
f.	<pre>ArrayList<Guitar> gl = new ArrayList<>(); gl.add(new Guitar()); ArrayList<? extends Item> il = gl; Item x = il.get(0);</pre>	C	CE	NC

Part #2 (Short answer – 3 points each)

1. Is knowing that an algorithm is in $O(n^2)$ sufficient to conclude that it is not in $O(n)$? Explain.

2. Use the formal definition of Big-O notation as given in class to prove that $5n+100$ is $O(n)$. Assume $N_0=1$. Find a value of $M > 0$ such that the definition of Big-O is satisfied for all $n \geq 1$. Clearly state your value for the constant M and justify why it works.

3. A correctly performed binary search is applied to a sorted array of 100 elements. Explain why it is impossible for the search to require 10 comparisons.

4. Using selection sort, an array has 5 elements. Exactly how many comparisons are performed to sort the array? Show your reasoning by summing the comparisons done in each pass.

Part #3 (Code)

You will finish the **iterator method** for the code below. The code implements an **Iterable class that iterates over only the vowels** in a character array. The iterator should return the vowels **in the order they appear in the array**.

In this design:

- The class stores the characters in a **char[]** array named **letters**.
- The iterator should **skip all non-vowel characters**.
- Vowels include both **uppercase and lowercase** versions of **a, e, i, o, u**.
- The iterator should return the vowels **in the same order they appear in the array**.
- If there are no remaining vowels, **hasNext()** should return **false**.

Directory ID:

You will implement the `iterator()` method using an **anonymous class** that implements `Iterator<Character>`.

In addition to `next()` and `hasNext()` method, the anonymous iterator class you may include:

- Any **private fields** you need.
- An **initialization block**.
- Up to **two private helper methods**.

You may **not add any additional fields or methods to the outer class**.

Assume:

- The array `letters` is **not null**.
- The iterator should **not modify the array**.
- The `next()` method should throw a `NoSuchElementException` if there are no more vowels to return.
- Only allowed library methods are making the `NoSuchElementException` object, the `Iterator<Character>` object to be returned, and using `Character.toLowerCase` method to make the code not be case-sensitive to vowels. Using the `length` field of the array is allowed.

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class VowelIterator implements Iterable<Character> {

    private char[] letters;

    public VowelIterator(char[] letters) {
        this.letters = letters;
    }

    @Override
    public Iterator<Character> iterator() {
        // YOU WILL CODE THIS METHOD
    }

    // simple test driver
    public static void main(String[] args) {
        char[] letters = {'a', 'b', 'E', 'i', 'o', 'u', 'x', 'a', 'y'};
        VowelIterator vowels = new VowelIterator(letters);

        System.out.print("Vowels in the array: ");
        for (char c : vowels) {
            System.out.print(c + " ");
        }
        System.out.println();
    }
}
```

<p><u>OUTPUT</u> Vowels in the array: a E i o u a</p>
--

```
public Iterator<Character> iterator() {
```

Part #4 (Code)

You will complete **two methods** in the **MyLinkedList** class below. The class implements a **generic singly linked list** where elements are added to the **front** of the list.

You will finish the following methods:

1. **findSecondMaxAux** – a **recursive helper method** used to compute the second largest element in the list.
2. **remove** – an **iterative method** that modifies the list.

For full credit:

- You may use the **compareTo** method, but **not any other methods from the Java library**.
- You **may not add additional fields, methods, or constructors**.
- You must use the **provided Node structure**.

Assume:

- The linked list contains **at least two elements**.
- The list contains **no null data values**.
- **All elements in the list are unique**.

```
public class MyLinkedList<T extends Comparable<T>> {
    private class Node {
        private T data;
        private Node next;
        private Node(T data) {
            this.data = data;
        }
    }
    private Node head;
    public MyLinkedList() {
        head = null; }

    /* Assume public MyLinkedList<T> add(T data) as seen in class code example that adds to the front. */

    public T findSecondMax() {
        return findSecondMaxAux(head, null, null); }

    private T findSecondMaxAux(Node current, T max, T secondMax) {
        /* YOU WILL CODE THIS*/ }

    /* Assume public void printList() that will print the data in each node of the list followed by an -> */

    public void remove() {
        /* YOU WILL CODE THIS*/ }

    public static void main(String[] args) {
        MyLinkedList<Integer> list = new MyLinkedList<>();
        list.add(4).add(9).add(2).add(7).add(5);
        list.printList();
        System.out.println("Second Largest: " + list.findSecondMax());
        list.remove();
        list.printList();
    } }
```

<u>OUTPUT</u>
5 -> 7 -> 2 -> 9 -> 4
Second Largest: 7
7 -> 9

4.1 Your task is to **complete the recursive helper method** so that it returns the **second largest element in the linked list**. **No points if your solution is not recursive or it uses a loop.** You may use the `compareTo` method, but not any other methods from the Java library.

The recursive method receives three parameters:

- **current** – the node currently being examined.
- **max** – the **largest value found so far**.
- **secondMax** – the **second largest value found so far**.

The first call is already in the given code: `return findSecondMaxAux(head, null, null);`

```
private T findSecondMaxAux(Node current, T max, T secondMax) {
```

4.2 This method should **modify the linked list so that only the 2nd, 4th, 6th, ... nodes remain in the list**. In other words, the method should **remove the first node, keep the second node, remove the third node, keep the fourth node, and so on**. In your code, you can just detach the unwanted nodes and do not need to modify the next pointer of the removed nodes. Reminder that the linked list contains **at least two elements**. For full credit, the method must be implemented **iteratively** (no recursion) using only ONE loop. No library method calls allowed.

```
public void remove() {
```