Points off	1	2	3	4			Raw Points Off

Your Name: _____

Your UTEID: _____

Circle your TA's Name: **Aditya Ahmad Aman Anna Bersam Brad Brayden Casey Eliza Gracelynn Lauren Namish Nidhi Pavan Sai**

Instructions:

1. There are **4** questions on this test. 100 points available. Scores shall be scaled to 250 points.
2. You have 2 hours to complete the test.
3. Place your final answers on this test. Not on the scratch paper. **Answer in pencil**. Exams not completed in pencil are not eligible for a regrade. You may use highlighters on the exam.
4. **This exam shall be entirely your own work.** You may not use **outside resources of any kind** while taking the test. Please remove any smart watches and put them and any mobile devices away.
5. When answering coding questions, ensure you follow the restrictions of the question.
6. Do not write code to check the preconditions.
7. On coding questions 2 and 3 (but not 4), you may implement your own helper methods.
8. On coding questions make your solutions as efficient as possible given the restrictions of the question.
9. Test proctors will not address any questions regarding the content of the exam. If you think a question is ambiguous or has an error, state your assumptions and answer based on those assumptions.
10. When you complete the test show the proctor your UTID and give them the test. Please place used and used scratch paper in the appropriate boxes at the front of the room. Please leave the room quietly.

1. (2 points each, 50 points total) Short answer. Place your answer on the line next to or under the question. Assume all necessary imports have been made.

   a. If a question contains a syntax error or compile error, answer **compile error**.
   b. If a question would result in a runtime error or exception, answer **runtime error**.
   c. If a question results in an infinite loop, answer **infinite loop**.
   d. Recall when asked for Big O your answer shall be the most restrictive correct Big O function. For example, Selection Sort is average case $O(N^{2})$, but per the formal definition of Big O it is correct to say Selection Sort is $O(N^{3})$, $O(N^{4})$ and so forth.
      Give the most restrictive, correct Big O function. (Closest without going under.)
   e. Assume $\log_2(1{,}000) = 10$ and $\log_2(1{,}000{,}000) = 20$.

A. What is output by the method call **`a(8, 1)`**? _____

```
public static void a(int x, int y) {
    if (x <= 1)
        System.out.print("!");
    else {
        System.out.print(y);
        y *= 2;
        a(x - y, y);
        System.out.print(y);
    }
}
```

B. What is output by the line of code **`System.out.println(b(13));`**? _____

```
public static int b(int x) {
    if (x == 0)
        return 1;
    else
        return 2 + b(x - 2);
}
```

C. What is output by the line of code **`System.out.println(c(-4));`**? _____

```
public static int c(int x) {
    if (x >= 0)
        return 10;
    return x + c(x + 1) + c(x + 2);
}
```

D. The following method takes 1 second to complete when **`x = 30`**. What is the expected time for the method to complete when **`x = 34`**?

_____

```
public static int d(int x) {
    if (x <= 0)
        return 2;
    else
        return 1 + d(x - 1) + d(x - 1);
}
```

E. The following method takes 0.001 seconds to complete when **`x = 7_500`** What is the expected time for the method to complete when **`x = 15_000`**?

_____

```
public static int e(int x) {
    if (x <= 0)
        return 1;
    else
        return x + e(x - 1) + x * 2;
}
```

F. What is output by the following code? Recall the output of the **`toString`** method from the **`Map`** interface: **`{key_1=value_1, key_2=value_2, ..., key_n=value_n}`**

_____

```
TreeMap<Integer, String> m1 = new TreeMap<>();
m1.put(5, "C");
m1.put(1, "M");
m1.put(3, "T");
m1.put(1, m1.get(3));
m1.put(m1.size(), "Z");
System.out.print(m1);
```

G. What is output by the following code? Recall the output of the **`toString`** method from the **`Map`** interface: **`{key_1=value_1, key_2=value_2, ..., key_n=value_n}`**

_____

```
TreeMap<Integer, Integer> m2 = new TreeMap<>();
int[] data = {4, 2, 4, 0, 0, 2, 4};
for (int i = data.length - 1; i >= 0; i--) {
    m2.put(data[i], i); // key, value
}
System.out.print(m2);
```

H. The following method takes 0.05 seconds to complete when **`n = 1_000_000`**. What is the expected time for the method to complete when **`n = 4_000_000`**? Assume the **`Random.nextInt`** method is O(1).

_____

```
public static Map<Integer, Integer> h(int n, Random r) {
   Map<Integer, Integer> result = new HashMap<>(); // A
   for (int i = 0; i < n; i++) {
       result.put(r.nextInt(), i);
   }
   return result;
}
```

I. If the line marked **`// A`** is changed to the following:

```
Map<Integer, Integer> result = new TreeMap<>(); // A
```

The following method takes 0.1 seconds to complete when **`n = 1_000_000`**. What is the expected time for the method to complete when **`n = 4_000_000`**? Single number for an answer. No fractions or functions.

_____

J. Which of the following statements about abstract classes are true? _____

1. Abstract class can be declared as a **`final`** class.
2. Abstract classes cannot have constructors.
3. Abstract classes can have methods declared **`default.`**
4. Abstract classes can have instance variables.

K. What is output by the following code? _____

```
public abstract class Counter {
    public abstract int getCount();

    public String toString() { return (getCount()) * 2 + ""; }
}

public class StringCounter extends Counter {
    private String str;

    public StringCounter(String s) {
        str = s;
    }

    public int getCount() { return str.length(); }
}

// client code
StringCounter sc = new StringCounter("CS314");
System.out.println(sc);
```

L. What is the worst-case order of the following code? N = **`list.size().list`** is a **`java.util.LinkedList`** object.

_____

```
// pre: list != null
public static int remove(LinkedList<Integer> list,
                         int tgt) {
    int count = 0;
    while (list.size() > 0 && list.get(0) < tgt) {
        count++;
        list.remove(0);
    }
    return count;
 }
```

M. What is the worst-case order of the following code? N = **`list1.size() = list2.size().`** Both lists are **`java.util.LinkedList`** objects.

_____

```
// pre: list1 != null, list2 != null, list1.size() == list2.size()
public static int countUp(LinkedList<Integer> list1,
                          LinkedList<Integer> list2) {
    int total = 0;
    for (int i = 0; i < list1.size(); i++) {
        if (list1.get(i) != list2.get(i)) {
            total += list1.get(i) + list2.get(i);
        }
    }
    return total;
}
```

N. Consider the following timing data for a method that sorts arrays of ints. Of the sorts we studied in class which one does the method **most** likely use based on the timing data?

_____

Number of elements in array	Time to sort array of N distinct elements in random order.	Time to sort array of N elements that all equal the same value.
1,000,000	10 seconds	2,000 seconds
2,000,000	21seconds	8,000 seconds
4,000,000	44 seconds	32,000 seconds

O. What is output by the following code? The stack class is the one developed in lecture.

_____

```
Stack314<Integer> st = new Stack314<>();
int[] data1 = {4, 2, 5, 6, 7, 8, 3, 1, 0};
for (int x : data1)
    if (x % 2 == 0)
        st.push(x + 1);
while(!st.isEmpty())
    System.out.print(st.pop() + " ");
```

P. We perform a binary search on an array with 1,000,000 elements 100 times. The value we are looking for is never actual present in the array. How many times in total does the algorithm access an element from the array during the 100 searches?

_____

Q. We perform a linear search on an array with 1,000,000 elements 100 times. The value we are looking for is never actual present in the array. How many times in total does the algorithm access an element from the array during the 100 searches?

_____

R. The following code takes 2 seconds to complete when the queue initially contains **`100_000`** elements. What is the expected time for the method to complete when the queue initially contains **`200_000`** elements?

The **`Queue314`** class uses a **`LinkedList314`** object as its internal storage container. Recall the **`LinkedList314`** class we developed in lecture used singly linked nodes with references to the first and last nodes. The **`Queue314`** class treats the front of the linked list as the **back** (or end) **of the queue** and the back (or end) of the linked list as the **front of the queue**.

_____

```
public static int r(Queue314<String> q) {
    int total = 0;
    while (!q.isEmpty()) {
        total += q.dequeue().length();
    }
    return total;
}
```

S. In the **`java.util.Map`** interface the **`values()`** method, that returns all the values in the map, returns a **`java.util.Collection`** with the values. Would it be reasonable to change the return type of the **`values()`** method to a **`java.util.Set`**? Why or why not in one sentence?

_____

T. What is the worst case order of the following method if **`list`** is a **`java.util.LinkedList`**? **`N = list.size().`**

_____

```
public static void t(List<Integer> list) {
    int tgt = list.get(list.size() / 2);
    Iterator<Integer> it = list.iterator();
    while (it.hasNext()) {
        if (it.next() <= tgt) {
            it.remove();
        }
     }
}
```

U. What is the worst case order of the method **`t`** from above if **`list`** is a **`java.util.ArrayList`**? **`N = list.size().`**

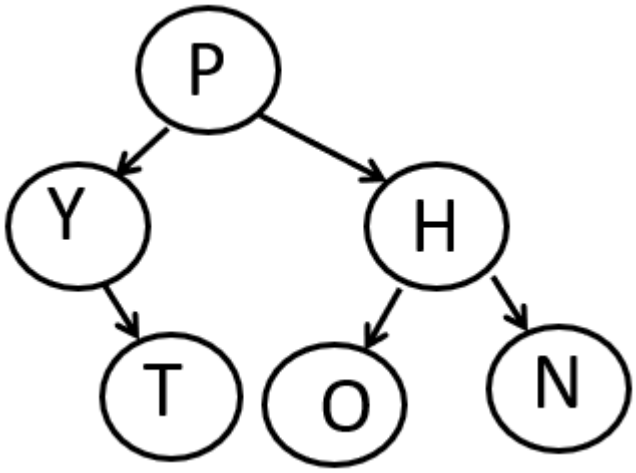
_____

V. In a full binary tree, as defined in class, with 13 nodes what is the minimum number of nodes in the tree with a depth of 3?

_____

W. In a complete tree with 20 nodes, how many nodes have a depth of 3? _____

X. What is the result of an in-order traversal of the binary tree shown to the right?



_____

Y. What is the result of n post-order traversal of the binary tree shown to the right?

_____

2. **Maps and Sets** (18 points) Complete a method that given the results of volleyball games, returns an array with 2 objects: a **`Map`** version of the results as described below and a **`String`** with the name of the team with the highest win percentage.

The parameter to the method is an array of **`String`**s. Elements in the even indices are the names of college volleyball teams. Elements in the odd indices are the results of a game for the preceding team, "W" for a win and "L" for a loss. For example (quotes not shown)

```
[Texas, W, TCU, L, Baylor, W, Texas Tech, L, Texas, W, Baylor, L]
```

The first element in the returned array of **`Object`**s is a **`TreeMap<String, int[]>`**

Each distinct team in the array shall become a key in the map. The values for each team shall be an array of **`int`**s with a length of 2. The first element shall store the number of games the associated team won and the second element shall store the number of games the associated team lost. In other words, the win - loss record for the team.

The second element in the returned array shall be a **`String`**, the name of the team with the highest win percentage. Win percentage is the number of games won divided by the total games played. If there is a tie you can return the name of any of the team tied for the highest win percentage.

Create a single **`TreeMap<String, int[]>`**, arrays of i**`nt`**s of length 2 for each distinct team in the data array, and an array of **`Objects`** of length 2 to return. Do not create unnecessary arrays.

You may us the following methods:

From the **`TreeMap`** class:

```
public Set<K> keySet()
public boolean containsKey(K key)
public V get(K key)
public V put(K key, V val)
public V remove(K key)
```

From the **`Set`** interface:

```
public Iterator<E> iterator
```

You may use all methods from the **`Iterator`** interface either explicitly or implicitly.

Do not use any other Java classes or methods other than those listed above.

Do NOT use recursion in your solution.

The array of `Strings` is not altered by this method.

```
/* pre: results != null. results.length > 0, results.length % 2 == 0
        No elements of results = null. All elements in odd indices shall
        be either "W" or "L".
  post: per the problem description. results not altered by this method.*/
public static Object[] getMap(String[] results) {
```

3. **Linked Lists.** (16 points) Complete the **`longestRun`** instance method for the **`LinkedList314`** class. The method determines the longest run of a given target value.

- **<u>You may not use any other methods in the LinkedList314 class unless you implement them yourself as a part of your solution.</u>**
- The **`LinkedList314`** class uses singly linked nodes.
- The list only has a reference to the first node in the chain of nodes. No size, no last.
- When the list is empty, first stores **`null`**.
- The lists does **not** store **`null`** values. (Hooray!)
- If the list is not empty, the last node in the structure stores **`null`** in its **`next`** variable.
- You may use the nested **`Node`** class and the **`equals`** method on objects.
- **You may not use any other Java classes or native arrays. Do not create ANY new data structures.**
- **Do not use recursion in your answer.**

```
public class LinkedList314<E> {

     // refers to first node in the chain of nodes.
     private Node<E> first;

     // The nested Node class.
     private static class Node<E> {
          private E data;
          private Node<E> next;
     }
}
```

Examples of calls to **`longestRun(E tgt)`**.
In the examples below the elements of the list are **`String`** objects.

`[A, B, A, A, C, A, D].longestRun(A) -> returns 2`

`[A, B, C, A, D].longestRun(A) -> returns 1`

`[BB, B, CC, C, AA, D].longestRun(A) -> returns 0`

`[].longestRun(A) -> returns 0`

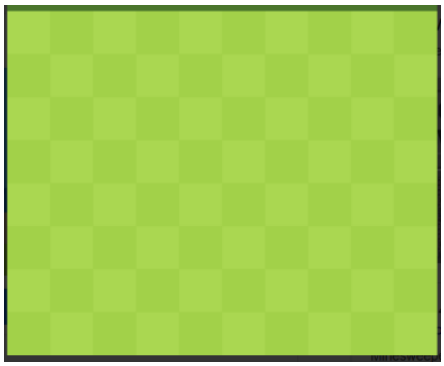
`[A, A, A, A, A].longestRun(A) -> returns 5`

`[B, A, A, A, GG, A, A, A, A].longestRun(A) -> returns 4`

Complete the **`longestRun`** method on the next page.

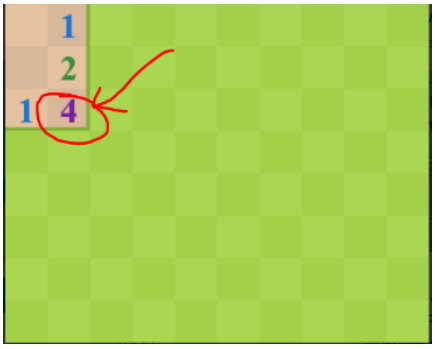
```
/* pre: tgt != null
   post: Return the length of the longest run of the given target
         value in this list. In other words what is the length of the
         longest consecutive streak of the given value in this list?
*/
public int longestRun(E tgt) {
```

4. **Recursive Back Tracking** (16 points) Complete the **`update`** method for a **`Minesweeper`** class as described below. The **`Minesweeper`** class represents the board for the game of the same name. The game takes place on a rectangular grid of cells. Initially all cells are hidden. Some cells contain mines and if the player selects a cell with a mine it explodes and the game is over. If the player picks a cell that does not contain a mine the cell is revealed. If a cell does not contain a mine it displays the number of the (up to 8) bordering cells that **do** have mines. If a cell has 0 mines bordering it, all cells around it are revealed as a shortcut to help the player.



Initial board as seen by the player ---------->

Then if the player selects the cell at row 2, col 1, a 4 is revealed. This means 4 of the 8 cells bordering the cell at row 2, col 1 contain mines.



Then if the player selects the cell at row 0, column 0 it has 0 cells bordering it. It is revealed. (Typically a) is not displayed, just an emoty cell.) **And** the cells at (0, 1), (1, 0), and (1, 1) are revealed. The cell at (0, 1) also has 0 mines bordering it so the 1 is revealed at cell (0, 2). The cell at (2, 1) is already revealed so nothing needs to happen to that cell.

If the user clicks a cell that contains a mine, then the cell is revealed and the game is over.

The **`update`** is called to reveal cells. Initially it will reveal the cell selected by the user. The method updates the 2d array of **`booleans`** in the **`Minesweeper`** class named **`revealed`**. If necessary the game over **`boolean`** is set to **`true`**. The 2d array of **`ints`** in the **`Minesweeper`** class name **`numMines`** is not altered by the **`update`** method.

```
public class Minesweeper {

    private static final int MINE = -1; // used in actual for mines

    // revealed and numMines are rectangular 2d arrays.
    // Which cells are revealed to the user?
    private boolean[][] revealed;

    // How many mines border this cell? -1 if a MINE in the cell.
    // All values in the range [-1, 8]. Already computed!
    private int[][] numMines;

    // Is the game over? Set to true when user clicks a cell with a mine.
    private boolean gameOver;
```

You may NOT add any other helper methods, class variables, or instance variables.

Do not create any new data structures.

Your solution must use recursive backtracking.

```
/* pre: none
   post: per the problem description */
public static void update(int row, int col) {
```