Points off	1	2	3	4	5		Total off	Net Score

Your Name: _____

Your UTEID: _____

Circle your TA's Name: **David K.** **David T.** **Elizabeth** **Henry** **Lilly** **Nina** **Pranav** **Skyler** **Sam** **Trisha**

Instructions:

1. There are **5** questions on this test. 100 points available. Scores will 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**.
4. You may not use a calculator or **outside resources of any kind** while taking the test.
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, you may implement your own helper methods **except question 5.**
8. On coding questions make your solutions as efficient as possible given the restrictions of the question.
9. Test proctors will not answer 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, give them the test and all the scratch paper, used or not, and 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 returned by the method call **`a(-3)`**? _____

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

B. What is output by the method call b("2022CS314")? _____

```
public static void b(String s) {
    if (s.length() <= 2) {
        System.out.print("*");
    } else {
        System.out.print(s.length());
        b(s.substring(2));
        System.out.print("-"); // a hyphen
    }
}
```

C. What is the output by the following client code? _____

```
int[] count = {0};
recC(5, count);
System.out.println(count[0]);

public static int recC(int x, int[] count) {
    count[0]++;
    if (x <= 1) {
        return count[0];
    }
    return 1 + recC(x - 1, count) + recC(x - 2, count);
}
```

D. What is returned by the method call **`d(5, -1)`**? _____

```
public static int d(int x, int y) {
    if (x < y) {
        return x;
    }
    return (x - y) + d(x - 1, y + 1);
}
```

E. What is output by the following client 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<String, Integer> m;
m = new TreeMap<>();
m.put("S", 314);
m.put("B", 37);
m.put("C", 10);
m.put("S", 25);
m.put("A", m.get("S") % 10);
m.remove("B");
m.put("C", m.get("C") / 5);
System.out.println(m);
```

_____

F. The following method is an implementation of the binary search using recursion and the built in Java **`LinkedList`** class. The elements in the list are already in sorted order.

Assuming s is initially 0 and h is initially list.size() - 1 what are the best case and worst case orders for the method?

Best: _____ Worst: _____

```
public static int bSearch(LinkedList<Integer> list, int s, int h, int t) {
    if (s > h)
        return -1;
    int m = (s + h) / 2;
    if (t == list.get(m))
        return m;
    else if (t < list.get(m))
        return bSearch(list, s, m - 1, t);
    else
        return bSearch(list, m + 1, h, t);
}
```

G. The following method takes 0.1 seconds to complete when x = 25.
What is the expected time for the method to complete when x = 35? _____

```
public static int g(int x) {
    if (x <= 0) {
        return 3;
    }
    return g(x - 1) + (x % 10) + g(x - 1);
}
```

H. What is the order of the following method? N = the parameter n _____
Assume the **`Random.nextInt`** method is O(1).

```
public static Map<Integer, Integer> h(int n, Random r) {
    HashMap<Integer, Integer> result = new HashMap<>();
    for (int i = 0; i < n; i++)
        result.put(i, r.nextInt(n * 3));
    return result;
}
```

I. What is the order of method empty shown below? N = **`list.size()`**. The **`LinkedList314`** class is the same as the one implemented in lecture using singly linked nodes and a reference to the first and last nodes in the linked structure of nodes.

_____

```
public static void empty(LinkedList314<String> list) {
    while (list.size() > 0) {
        list.remove(list.size() - 1); // remove based on position
    }
}
```

J. What is the order of method empty shown below? **N = list.size().** For this question the **LinkedList** class uses doubly linked nodes, a header node, and is circular.

_____

```
public static void empty(LinkedList<String> list) {
    while (list.size() > 0) {
        list.remove(list.size() - 1); // remove based on position
    }
}
```

K. What is the order of the following method? Assume the method uses the Java **LinkedList** class. Assume the lists have the same size. **N = list1.size() = list2.size().**

_____

```
public class double k(LinkedList<Double> list1, LinkedList<Double> list2) {
    double t = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i) == list2.get(j) {
                t += list2.get(j);
            }
        }
    }
    return t;
}
```

L. What is the order (Big O) of the **methodL** method shown below? The **sortL** method sorts the array using the insertion sort algorithm as shown in lecture. N = n

_____

```
public static void methodL(int n) {
    int[] ar = new int[n];
    for (int i = 0; i < ar.length; i++) {
        ar[i] = i * 2;
    }
    sortL(ar);
}
```

M. What is the order (Big O) of the **methodM** method shown below? The **sortM** method sorts the array using the quicksort sort algorithm as shown in lecture. N = n

_____

```
public static void methodM(int n) {
    int[] ar = new int[n];
    for (int i = 0; i < ar.length; i++) {
        ar[i] = i * 2;
    }
    sortM(ar);
}
```

N. The following method takes 1 second to complete when **`list.size()`** = 1,000,000 and half the elements in the list are less than **`tgt`**. The elements less that **`tgt`** are in no particular order. The linked list class in this question is the built in Java **`LinkedList`**.

How long will it take the method to complete when **`list.size()`** = 2,000,000? _____

```
public static void removeLess(LinkedList<Integer> list, int tgt) {
    Iterator<Integer> it = list.iterator();
    while (it.hasNext()) {
        if (it.next() < tgt) {
            it.remove();
        }
    }
}
```

O. The following method takes 1 second to complete when **`list.size()`** = 1,000,000 and half the elements in the list are less than **`tgt`**. The elements less that **`tgt`** are in no particular order.

How long will it take the method to complete when **`list.size()`** = 2,000,000? _____

```
public static void removeLess(ArrayList<Integer> list, int tgt) {
    Iterator<Integer> it = list.iterator();
    while (it.hasNext()) {
        if (it.next() < tgt) {
            it.remove();
        }
    }
}
```

P. Assume we have an array of primitive Java **`int`**s with 1,000 distinct elements (no repeats) in random order. Assume x is the number of times we plan to search the array to determine if a give value is present or not. Assume the value we are searching for is always present somewhere in the array.

For what value of x will it be roughly the same amount of work (computations) to: _____

1. Search the array using linear search. Don't bother to sort.
2. Sort the array using a faster sort such as quicksort or mergesort and then search using binary search?

Q. We want to implement a stack data structure. Out internal storage container shall be a linked list. The linked list class we are using consists of singly linked nodes. The instance variables in the linked class are references to the first and last nodes in the linked structure of nodes and an int for the size of the list. When the list is empty first and last are set to null.

Which end of the list (front. back, either end, or not possible) should be treat as the top of the stack so that all four basic stack operations (**`push, pop, top, and isEmpty`**) are O(1)?

_____

R. What is output by the following client code? Assume the **`Queue314`** class is the one implemented in lecture.

_____

```
Queue314<Integer> q = new Queue314<>();
int[] data = {12, 1, 15, 10, 6, 3, 9, 11, 4, 7};
for (int x : data) {
    if (x % 2 != 0)
        q.enqueue(x);
    else if (!q.isEmpty())
        q.dequeue();
}
while (!q.isEmpty())
    System.out.print(q.dequeue() + " ");
```

S. Does the follow class compile as shown? _____

```
public abstract class QuestionS {
    private int x;

    public QuestionS() { x = process(); }

    public abstract int process();
}
```

T. Consider a singly linked node class like the one used in lecture when we developed our singly linked list. This **`Node`** class does not use Java generics. Each node has a reference to an object and another **`Node`**. Consider the following code. The next reference in each **`Node`** is a public variable of type Node. The zero-argument constructor sets the data and next instance variables to null. The constructor that accepts an argument of type **`Node`** sets the new **`Node`**'s next reference to the given **`Node`**. Draw the variables and objects that exist and all references. Use a / for **`null`**.

```
Node n1 = new Node();
n1.next = n1;
Node n2 = new Node(n1);
```

U. Consider the task of implementing an instance method to determine if a singly linked list contains no duplicate elements. The elements are in no particular order. (random order) The list has instance variables for the first node in the structure, the last node in the structure, and the size. The method must be O(1) space, meaning it uses the same amount of memory regardless of the number of items in the list.

What is the worst-case order of the most efficient way of completing the instance method described?

_____

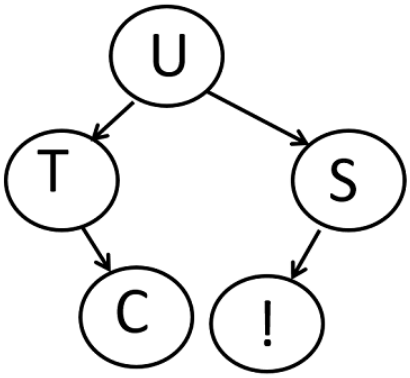
V. What is output by the following code? Recall the output of the toString method from the Map interface: **{key_1=value_1, key_2=value_2, ..., key_n=value_n}**

_____

```
TreeMap<Integer, Integer> map = new TreeMap<>();
int[] data = {1, 3, 1, 5, 3, 5};
for (int i = 0; i < data.length; i++) {
    map.put(data[i], i);
}
System.out.println(map);
```

W. What is the height of the root node in a complete binary tree with 10 nodes? _____

For questions X and Y consider the following binary tree:



X. What is the result of a printing out a post-order traversal of the tree?

_____

Y. What is the result of a printing out an in-order traversal of the tree?

_____

Extra Credit. What is your instructor's favorite sports movie? _____

2. Maps (14 points) - Complete a static method that creates a **back of the book index for** an array of array of Strings representing a book or other document given another array of Strings that represents the key words of the book or document. **It is a precondition of the method that every key word appears at least once in the book.** From **the** Wiki: "In a traditional back-of-the-book index, the headings will include names of people, places, events, and concepts selected by the **indexer** as being relevant and of interest to a possible reader of the book. The indexer may be the author, the editor, or a professional indexer working as a third party. The pointers [not the same as our pointers, but a very similar idea - Mike] are typically page numbers, paragraph numbers or section numbers."

The index produced and returned shall be a **`TreeMap<String, ArrayList<Integer>>`.**

Consider the following simple example. The end of each page is noted with [page x] although the [page x] would not actually be included in the array for a given page. The words on that page would be the elements in the array for that page. Words that are included in the index for this example are underlined for clarity. The spaces would not actually be included. All punctuation exists as separate Strings, meaning the array would contain "net" followed by "." **not** "net."

Volleyball is a team sport that uses a ball and a net . [page 1]

A team typically consist of six players . Members of the team typically specialize . [page 2]

Usually the ball is struck with hands or arms . [page 3]

Volleyball is an Olympic sport and played at many colleges . [page 4]

A team can hit the ball three times . The ball must pass over the net by the third hit . [page 5]

Assume the key words for this example are [Volleyball, ball, team]. Then, the map returned by the method would be:

key	value
Volleyball	[1, 4]
ball	[1, 3, 5]
team	[1, 2, 5]

Note, the key words **are case sensitive**. So, if the String "Ball" appeared on page 2 it would **not** be included in the resulting index.

Complete the method on the next page:

You may use the following methods and classes: (You are not required to use all the methods listed, they are simply the options available to you.)

- Create a TreeMap with the zero-arg constructor and Map methods: V put(K key, V value), boolean containsKey(Object key), V get(Object key), V remove(Object key)
- Create ArrayLists of Integer with th zero-arg constructor and ArrayList methods: add(E val), E get(int pos), boolean contains(Object val), int size()
- String boolean equals(Object other) method
- for-each loops

Do not use recursion.

```
/* pre: book != null,
no elements of book or Strings in elements of book are null.
keyWords != null, no elements of keyWords are null,
ALL elements of keyWords are present in book at least once. */
public static TreeMap<String, ArrayList<Integer>> getIndex(String[][] book,
                                                           String[] keyWords) {
```

3. Linked Lists 1 (10 points) Linked Lists - Complete the following instance method for the **LinkedIntList** class. The **LinkedIntList** class models a list of **int**s and uses a singly linked list of nodes to store the elements of the list.

```
/* pre: other != null. post: per the problem description.
Neither this or other are altered. */
public int compareTo(LinkedIntList other) {
```

The method returns an int < 0 if the sum of the elements in the calling **LinkedIntList** (this) are less than the sum of the elements in the parameter other, 0 if the sums are the same, or an int > 0 if the sum of the elements in the calling **LinkedIntList** (this) are greater than the sum of the elements in the parameter other.

Examples of calls to the **compareTo** method:

```
[].compareTo([6, 5, 1]) -> returns an int < 0
[].compareTo([-6, -5, 8]) -> returns an int > 0
[6, 5, 10, 20].compareTo([6, 5, 1]) -> returns an int > 0
[].compareTo([]) -> returns 0
[6, 1, 3, 1, 1, 1, 1].compareTo([100]) -> returns an int < 0
```

The **LinkedIntList** class for this question.

```
public class LinkedIntList {

    private IntNode first; // first == null iff list is empty

    /* If the list isn't empty the last node in the linked
       structure of nodes has its next reference set to null. */

    // No other instance variables.

    private static class IntNode {
        private int val;
        private IntNode next;
   }
}
```

You may not use any methods from the `LinkedIntList` class unless you implement them as a part of your solution.

Do not add any instance or class variables to the `LinkedIntList` class.

Do not use any other Java classes or methods except the nested `IntNode` class.

Do not add any methods or instance variables to the `IntNode` class.

Do not use recursion.

```
/* pre: other != null. post: per the problem description.
Neither this or other are altered. */
public int compareTo(LinkedIntList other) {
```

4. Linked Lists 2 (13 points) - Complete the following instance method for the **`LinkedList314`** class. The method removes the last element **IF** it equals a given target value. The **`LinkedList314`** class models a list and uses singly linked nodes. No elements (data) of the list are null.

```
/* pre: tgt != null
   post: remove the last element of this list IF it is
         equal to tgt. Return true if this list was altered as a
         result of this method call, false otherwise. */
public boolean removeLast(E tgt) {
```

Examples of calls to **`removeLast`.** Assume the values shown in these examples are **`String`** objects.

```
[A, B, C, A, A].removeLast(A) -> returns true, list becomes [A, B, C, A]

[A, B, C, A, B].removeLast(A) -> returns false, list unaltered
[G, B, C, G, B].removeLast(A) -> returns false, list unaltered
[].removeLast(B) -> returns false, list unaltered
[G, B, C, B, G].removeLast(G) -> returns true, list becomes [G, B, C, B]
[B].removeLast(B) -> returns true, list becomes []
[B].removeLast(A) -> returns false, list unaltered
```

The **`LinkedList314`** class for this question.

```
public class LinkedList314<E> {

    private Node<E> first; // first == null iff list is empty

    /* If the list isn't empty the last node in the linked
       structure of nodes has its next reference set to null. */
    // No other instance variables.

    private static class Node {
        private E val;
        private Node<E>  next;
   }
}
```

You may not use any methods from the `LinkedList314` class unless you implement them as a part of your solution.

Do not add any instance or class variables to the `LinkedList314` class.

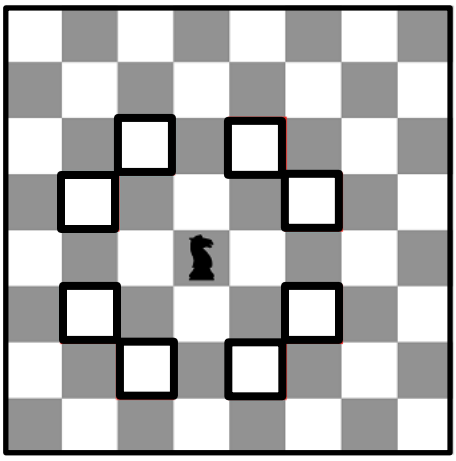
Do not use any other Java classes or methods except the nested `Node` class and the `equals` method.

Do not add any methods or instance variables to the `Node` class.

Do not use recursion.

```
/* pre: tgt != null
   post: remove the last element of this list IF it is
         equal to tgt. Return true if this list was altered as a
         result of this method call, false otherwise. */
public boolean removeLast(E tgt) {
```

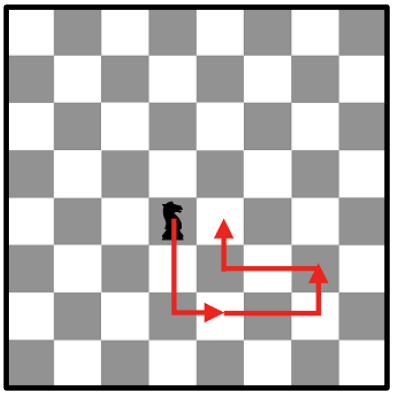
5. Recursion (13 points) -In chess the knight pieces move in an interesting way. Knights move in an "L" shape, meaning they move 2 spaces on one axis and 1 space on the other axis from their current position. This gives them a maximum of 8 possible moves.



The picture to the right shows the spaces the knight shown can move to (darker outline) from its current position. The given **`KNIGHT_DIRECTIONS`** constant below simplifies how to change the position of knight.

Write a method that determines if a knight can move to a given space from its starting space within some number of moves or less.

For example, it takes at least 3 moves for a knight to move to the space to the right of its current space as shown below and to the right.



You may use the following **`Position`** class.

```
public class Position {
    public int row;
    public int col;

   // Creates a Position with the given row and column.
    public Position(int r, int c)
}
```

A knight cannot move off the board and the knight does not "wrap around", meaning it does not move off one edges and appear on the other side.

All positions use 0 based indexing.

You may also use the following class constant in the same class as the method you are completing.

```
private static final int[][] KNIGHT_DIRECTIONS =   {{2, 1}, {2, -1},
        {-2, 1}, {-2, -1}, {1, 2}, {1, -2}, {-1, 2}, {-1, -2}};
```

- **You may not use any Java classes or methods besides the given `Position` class.**
- **You may not add methods or variables to the `Position` class.**
- **You may create and use new `Position` objects.**
- **You must use recursion in your answer.**
- **For this question only, do not create any other methods. All your code must be in the `knightCanReach` method.**

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

```
/* pre: rows > 0, knight != null, target != null,
 * rows is the number of rows on the board, the board is
 * square so the number of columns == the number of rows.
 * knight is the current position of the knight piece.
 * post: returns true iff the knight can move to the target
 * Position in numMoves or less. */
public static boolean knightCanReach(int rows, Position knight,
        Position target, int numMoves) {
```