Points off	1	2	3	4	5	6	Total off	Raw Score

CS 314 – Exam 2 – Spring 2016

Your Name_____

Your UTEID _____

Instructions:

1. There are **6** questions on this test. 100 points available. Scores will be scaled to 200 points.
2. You have 2 hours to complete the test.
3. Place you final answers on this test. Not on scratch paper. Answer in pencil.
4. You may not use a calculator or any other electronic devices 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.
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. (1 point each, 20 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 other 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 should be the most restrictive correct Big O function. For example Selection Sort has an average case Big O of $O(N^2)$, but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of $O(N^3)$ or $O(N^4)$. I want the most restrictive, correct Big O function. (Closest without going under.)

A. What is returned by the method call `methodA(21)`?

```
public int methodA(int x) {
     if (x <= 1) {
          return 3;
     } else if (x % 2 == 0) {
          return 2 + methodA(x - 3);
     } else {
          return 1 + methodA(x / 2);
     }
}
```

_____

B. What is output by the method call `methodB(4)`? _____

```
public void methodB(int x) {
     if (x >= 15) {
          System.out.print("!");
     }
     else {
          System.out.print(x);
          methodB(x + x/2);
          System.out.print(x);
     }
}
```

C. What is returned by the method call `methodC("joker")`? _____

```
public int methodC(String str) {
     int result = str.length();
     if (str.length() > 2) {
          result += methodC(str.substring(0));
          result += methodC(str.substring(2));
     }
     return result;
}
```

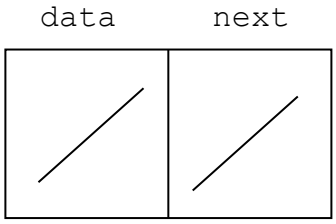
D. What is returned by the method call `methodD(5)`? _____

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

E. What is the worst case order (Big O) of `methodE`?
The method uses the Java LinkedList class.
Assume `list1.size() = list2.size()` = N _____

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

F. Draw the variables, references, and objects that exist after the following code executes. Draw node objects as shown below and boxes for variables. (The example has both instance variables set to `null`. The example does not show any of the variables that actually refer to the Node object. You must show all variables and their references in your drawing.) Use arrows to show references and a forward slash to indicate variables that store `null`. Assume the Node class is the one from our singly linked list examples in lecture.




```
Node<Object> n1 = new Node<Object>(null, null); // data, next
Node<Object> n2 = new Node<Object>("ST", null);
n1.setData(n2.getData());
n1.setNext(n2);
n2.setNext(n1);
Node<Object> n3 = new Node<Object>(null, n2.getNext().getNext());
```

G. What is the worst case order (Big O) of `methodG`? N = `list.size()` _____

```
public ArrayList<String> methodG(ArrayList<String> list, int t) {
     // create result with initial capacity equal to size of list
     ArrayList<String> result = new ArrayList<String>(list.size());

     Iterator<String> it = list.iterator();
     while (it.hasNext()) {
          String str = it.next();
          if (str.length() >= t) {
               result.add(0, str);
               it.remove();
          }
     }
     return result;
}
```

H. What is the result of the following postfix expression? (single integer for answer) _____

```
10 40 8 / 3 5 - * *
```

I. A method uses the merge sort algorithm. Given an array of 1,000,000 ints in random order it takes the method 5 seconds to complete. What is the expected runtime when given an array of 4,000,000 elements in random order?

_____

J. Consider the following timing data for a method that sorts arrays of ints.

Number of elements	Time to sort array with elements in random order.	Time to sort array with elements already in ascending order.
50,000	.01 seconds	5 seconds
100,000	.021 seconds	20 seconds
200,000	.044 seconds	80 seconds

Which sorting algorithm we studied does the method most likely use?

_____

K. A method performs a linear search on an array of ints. It takes 10 seconds for the method to complete 10,000 searches on an array with 100,000 elements. How long do you expect the method take to complete 50,000 searches on an array with 300,000 elements?

_____

L. What is output by the following code? The code uses the Java `Stack` class.

```
Stack<Integer> st = new Stack<Integer>();
int sub = 0;
for(int i = 20; i > 0; i -= sub) {
     st.push(i);
     sub++;
}

for(int i = 0; i < 4; i++) {
     System.out.print(st.pop() + " ");
}
```

M. Java does not allow the instantiation of abstract classes. Consider the following partial class:

```
public abstract class AbstractClass<E> {
     public AbstractClass(int x) // rest of constructor not shown

     // rest of class not shown
}
```

The following code results in a compile error:

```
AbstractClass ac = new AbstractClass(10);
```

Why would an abstract class have constructors if they cannot be instantiated? Be brief.

_____

N. Consider the following method:

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

When x = 20 it takes method mystery 1 second to complete. What is the expected time in seconds for the method to run when x = 30?

_____

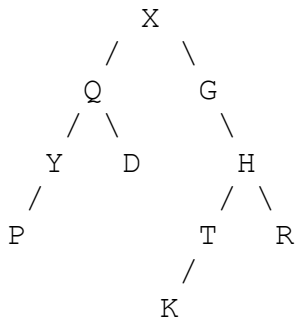
O. The binary search tree class in the code below uses the simple, naïve add algorithm demonstrated in class. It takes 20 seconds for the method to complete when the array has 1,000,000 distinct elements in random order. What is the expected time for the method to complete when the array has 2,000,000 distinct elements in random order?

_____

```
public BST<Integer> makeTree(int[] data) {
     BST<Integer> result = new BST<Integer>();
     for(int i = 0; i < data.length; i++) {
          result.add(data[i]);
     }
     return result;
}
```

P. When the array named `data` in question O contains 10,000 distinct elements in sorted descending order it takes the makeTree method 10 seconds to complete. What is the expected time for the method to complete when the array has 40,000 distinct elements in descending order?

Consider the following binary tree. `X` is the root of the tree.



Q. What is the result of a level-order traversal of the binary tree shown above?

_____

R. What is the result of a pre-order traversal of the binary tree shown above?

_____

S. What is the result of a post-order traversal of the binary tree shown above?

_____

T. The following values are inserted one at a time into a binary search tree using the simple, naïve algorithm demonstrated in class. Draw the result tree.

```
-5   10   0    10   5    -5   15
```

2. Maps (16 points) - Write a method that given a map representing people and the countries they have visited, returns a map of countries with the people who have visited those countries.

The given map stores the names of people (as Strings). The associated value is a list of countries that person has visited. A country appears once for each time the person has visited that country. So for example if Isabelle has visited France twice, "France" will appear twice in the list associated with "Isabelle". Example map with the keys followed by the associated values:

```
KEYS    VALUES
David, [Mexico, Oman, Japan, Mexico, Canada, France, Mexico]
Kelly, [Mexico, Canada, Mexico, Canada, France]
James, [USA, France, Japan]
Isabelle, [Mexico]
Olivia, [Canada, Mexico, Canada, France, Mexico, Mexico]
```

Given the map above, the method you are writing would return the following map. Note, the countries are in "asciibetical" order. Also, even if a person has visited a county multiple times, they only appear once in the list of people who have visited the country. The names of the people who have visited the country can appear in any order in the associated list.

```
KEYS    VALUES
Canada, [David, Kelly, Olivia]
France, [David, Kelly, James, Olivia]
Japan, [David, James]
Mexico, [David, Kelly, Isabelle, Olivia]
Oman, [David]
USA, [James]
```

You may use the default Map constructors and the following methods from the Map class:

- `Set<K>` **`keySet`**`()` - Returns a Set view of the keys contained in this map.
- `V` **`get`**`(Object key)` - Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
- `V` **`put`**`(K key, V value)` - Associates the specified value with the specified key in this map

You may obtain iterators for `Collection`s and use the methods from the `Iterator` interface.

You may use the default constructor, `add()`, `size()`, and `contains(E obj)` method from the Java `ArrayList` class.

You may not use any other methods or classes.

Complete the following method:

```
// pre: countriesVisited != null,
//                        no values in countriesVisited == null
// post: per the problem description
public Map<String, ArrayList<String>> getVisitorMap(
                    Map<String, ArrayList<String>> countriesVisited) {
```

3. Linked Lists (16 points) - Complete the `frequency` instance method for the `LinkedList314` class. The method is an accessor. The method returns the number of elements in the linked list equal to a target value which may be `null`.

- You may not use any other methods in the `LinkedList314` class unless you implement them yourself as a part of your solution.
- The `LinkedList314` class uses singly linked nodes.
- The list has references to the first node in the linked structure of nodes.
- When the list is empty, `first` is set to `null`.
- **Some of the data in the list may equal `null`.**
- If the list is not empty the last node in the list has its next reference set to `null`.
- You may use the nested `Node` class and the Object `equals` method.
- **You may not use any other Java classes or native arrays.**

```
public class LinkedList314<E> {

     // refers to first node in the chain of nodes.
     private Node<E> first;
     // No other instance variables


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

}
```

Examples of calls to `frequency(E target)`. In this example the list contains Strings, but in the example below `null` indicates a null reference, **not** the String "`null`".

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

[A].frequency(A) ->  returns 1

[A, B].frequency(B)  ->  returns 1

[A, B, C, B].frequency(B)  ->  returns 2

[A, B, C, B].frequency(X)  ->  returns 0

[A, B, C, B].frequency(null)  ->  returns 0

[A, null, C, null].frequency(null)  ->  returns 2

[A, null, C, null].frequency(X)  ->  returns 0
```

Complete the following instance method of the `LinkedList314` class.

```
/* pre: none
   post: returns the number of elements in this linked list equal to
tgt value */
public int frequency(E tgt) {
```

4. Queues (16 points) - Write a method that interleaves two queues of integers into a new, single queue.

Consider the following example:

```
first queue:     front [12, 7, 9, 17] back

second queue:    front [5, 8] back

resulting queue: front [5, 12, 7, 8, 9, 17] back
```

The new queue contains interleaved pairs of values from the same positions in the two original queues.

The smaller of the two integers appears first in the interleaved pair in the resulting queue. For example:

12 and 5 are the first values in the two original queues and the values appear as 5, 12 in the resulting queue because 5 < 12.

7 and 8 are the second values in the two original queues and the values appears as 7, 8 in the resulting queue because 7 < 8.

If one queue is larger than the other the remaining values are added to the end of the resulting queue, after the interleaved pairs of values.

You may create and use one queue for your result.

The Queue class has the following methods: `void enqueue(E val), E dequeue(), E front(), boolean isEmpty()`

Complete the method on the next page.

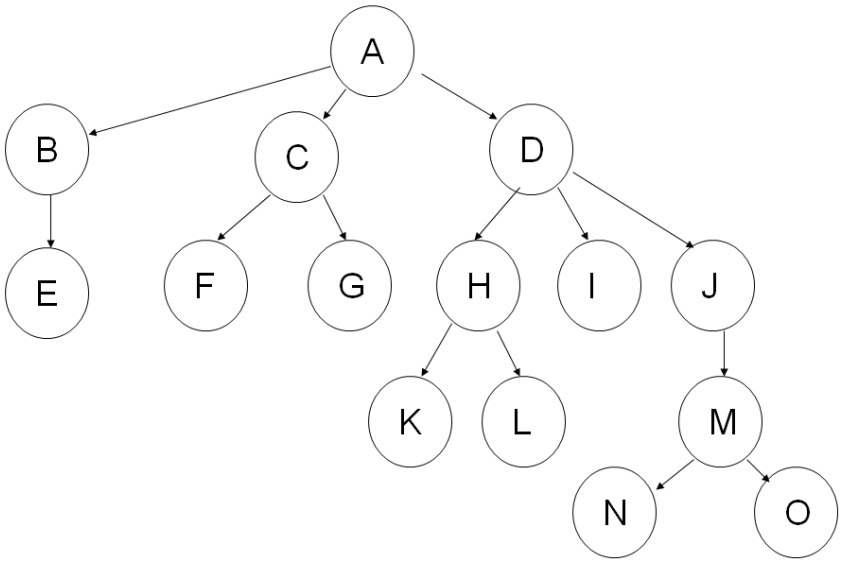
```
// pre: q1 != null, q2 != null, no elements of q1 or q2 == null
// post: return an interleaved queues as described in the question,
// q1.isEmpty() == true, q2.isEmpty() == true
public Queue<Integer> interleave(Queue<Integer> q1, Queue<Integer> q2){
```

5. Trees (16 points) - Implement a helper method for an instance method for a tree class that determines and returns the number of nodes in the tree with a given number of children.

This tree class is a general tree, not a binary tree. The number of child nodes is not restricted.

```
public class Tree<E> {

     private TNode<E> root; // root == null if tree is empty

     // nested TNode class
     private static class TNode<E> {

          // the data this node stores
          private E data;

          // References to this nodes child nodes.
          // If this is a leaf node, children.size() == 0
          private ArrayList<TNode<E>> children;
     }
}
```

Consider the following tree, `t`.




```
t.nodesWithNumChildren(0) returns 8

t.nodesWithNumChildren(1) returns 2

t.nodesWithNumChildren(2) returns 3

t.nodesWithNumChildren(3) returns 2

t.nodesWithNumChildren(4) returns 0
```

Do not create any new nodes or other data structures.

You may use the methods of the `ArrayList` class.

```
// instance method of Tree class, pre: numChildrenTarget >= 0
public int nodesWithNumChildren(int numChildrenTarget) {
     return helper(root, numChildrenTarget);
}
```

Complete the following private helper method in the `Tree` class.

```
private int helper(TNode<E> n, int numChildrenTarget) {
```

6. Recursive Backtracking (16 points) - Weird Numbers (A tiny topic in number theory. I am not making this up.)

Write a recursive method to determine if a number is semiperfect.

A ***natural number*** is an integer greater than or equal to 0.

The ***proper divisors or factors*** of a natural number N are the natural numbers that divide evenly into N using integer division, including 1 but not N.

An ***abundant number*** is one that is less than the sum of its proper divisors. For example:
8 is not abundant. The proper divisors of 8 are 1, 2, 4. 1 + 2 + 4 = 7 7 <= 8.
12 is abundant. The proper divisors of 12 are 1, 2, 3, 4, 6. 1 + 2 + 3 + 4 + 6 = 15 15 > 12

A ***semiperfect number*** is a natural number that is equal to some subset of its proper divisors.

For example:
6 is semi perfect. 1, 2, and 3 are the proper divisors of 6 and 1 + 2 + 3 = 6. (The subset can be the entire set of proper divisors. In other words a set is a subset of itself.)

12 is semiperfect. 2, 4, and 6 are a subset of the proper divisors of 12 and 2 + 4 + 6 = 12.

70 is not semiperfect. The proper divisors of 70 are 1, 2, 5, 7, 10, 14, and 35.
But no subset of those numbers sums to 70.
True, 35 + 35 = 70, but we can only use the 35 once in our sum.

"A ***weird number*** is a number that is abundant, but NOT semiperfect. In other words, the sum of the proper divisors (divisors including 1 bit not itself) of the number is greater than the number, but no subset of those divisors sums to the number itself."

70 is the smallest weird number.
It is an open question in number theory if there are any odd weird numbers.

Complete a recursive method that, given a value and the proper divisors of that value, returns true if a number is ***semiperfect***, false otherwise. **As part of your solution implement a recursive backtracking helper method that does the majority of the work.**

Complete the method and the helper method on the next page.

Note, you are only writing the method to determine if a number is semiperfect.

You are not implementing the method to determine if a number is weird.

```
// pre: properDivisors contains the proper divisors of value,
// value >= 1.
// post: return true if value is semiperfect, false otherwise
public boolean isSemiperfect(int[] properDivisors, int value) {
```