Points off	1	2	3	4	5	6	Total off	

CS 314 – Final Exam – Spring 2019

Your Name_____Your UTEID _____

Instructions:

1. There are **6** questions on this test. 100 points available. Scores will be scaled to 300 points.
2. You have 3 hours to complete the test.
3. Place your final answers on this test, not on scratch paper. Answer in pencil.
4. No outside assistance of any kind is allowed on the exam.
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. Short answer - 1 point each, 20 points total. 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 shall 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.)
e. Assume $\log_2(1{,}000) = 10$ and $\log_2(1{,}000{,}000) = 20$.

A. Using the techniques developed in class, what is the T(N) of the following code? `N = data.length`. _____

```
public static int a(int[] data) {
     int t = 0;
     for (int i = 0; i < data.length; i++)
          for (int j = 0; j < data.length; j++)
               for (int k = 0; k < data.length; k++)
                    t += data[i] * data[j] * data[k];
     return t;
}
```

B. What is the output when we make the method call `b(9, 2);` ?

_____

```
public static int b(int x , int y) {
     if (x <= 1) {
          System.out.print(x * y + " ");
          return x * y;
     }
     else {
          System.out.print(x + " ");
          int r = b(x - y, y * 2);
          System.out.print(y + " ");
          return r;
     }
}
```

C. What is returned by the method call `c(2)`? _____

```
private static int c(int v) {
     if (v >= 6)
          return 1;
     else
          return c(v + 2) + 1 + c(v + 1);
}
```

D. The following method takes 5 seconds to complete when the size of both lists equal 10,000. What is the expected time for the method to complete when the size of both lists equal 20,000?

_____

```
public static int d(LinkedList<Integer> list1,
                                    LinkedList<Integer> list2) {
     int r = 0;
     for (int i = 0; i < list1.size(); i++)
          for (int j = i + 1; j < list2.size(); j++)
               if (list1.get(i) == list2.get(j))
                    r += list1.get(i) + list2.get(j);
     return r;
}
```

E. The following values are inserted one at a time in the order shown to an initially empty binary search tree using the simple algorithm presented in class. What is the result of a pre-order traversal of the resulting tree?

_____

5 3 9 6 0 4 2 9

F. What is output by the following code. Recall the format of the `Map`'s `toString` method: `{k1=v1, k2=v2, ..., kn=vn}`

_____

```
int[] data = {4, 1, 4, 2, 3, 4, 5};
TreeMap<Integer, Integer> m = new TreeMap<>();
for (int i : data) {
    m.put(data[i], i);
}
System.out.print(m);
```

G. What is output by the following code? It uses the `Stack` class developed in class.

_____

```
String sg = "DYNAMIC_PROG";
Stack<Character> st = new Stack<>();
for (int i = 0; i < sg.length(); i = i + 2) {
     st.push(sg.charAt(i));
}
while (!st.isEmpty()) {
     System.out.print(st.pop() + " ");
}
```

H. What is output by the following code? It uses the Java `PriorityQueue` class.

_____

```
int[] data2 = {12, 5, 8, 5, 8, 1, 9};
PriorityQueue<Integer> pq = new PriorityQueue<>();
for (int x : data2) {
     pq.add(x); // enqueue
}
for (int i = 0; i < pq.size(); i++) {
     System.out.print(pq.remove() + " "); // dequeue
}
```

I. What is the maximum possible height of a Red-Black Tree that contains 8 nodes?

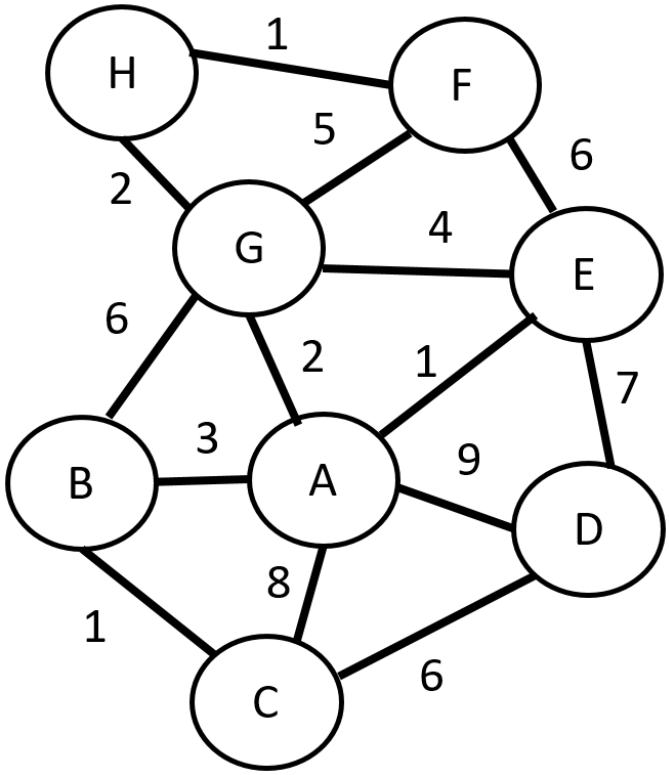
_____

J. The following values are inserted one at a time into a Red-Black tree using the algorithm demonstrated in class. Draw the resulting tree.
Label Red nodes with an R and Black nodes with a B.

7 5 8 3

K. When would you make the internal storage container for a Graph class an adjacency matrix instead of the adjacency lists used in assignment 11?

L. Given the graph to the right, what is the cost of the lowest cost path from vertex C to Vertex F?



M. What is the sum of the edge weights for a minimum spanning tree for the graph to the right?

N. The following method takes 3 seconds to complete when m = 100. What is the expected time for the method to complete when m = 200? The BST uses the simple algorithm presented in class.

```
public static BST<Integer> n(int m){
     BST<Integer> result = new BST<>();
     for (int i = m * m; i > 0; i--)
          result.add(i);
     return result;
}
```

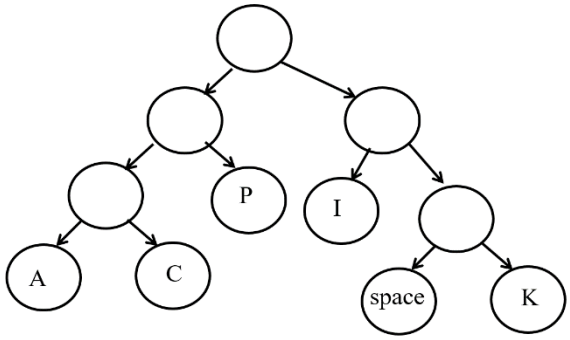
O. The following values are added one at a time to an initially empty min heap using the algorithm presented in class. Draw the resulting min heap.

10 7 12 12 5 2

P. Given the Huffman code tree to the right what does the following decode to?

0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0

_____



Q. The following method takes 1 second to complete when n = 1,000,000. What is the expected time for the the method to complete when N = 2,000,000?

_____

```
public static HashMap<Integer, Integer> q(int n) {
     HashMap<Integer, Integer> hm = new HashMap<>();
     Random r = new Random(); // O(1)
     for (int i = 0; i < n; i++) {
          int k = r.nextInt(); // O(1)
          hm.put(k, i);
     }
     return hm;
}
```

R. Recall the problem of finding the minimum coins needed to make a certain value of change. We demonstrated a recursive backtracking solution and a dynamic programming solution in class. Why was the recursive backtracking solution so much slower than the dynamic programming solution?

_____

S. The following words are added to a Trie as shown in class. Draw the resulting Trie. Use a check mark to indicate nodes that are the terminator for a valid word.

SAT, SAVE, SATS, SIT

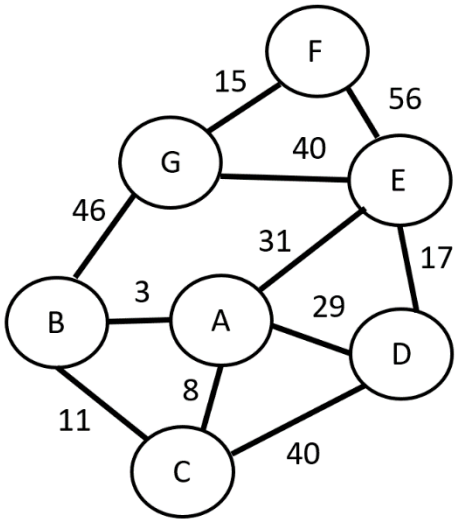
T. What is output by the following code? _____

```
int x1 = (int) IntStream.rangeClosed(3, 20)
          .filter(x -> x % 4 == 0)
          .map(y -> y / 2)
          .filter(z -> z <= 6)
          .count();
System.out.print(x1);
```

2. Graphs and Recursion- 16 points. Complete an instance method for the **`Graph`** class that uses **recursive backtracking** to find the ***widest path*** from one vertex to another.

The widest path from one vertex to another is defined to be the path (set of edges that connects one vertex to another without cycles) that **maximizes** the **minimum** edge weight among the edges in the path.

Consider the graph to the right. There are many paths without cycles from vertex C to vertex B such as: (note this is not a complete list of the paths from C to B)



- The path with one edge with a weight of 11.
  In this path the minimum edge weight is 11. (C to B edge)
- The path from C to A to B.
  In this path the minimum edge weight is 3. (A to B edge)
- The path from C to A to E to G to B
  In this path the minimum weight is 8. (C to A edge)
- The path from C to A to D to E to G to B
  In this path the minimum weight is still 8. (C to A edge)
- The path from C to D to A to E to F to G to B.
  In this path the minimum weight is 15. (F to G edge)
- The path from C to D to E to G to B.
  In this path the minimum weight is 17. (D to E edge)
- The path from C to D to A to E to G to B.
  In this path the minimum weight is 29. (D to A edge)

For the given graph, the last path with a minimum edge weight of 29, is the **widest path** to get from vertex C to vertex B. It is in the path that maximizes the smallest edge weight in the path.

Note, when finding the widest path, we are not concerned with the number of edges in the path or the total cost of the path. The path may be very circuitous as we are simply try to find the path where the smallest edge weight in the path is as large as possible.

Aside: Some routing algorithms in graphs use the widest path to try to minimize bottlenecks in the network represented by the graph.

```
public class Graph {
    // The vertices in the graph.
    private Map<String, Vertex> vertices;
    private static final double INFINITY = Double.MAX_VALUE;

     // for all vertices, set scratch to 0 and prev to null.
    private void clearAll()

    private static class Vertex {
        private String name;
        private List<Edge> adjacent;
        private int scratch;
        private Vertex prev; }
```

```
    private static class Edge {
        private Vertex dest;
        private double cost;
        // equals NOT overridden
    }
}
```

You may use the **get** and **size** methods from the **List** interface.

You may use the for-each loop.

You may use the **equals** method for **String**s.

Do not alter the Graph or its elements in any way other than changing the **scratch** instance variable of **Vertex** objects.

All costs in the **Graph** are **> 0**.

The Graph is undirected. If an edge exists from vertex A to vertex B with a cost of X, there will be an edge from vertex B to vertex A with a cost of X as well.

The method returns the minimum edge weight in the widest path, as defined above, from the start vertex to the destination vertex or **INFINITY** if there is no path from the start vertex to the destination vertex.

Do not create ANY additional data structures.

You must implement a recursive backtracking solution in the helper method.

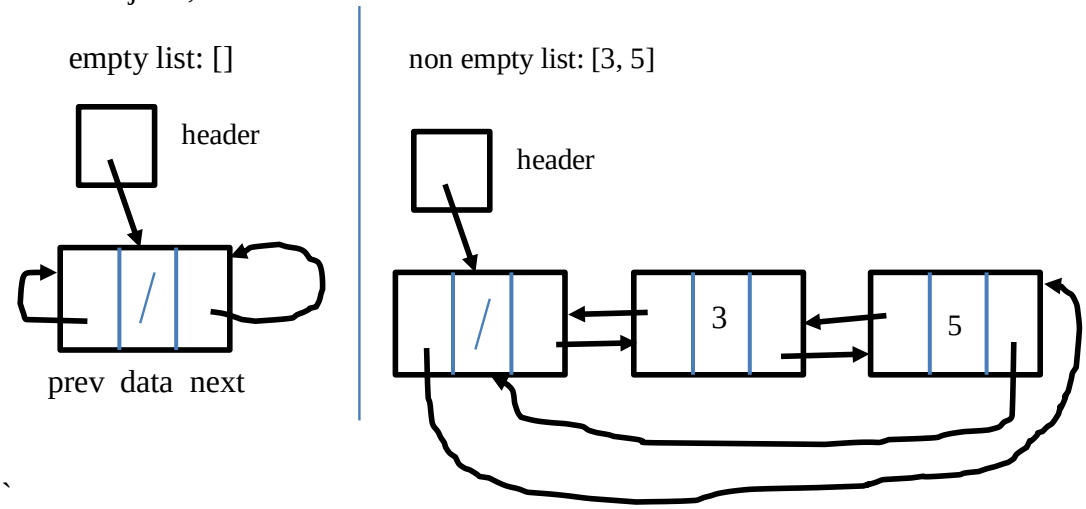
For this question only do not add any other helper methods.

```
/* pre: start != null, dest != null, !start.equals(dest),
     start and dest are both present in this Graph although it is not
     guaranteed a path exists between them.
   post: returns the minimum edge weight in the widest path, as
defined above, from the start vertex to the destination vertex or
INFINITY if there is no path from the start vertex to the destination
vertex.
*/
public double widestPath(String start, String dest) {
     clearAll();
     return help(vertices.get(start), dest, INFINITY);
}

// Complete the help method on the next page.
```

```
private double help(Vertex cur, String dest, double minEdgeSoFar) {
     // Do not add any helper methods on this question.
```

3. Linked Lists - (16 points) Complete the **`placeBetween`** method for a linked list class.

- **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 doubly linked nodes.
- The list maintains a reference to a header node.
- If the list is empty, the header's next and previous references refer to the same node as header.
- If the list isn't empty, the next reference in the header node refers to the first node in the list with data. The previous reference in the header node refers to the last node in the list with data.
- The list does not maintain a size instance variable.
- Null elements are not allowed as elements of the list itself.
- You may use the nested `Node` class.
- **You may not use any other Java classes or methods besides the `equals` method.**
- The figure below shows the structure of the list. The data in the nodes are actually references to objects, not value variables.



`

```
public class LinkedList314<E> {

      private Node<E> header;

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

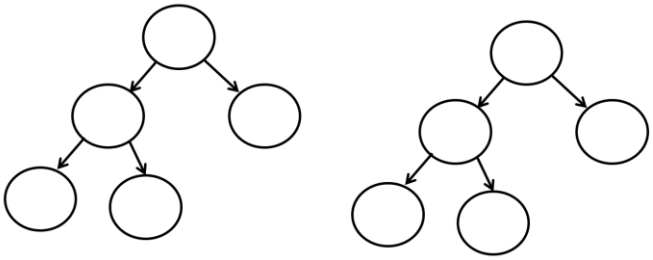
The **`placeBetween`** method accepts 3 parameters, `first`, `second`, and `val`. The method places `val` in between any occurrences of `first` immediately followed by `second`. The method returns the number of elements added to the list. In the examples below the parameters are first, second, and val.

```
[7, 3, 5, 7].placeBetween(7, 3, 2) -> resulting list [7, 2, 3, 5, 7],
returns 1
[]. placeBetween (7, 3, 2) -> resulting list [], returns 0
```

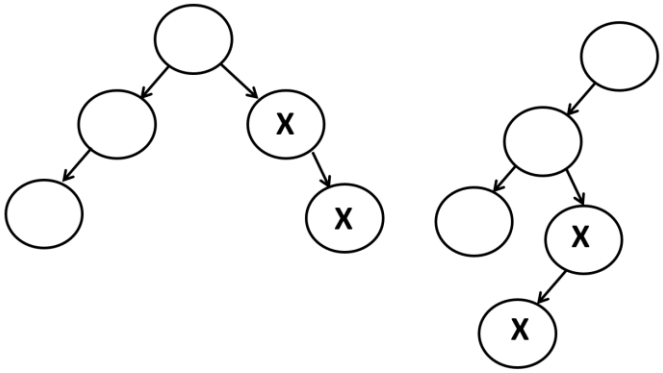
```
[7, 3, 7, 3, 7, 3].placeBetween(3, 7, 6) -> resulting list
[7, 3, 6, 7, 3, 6, 7, 7], returns 2

[7, 7, 5, 7, 7].placeBetween(7, 7, 7) -> result list
[7, 7, 7, 5, 7, 7, 7]

/* pre: first != null, second != null, val != null,
   post: per the problem description. */
public int placeBetween(E first, E second, E val) {
```

4. Trees (16 points) - Write a method that determines how many nodes different two binary trees are based on their structure, ignoring the values actually in the nodes.

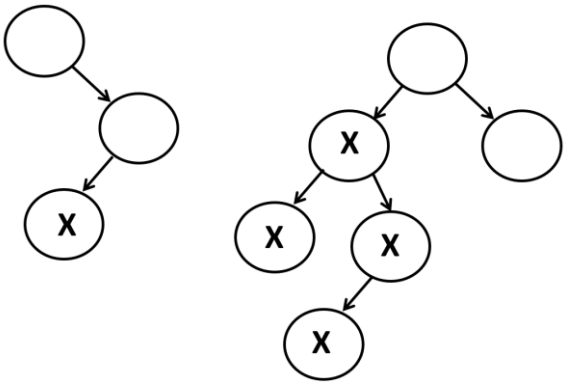
Consider these example. In the first example, the trees are the same and there are no differences.



In the second example the trees have the same number of nodes, but **four** nodes (the ones marked with an X for visualization) are in different places.



In the third example the trees have a different number of nodes and nodes in different locations. Again the nodes considered to be different are marked with an X. Given these two trees there are 5 differences.



Complete an instance method for the `BinaryTree` class, **`numDifferences`**, that returns the number of nodes that are structurally different in two `BinaryTrees` as described above. If both trees are empty they have 0 differences.

```
public class BinaryTree {

     private BNode root; // root == null iff tree is empty

     private static class BNode {

          private int data;
          // left and right child store null if no child on that side
          private BNode left;
          private BNode right; }}// end of nested class and BinaryTree
```

Do not create any new nodes or other data structures. You may use the `BNode` class, but do not use any other Java classes or methods unless you implement them yourself.

```
/*   pre: other != null
     post: per the problem description.
     Neither this BinaryTree or other are altered as a result of
     this method call.*/
public int numDifferences(BinaryTree other) {
```

5. Encoding and Huffman Coding - 16 points. On the Huffman Coding assignment, the decoder rebuilt the Huffman code tree for the compressed file. We then used the tree to convert the data encoded with our Huffman codes to its original encoding.

This question uses a different approach to convert the Huffman encoded data back to its original encoding.

Instead of rebuilding the tree, the decoder uses an array of **`HuffCode`** objects, a new class for this question.

A **`HuffCode`** object is used to represent both the Huffman code for a value and the original integer value.

For example, when compressing "Eerie eyes seen near lake." The new code for 69 ('e') is "10". The corresponding **`HuffCode`** object would be:

```
numBits: 2, encodeVal: 2, decodeVal: 69
```

Instead of storing the Huffman code as a **`String`** the **`HuffCode`** object stores the number of bits the code consists of and its base 10 value. 10_2 is 2_{10}. Storing the number of bits is necessary as other codes may have the same encode value. When compressing "Eerie eyes seen near lake." the new code for 108 ('l") is "0010". When expressed as a base 10 int this also has a value of 2_{10}, but the corresponding **`HuffCode`** object would be:

```
numBits: 4, encodeVal: 2, decodeVal: 108
```

The classes for this question:

```
public class Decoder {

      private static final int PEOF = IHuffConstants.PSEUDO_EOF;
      private static final int BPW = IHuffConstants.BITS_PER_WORD;

      private HuffCode[] codes; // no extra capacity

      private class HuffCode {
            private int numBits; // number of bits in this code
            private int encodeVal; // base 10 value of code
            private int decodeVal; // value to decode to
```

Complete the `decode` method for the `Decoder` class. The method accepts a `BitInputStream` connected to a data source encoded with Huffman coding and a `BitOutputStream`. The `BitInputStream` is positioned at the start of the actual encoded data. The header has already been read and used to build the `Decoder`'s codes array.

The `codes` array contains all the necessary **`HuffCode`** objects for the data source the `BitInputStream` is connect to, including a **`HuffCode`** object to represent the code for the Pseudo-EOF value. None of the elements of the code array equal **`null`**. **The elements of the `codes` array are in no particular order.**

Recall the appropriate methods from the BitInputStream and BitOutputStream classes:

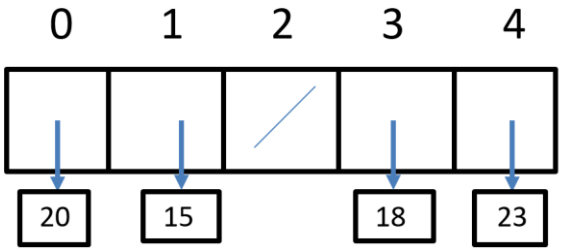
```
public int readBits(int howManyBits) // from BitInputStream
```
Returns the number of bits requested as rightmost bits in returned value, returns -1 if not enough bits available to satisfy the request.

```
public void writeBits(int howManyBits, int value) // from BitOutputStream
```
Writes specified number of bits from `value`. The rightmost `howManyBits` of `value` are written.

Do not use any other Java classes or methods in your answer besides the HuffCode, BitInputStream, and BitOutputStream. In particular, do not use `Strings`. You can of course, use the array.length field.

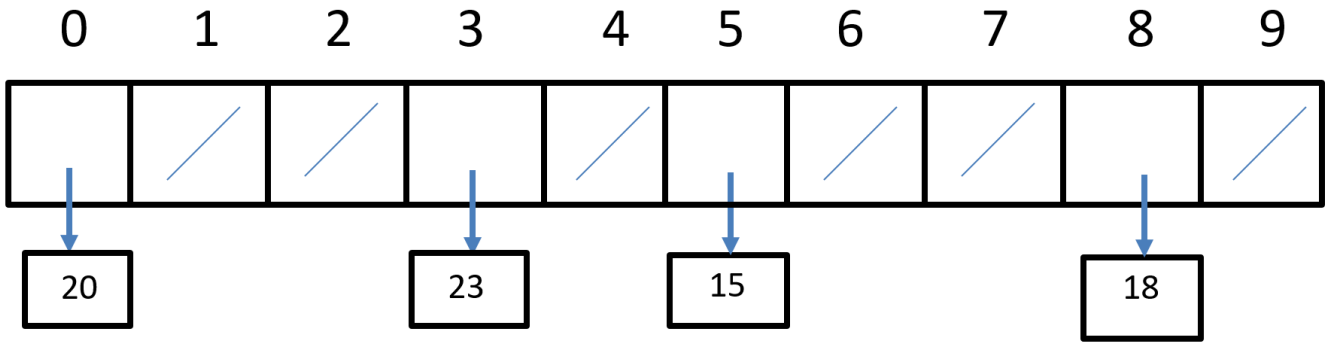
```
/*   pre: bis != null, bis is positioned at the start of the data
portion of the file we are decoding. bos != null, bos is at the start
of the output file.
     post: per the problem description. */
public void decode(BitInputStream bis, BitOutputstream bos) {
```

6. Hash tables - 16 points. Complete the private `resize` instance method for a hash table that uses open addressing to store the elements of the table. Consider the following model of the internal storage container of a hash table that stores four elements. Forward slashes represent variables that store `null`. For illustration purposes the hash code of the elements, not the actual elements are shown.



Implement a method that doubles the capacity of the hash table's array and properly places elements in the new container. Use linear probing and wrap around to resolve collisions when placing elements during the resize operation.

The hash table's container above, the new container after calling the resize method would look like this:



You may use the `hashCode` method (which returns an `int`) and the absolute value (`abs`) method from the `Math` class.

You may create a new array and of course you may use the array length field.

Do not use any other Java classes or methods.

The `Hashtable` class for this question:

```
public class Hashtable<E> {

     private int size;
     private E[] con; // container elements

     private void resize() // to be completed
```

Complete the method on the next page.

```
/* pre: con != null, con.length > 0
   post: per the problem description */
private void resize() {
```