Points off	1	2	3	4	5	6	Total off	

CS 314 – Final Exam – Fall 2018

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, 24 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 r = 0;
     final int LIM = data.length / 2;
     for (int i = 0; i < LIM; i++) {
          for (int j = 0; j < data.length; j += 2) {
               r += data[i] - data[j];
               r += data[i] * data[j];
          }
     }
     return r;
}
```

B. What is the worst case order of the following method?
`N = data.length`. _______________________

```
public static int b(int[] data) {
     int r = 0;
     for (int i = 1; i <= data.length; i *= 2)
          for (int j = 0; j < i; j++)
               if (data[j] == data[i])
                    r++;
     return r;
}
```

C. What is the worst case order of the following method?
Both arrays have the same number of elements, N. _______________________

```
private int c(int[] d1, int[] d2) {
     int r = 0;
     int i1 = 0;
     int i2 = 0;
     while (i1 != d1.length && i2 != d2.length) {
          if (d1[i1] == d2[i2]) {
               r++;
               i2 = 0;
               i1++;
          } else {
               i2++;
          }
     }
     return r;
}
```

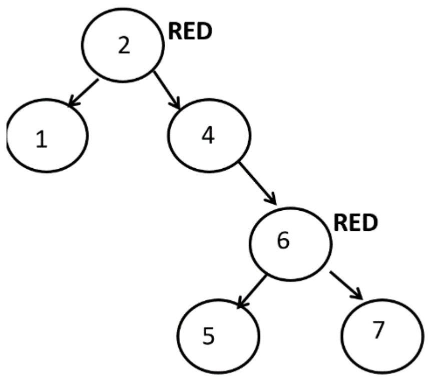
D. The following code uses the naive insertion algorithm presented in lecture. The code takes 2 seconds to complete when N = 2,500. What is the expected time for the code to complete when N is set 5,000? ____________

```
int N = 2500; // second part, 5000 instead
BinarySearchTree<Integer> b = new BinarySearchTree<>();
for (int i = N; i >= -N; i--) {
     if (!b.contains(i)) {
          b.add(i);
     }
}
```

E. What is the result of the following postfix expression? _______________________

```
12 7 - 2 3 * + -10 -
```

F. The tree to the right is not a Red-Black tree. Number and explain all the inconsistencies in the tree. Nodes not labeled RED are BLACK.



G. What is returned by the method call `g(12, 1)`? ____________________

```
public static int g(int x, int y) {
     if (x < y)
          return y;
     return 2 + g(x - 2 * y, y + 1);
}
```

H. We did not implement an iterator for our binary search tree class. If we were to implement an iterator for the binary search tree, what other data structure would we most likely use and why?

____

I. The following values are inserted one at a time in the order shown (left to right) to an initially empty binary search tree using the naive insertion algorithm shown in lecture. What is the depth of the node that contains 73? ____________________

19, 14, 36, 25, 36, 56, 50, 56, 73

J. The following method uses the min heap presented in lecture. It takes the method 2 seconds to complete when N = 1,000,000. What is the expected time for the method to complete when N = 2,000,000? ___________________________

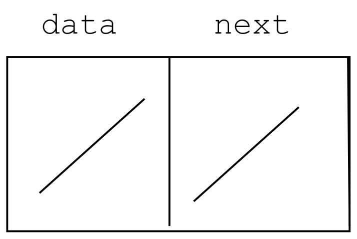
```
public static MinHeap<Integer> j(int N) {
     MinHeap<Integer> result = new MinHeap<>();
     for (int i = 0; i < N; i++) {
          result.add(i);
     }
     return result;
}
```

K. The following method uses the min heap presented in lecture. It takes the method 10 seconds to complete when N = 1,000,000. What is the the expected for the method to complete when N = 2,000,000? Assume `Math.random` is O(1).

____

```
public static MinHeap<Double> k(int N) {
     MinHeap<Double> result = new MinHeap<>();
     for (int i = 0; i < N; i++) {
          result.add(Math.random());
     }
     return result;
}
```

L. 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 all 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 all 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 singly linked node from the linked list examples we did in class and that the fields of the class are all `public`.



```
Node<String> n1;
n1 = new Node<>("AB", new Node<String>("C", null));
Node<String> n2 = new Node<>(n1.next.data, n1.next);
n1.next.next = n2;
```

M. What is the worst case order of the following code?
Each list contains N elements.

____

```
public static int m(LinkedList<Integer> l1, ArrayList<Integer> l2) {
     int r = 0;
     for (int i = 0; i < l1.size() - 1; i++) {
          int x = l2.get(i);
          if (l1.get(i + 1) == x && x < l2.size()) {
               r += l1.get(i);
          }
     }
     return r;
}
```

N. What is output by the following code? Recall the **toString** for maps: **{k1=v1, k2=v2}**

____

```
TreeMap<Integer, String> tm = new TreeMap<>();
tm.put(0, "B");
tm.put(-3, "A");
tm.put(5, "C");
tm.put(2, tm.put(0, "D"));
tm.put(-3, "Z");
tm.remove(0);
System.out.println(tm);
```

O. What is returned by the method call **o(8, 1)**? ______________________

```
public static int o(int x, int y) {
     if (x <= 0)
          return x;
     return y + o(x - 2, y + 1) + o (x - 4, y + 1);
}
```

P. The following method takes 0.5 seconds to complete when N = 20.
What is the expected time for the method to complete when N = 24? _________________

```
public static double p (int N) {
     if (N < -5.0)
          return 0.25;
     return p(N - 1) + (N / 0.5) + p(N - 1);
}
```

Q. Consider the following adjacency matrix representing a directed, unweighted graph. Each row indicates the edges that originate at the vertex specified by the row index using zero based indexing. A 1 indicates an edge exists, a 0 indicates no edge exists. Draw the graph represented by the given adjacency matrix---------------->

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

R. We are going to use a linked list to implement a pure stack (push, pop, top, isEmpty only). Should we use a singly linked list, a doubly linked list, or is there no difference? Explain your answer.

____

S. The following values are inserted in the order shown (left to right) to an initially empty max heap. Draw the resulting max heap. ---------------------->

```
17, 25, 18, 12, 25
```

T. How did we define the central node in our graph class when the graph was undirected and unweighted?

____

U. The following method takes 10 seconds to complete when N = 1,000,000. What is the expected time for the method to complete when N = 4,000,000? _________________

```
public static HashSet<Integer> v(int[] data) {
     HashSet<Integer> hs = new HashSet<>();
     for (int x : data)
          if (!hs.contains(x))
               hs.add(x);
          else
               hs.remove(x);
     return hs;
}
```

V. Which lines from the following code contain compile errors? _________________

```
ArrayList av = new ArrayList(); // 1
av.add("AB"); // 2
av.add("" + av.get(0).charAt(0)); // 3
av.add(av.toString()); // 4
av.add((int) av.toString().charAt(0)); // 5
```

W. What is output by the following code? ___________________________________

```
int x1 = IntStream.of(-2, 5, 5, 10, -6)
          .map(x -> x / 2)
          .filter(y -> y > 0)
          .sum();
System.out.print(x1);
```

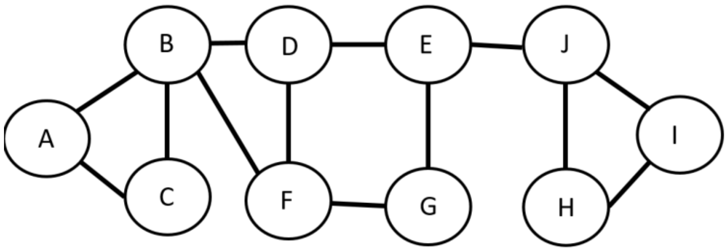
X. On the Huffman assignment files sometimes the file produced by the algorithm was larger than the original file. Why did this occur?

____

2. Graphs - (16 points) - An undirected graph is said to be ***connected*** if there is a path, a series of one or more edges, from each vertex to every other vertex. In other words, it is possible to start at an arbitrary vertex and follow a path that eventually reaches every other vertex. It is permissible for the path to reuse edges and revisit vertices.

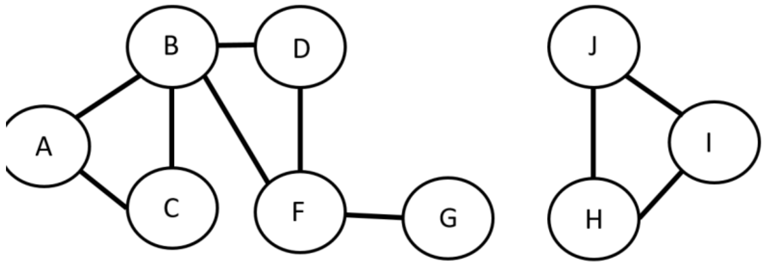
If a graph is not connected it is said to be *disconnected*.

Consider the following example:

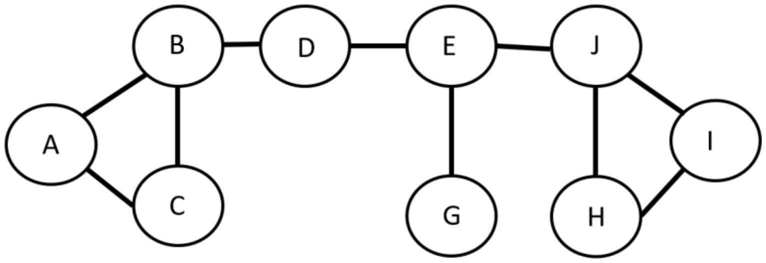


The graph shown is connected.

There are however, three ***critical*** vertices in the graph, B, E, and J. If any one of the vertices and its associated edges, were removed, the Graph would no longer be connected. For example, if the E vertex were removed the Graph would become:



On the other hand, we can remove the F vertex from the original graph and the graph is still connected.



Complete an instance method from the Graph class used in assignment 11 to determine if a Graph is connected or not while ignoring a given vertex. In other words, the method is sent a String indicating which vertex in the graph to ignore when checking to see if the graph is still connected.

Do not use recursion in your answer.

You may create and use a single Queue object. Do not create any other new data structures other than `Iterators`.

Recall the classes and methods associated with the **`Graph`** class.

```
public class Graph {

    // The vertices in the graph.
    private Map<String, Vertex> vertices;

     // 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 use the methods from the **`Map`**, **`Iterator`**, and **`List`** interfaces and the equals method for **`String`**s.

Do not alter this **`Graph`**'s **`Map`** of vertices in any way and do not alter any **`Vertex`** object other than its **`scratch`** and **`prev`** variables.

Recall, do not use recursion in your answer.

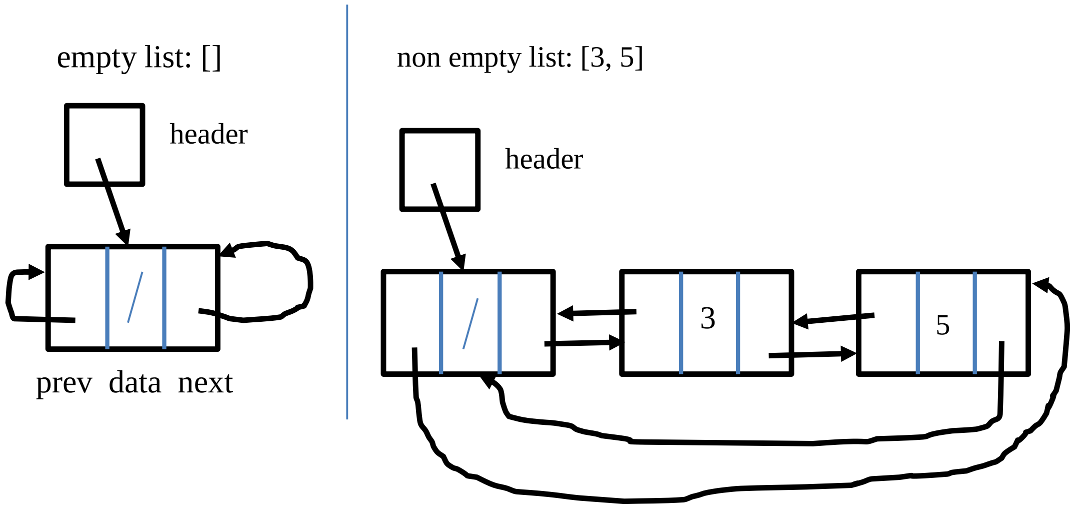
You may create and use a single Queue object. Do not create any other new data structures other than `Iterators`.

Assume the graph has at least two vertices.

```
/* pre: ignore is the name of a Vertex in this Graph.
     This Graph has at least two vertices.
   post: return true if this Graph is connected even with the Vertex
     specified by ignore not included in any paths.
     This graph is not altered as a result of this method. */
public boolean isConnected(String ignore) {
```

3. Linked Lists - (16 points) Complete the `removeAll` 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.
- The list may contain nodes whose data reference is set to **`null`**.
- 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 **`removeAll`** method removes all instances of the specified value from the list and returns the number of values removed.

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

```
/* pre: none
   post: Remove all occurrences of tgt from this list.
     The relative order of remaining elements is not altered.
     Returns the number of elements removed. */
public int removeAll(E tgt) {
```

4. Trees (13 points) - Consider a binary tree that stores `int` values. For this method we are interested in paths from the root node to descendant nodes and the sum of the values in the nodes that make up the path. The root node must be included in the path and nodes may not be skipped.

Write a method, **`numPaths`**, that returns the number of paths that exist in the tree from the root node to descendant nodes such that the sum of the values of all the nodes in the path equal some target value. Additionally, the path must contain at least one instance of a required value.

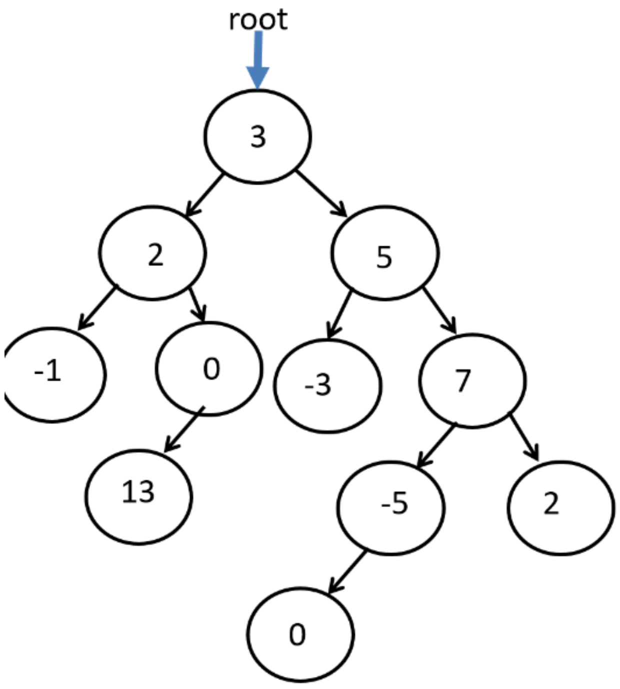
```
public class BinaryTree {

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

     // nested BNode class
     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;
     }
}
```

Consider the following tree and method calls to **`numPaths(int target, int required)`**:



`numPaths(5, 3) returns 3`
The paths are:
3, 2
3, 2, 0
3, 5, -3

`numPaths(5, 5) returns 1`
The path is:
3, 5, -3

`numPaths(18, 2) returns 1`
The paths is:
3, 2, 0, 13

`numPaths(0, 5) returns 0`

`numPaths(13, 0) returns 0`

`numPaths(-5, 0) returns 0`

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.

```
/*   pre: size() > 0 (this tree is not empty)
     post: per the problem description.
     This BinaryTree is not altered as a result of this method call.*/
public int numPaths(int tgt, int req) {
```

5. Encoding - 15 points. Parity bits are yet another way of detecting errors in binary data. Given a sequence of binary data, a parity bit is added to make the number of 1's in the sequence plus the parity bit odd or even.

In this question we use even parity. The number of bits in a chunk, including the data and the parity bit, must be even or there is an error somewhere in the chunk. We can't tell which bit is in error.

For example, if our data sequences were 4 bits long, we add a fifth bit so that the number of 1's in the 5 bits (4 data bits + 1 parity bit) is even. Again, this is called even parity.

Even Parity Example:

data bits	number of 1's in data	parity bit so that the total number of ones is even	complete sequence data + parity bit
`0000`	`0`	`0`	`0000`<u>`0`</u>
`0010`	`1`	`1`	`0010`<u>`1`</u>
`1010`	`2`	`0`	`1010`<u>`0`</u>
`1011`	`3`	`1`	`1011`<u>`1`</u>

A data chunk of **`0000`** and a parity bit of **`1`** indicate an error in a file encoded with even parity bits. Note, we cannot determine if the error is in the data or the parity bit.

For this question, the file includes header information describing the format of the data file.
The header information does not include any parity bits.

The first part of the header is a 6-bit integer indicating how many bits are in each data chunk. As a data chunk cannot have 0 bits, add 1 to the value to get the actual number of bits in the data chunks. For example, if the 6 bit integer is **`001011`** `readBits(6)` returns `11`, so we assume the data chunks are 12 bits each.

The second and final part of the header is a 32-bit integer indicating the number of data chunks in the file.

Your method shall read the header from the data source and then read all data chunks, writing out all data bits, and comparing the chunk to the parity bit at the end of the chunk.

The method shall return an **`ArrayList<Integer>`** indicating errors in the data. The elements of the list indicate which chunks had a disparity between the data and the parity bit. Report the number of the chunk in the data sequence using zero based indexing, not the actual data. So for example if we read 20 chunks of data and there was a discrepancy between the data and the parity bit in the first, tenth, and fifteenth data chunks the returned list would be **`[0, 9, 14]`**.

Write a method that accepts a **`BitInputStream`** connected to a source encoded using parity bits. It writes out the decoded data via a **`BitOutputStream`**.

You may use the **`BitInputStream`** and **`BitOutputStream`** classes from the Huffman Encoding assignment and their **`readBits`** and **`writeBits`** methods:

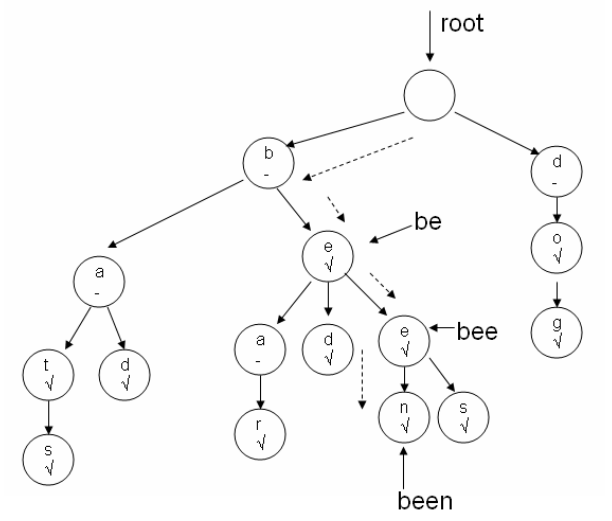
```
public int readBits(int howManyBits) // from BitInputStream
```
Returns the number of bits requested as rightmost bits in returned value, returns -1 if not enough bits available to satisfy the request.

```
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 data structures besides the **`ArrayList<Integer>`** used to report results. You may use whatever **`ArrayList`** instance methods you want on the resulting **`ArrayList`**.
Do not use any other Java classes or methods.

```
/* in != null, out != null, in and out are already connected to the
   input and output files.
   post: Per the problem description. */
public ArrayList<Integer> writeFile(BitInputStream in, BitOutputStream out)
            throws IOException {
      ArrayList<Integer> errors = new ArrayList<>();
```

6. New Data Structures and recursion - 16 points A *trie* is a tree based data structure that stores strings (in an interesting way) and supports fast look up of words given a prefix as well as other operations. The root node of the tree does not contain any values. Descendants of the root node contain a single character and a boolean to indicate if the characters in the path from the root to that node form a valid word. Here is a very small example of a trie.



Strings are built up by starting at the root and descending through the tree. Paths only move down, never back up the tree. A dash in a node indicates the string built up along that path is **not** a valid word, although it is a prefix to other valid words. A check in a node indicates the string built up along that path **is** a valid word.

The dashed arrows show how the word "been" is built. The path to "be" and "bee" are also shown. The words present in the above trie are: bat, bats, bad, be, bear, bed, bee, been, bees, do, and dog.

Do you see how a Trie would be useful for predictive texting or typing?

Here is the Trie class for this question:

```
public class Trie

      private TNode root; // the root of this Trie.

      public Trie() { root = new TNode(); }

      /* Returns the TNode at the end of the path in this Trie specified by
      pre or null if this Trie does not contain this prefix. */
      private TNode getNodeForPrefix(String pre)

      private static class TNode {
         // true if the path from root to this node forms a valid word.
           private boolean word;

         // The char in this node, always a lower case English letter.
           private char ch;

           // My child nodes. null if I am a leaf.
           private List<TNode> children;

           /* equals method overridden correctly.
              Two TNodes are considered equal if that have the same
              char ch. The boolean word and child list are ignored.*/
           public boolean equals(Object o)
```

Complete a method that returns the elements (words) of this **`Trie`** that have a give prefix. If the **`Trie`** was the small one shown in the problem statement and the given prefix was "bee" the method would return [bee, been, bees]. The elements in the result can appear in any order.

Do not use any data structures besides the **`Trie`**, **`TNode`**, the resulting **`List`**, and **`Strings`**.
Do not create any additional **`List`**s.

```
/* pre: pre != null, pre.length() >= 1
   post: return the elements of this Trie that have the given prefix.
This Trie is not altered by this method. */
public List<String> getWords(String pre) {
     List<String> result = new ArrayList<>();
     TNode start = getNodeForPrefix(pre);
```