1. [25 points] FindTriples returns true if there are three equal integers in the array A.

   ```java
   public static boolean FindTriples(int[] A) {
       for (int i = 0; i < A.length; i++)
           for (int j = i+1; j < A.length; j++)
               for (int k = j+1; k < A.length; k++)
       return false;
   }
   ```

   (a) What is the running time of this method?
   \( O(n^3) \)

   (b) Describe a more efficient algorithm than the one above and give its running time. No code is needed in your answer.
   We may first sort the array using mergesort and then check if \( A[i-2] == A[i] \) for \( i \) from 2 to \( A.length \). Sorting takes \( O(n \lg n) \) and the checking takes \( O(n) \). The total time is \( O(n \lg n) \).

2. [25 points] Suppose the class `Link` is used to define a linked list:

   ```java
   class Link {
       public long data;
       Link next;
       public Link(long d) { data = d; }
   }
   class LinkList {
       private Link first; ...
       public LinkList reverse() { ... }
   }
   ```

   Please complete the method `reverse` that reverses a linked list without destroying the original list. Syntax errors are allowed. What is the running time of your implementation?

   ```java
   public LinkList reverse() {
       Link current = this.first;
       LinkList rev = new LinkList();
       while (current != null) {
           Link node = new Link(current.data);
           node.next = rev.first;
           rev.first = node;
           current = current.next;
       }
       return rev;
   }
   ```

   The running time is \( O(n) \).
3. [25 points] The following method computes the size of a binary tree rooted at node x.

```java
public static int size( Node x ) {
    if (x == null) return 0;
    return 1 + size(x.leftChild) + size(x.rightChild);
}
```

Please provide another implementation of this method without recursive calls. What is the running time of your method?

```java
public static int size(Node x) {
    if (x == null) return 0;
    Stack<Node> stack = new Stack<Node>();
    int count = 0;
    while (true) {
        count++;
        if (x.leftChild != null) {
            if (x.rightChild != null) { stack.push(x.rightChild); }
            x = x.leftChild;
        } else if (x.rightChild != null) x = x.rightChild;
        else if (!stack.isEmpty()) x = stack.pop();
        else break;
    }
    return count;
}
```

The running time is O(n).

4. [25 points] Suppose a Red-Black tree is initially empty. Please draw the resulting Red-Black trees after each insertion of the following numbers (in that order): 1, 2, 3, 4, 5.

```
   1

  1   2

  1   2   3

  1   2   3

  4

  1   2   3   4

  1   2   3   4

  5
```