Name: ________________________________

Unless indicated otherwise, here’s the TreeNode class for this test.

```java
public class TreeNode{
    public String info;
    public TreeNode left,right;
    public TreeNode(String s, TreeNode lptr, TreeNode rptr){
        info = s;
        left = lptr;
        right = rptr;
    }
}
```
You must develop a program to sort long strings representing strands of DNA. A strand is composed of the letters 'c', 'a', 'g', and 't'; for example some strands are shown below on the left.

<table>
<thead>
<tr>
<th>strands</th>
<th>time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgaaatctcctttctatattaagc</td>
<td>4000</td>
</tr>
<tr>
<td>aatggcattaggccggttaa</td>
<td>8000</td>
</tr>
<tr>
<td>gtacgcctcgccctataactact</td>
<td>10000</td>
</tr>
<tr>
<td>tagggaatgcgcgaacgcaggctccgcgcg</td>
<td>100</td>
</tr>
</tbody>
</table>

The people who hire you require you to use insertion sort. You’ve developed a program and timed it on three input files. The results are shown above on the right in a table with the number of strands and the time to sort the strands.

**Part A**
Given this data, how long will it take the program to sort 20,000 strands of DNA? Show your reasoning.

**Part B**
Given this data, how long will it take the program to sort 1,000 strands of DNA? Show your reasoning.

**Part C**
You determine that the first letter in a strand is equally likely to be 'a', 'g', 't', or 'c'. You then make the following suggestion to sort the strands of DNA stored in a file.

1. As the strands are read in, store them in one of four vectors: one for strands beginning with 'a', one for strands beginning with 'g', one for 't', and one for 'c'.

2. Sort each of the four vectors.

3. Combine the four vectors into one vector taking first the strands from the 'a' vector, then the 'c' strands, then the 'g' strands, followed by the 't' strands.

This method will correctly sort the strands. Assuming there are roughly the same number of strands starting with 'a', 'c', 'g', or 't' in the files you’ll be sorting, roughly how long will it take with this new method to sort a file of 8,000 strands. Briefly justify.
PROBLEM 2:  (eerst (8 points))

Part A
The function sort below uses two helper functions and a binary search tree to sort a vector. The code is correct. What is the big-Oh complexity of sort assuming the vector list is in random order? Briefly justify your answer.

Part B
What is the complexity if the vector list is already sorted? Briefly justify your answer.

```java
/**
 * Add s to a search tree t, return the modified search tree
 */
TreeNode insert(TreeNode t, String s) {
    if (t == null) return new TreeNode(s,null,null);
    else if (s <= t.info) t.left = insert(t.left,s);
    else t.right = insert(t.right,s);
    return t;
}

/**
 * Copy values from a tree into an ArrayList
 */
void traverse(TreeNode t, ArrayList<String> list) {
    if (t != null) {
        traverse(t.left, list);
        list.add(t.info);
        traverse(t.right,list);
    }
}

void sort(ArrayList<String> list)  // post: list is sorted
{
    TreeNode t = null;
    for(int k=0; k < list.size(); k++) {
        t = insert(t, list[k]);
    }
    list.clear();
    traverse(t,list);  // list is now sorted
```
PROBLEM 3: (store e-sort (20 points))

Part A (5 points)
Put the letter from the list of speakers on the left next to the phrase that person did say or could have said based on what we've studied in class. You can use a letter more than once, you don't need to use all the letters.

A. Barack Obama 1. ______ Bubblesort’s not the way to go
B. Edsger Dijkstra 2. ______ I invented quicksort
C. Tony Hoare 3. ______ One of these days I gotta get myself organized
D. Fred Brooks 4. ______ My algorithm helps route packets in the Internet
E. Travis Bickle 5. ______ My compression algorithm is greedy and optimal
F. Niklaus Wirth
G. David Huffman

Part B (5 points)
The code below correctly sorts an array of String values. What is the big-Oh complexity of this code? Justify your answer. The java.util.PriorityQueue class is implemented using a heap.

```java
public static void psort(String[] list){
    PriorityQueue<String> pq = new PriorityQueue<String>();
pq.addAll(Arrays.asList(list));
    int index = 0;
    while (pq.size() != 0){
        list[index++] = pq.remove();
    }
}
```
Part C (5 points)
The code below sorts an array of Strings representing DNA by creating arrays for every possible 4-character prefix, e.g., "aaaa" "aaag" "aaat" ... "agtc" ... "gatt" ... "tttt"; sorting each of these arrays, and then combining these sorted arrays together.

```java
public static void dnasort(String[] dna){
    Map<String,ArrayList<String>> prefixMap = new TreeMap<String,ArrayList<String>>();
    for(String s : dna){
        String prefix = s.substring(0, 4);
        ArrayList<String> list = prefixMap.get(prefix);
        if (list == null){
            list = new ArrayList<String>();
            prefixMap.put(prefix, list);
        }
        list.add(s);
    }
    for(ArrayList<String> list : prefixMap.values()){ Collections.sort(list); }
    ArrayList<String> combined = new ArrayList<String>();
    for(ArrayList<String> list : prefixMap.values()){ combined.addAll(list); }
    System.arraycopy(combined.toArray(new String[0]), 0, dna, 0, dna.length);
}
```

The code above is faster than calling `Arrays.sort(dna)` for an array of one-million strings representing dna strands (e.g., all the strings contain just the characters 'a', 'g', 't', 'c'). The code is slower when sorting an array of one-thousand strands. Explain why.
Part D (5 points)
The code below reads in a file of integer values, stores these in arrays and an `ArrayList` and then sorts these using the `java.util` sorts. In class we discussed that `Arrays.sort` for an `int[]` uses a variant of quicksort whereas both `Arrays.sort` for `Object[]` and `Collections.sort` use a modified merge sort.

In the code below the values in the lists `list`, `alist`, `ilist` are the same before the sorts are called.

```java
public void sortParadise() throws FileNotFoundException{
    ArrayList<Integer> alist = new ArrayList<Integer>();
    Scanner s = new Scanner(new File("demonicintegers.txt"));
    while (s.hasNextInt()){
        alist.add(s.nextInt());
    }
    int[] list = new int[alist.size()];
    for(int k=0; k < alist.size(); k++){
        list[k] = alist.get(k);
    }
    Integer[] ilist = alist.toArray(new Integer[0]);

    Collections.sort(alist);
    Arrays.sort(ilist);
    Arrays.sort(list);
}
```

When this code is executed the last call to `Arrays.sort`, when the `int[] list` array is sorted, results in a stack-overflow error. This is because the file `demonicintegers.txt` contains a worst-case ordering of 250,000 integers. However, if this line is added after the `while` loop:

```java
Collections.shuffle(alist);
```

then there is no stack-overflow. Explain briefly why `Arrays.sort(ilist)` doesn’t generate a stack overflow but `Arrays.sort(list)` does and why shuffling the elements fixes the “problem”.

6