NAME:__________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max</th>
<th>Obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3 (a)</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3 (b)</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3 (c)</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4 (a)</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4 (b)</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>4 (c)</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>110</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1:

1. Is it true that in the worst case, a red-black tree insertion requires $O(1)$ rotations?

2. Is it true that walking a red-black tree with $n$ nodes in in-order takes $\Theta(n \log n)$ time?

3. Given a red-black tree with $n$ elements, how fast can you sort them using the tree?

4. How fast can we build a red-black tree with $n$ elements?

5. If a data structure supports an operation FOO such that a sequence of $n$ FOO’s takes $O(n \log n)$ time in the worst case, then the amortized time of a FOO operation is $\Theta(\quad)$ while the actual time of a single FOO operation could be as low as $\Theta(\quad)$ and as high as $\Theta(\quad)$. 
Problem 2:
In this problem we consider a data structure $\mathcal{D}$ for maintaining a set of integers under the normal INIT, INSERT, DELETE, and FIND operations, as well as a COUNT operation, defined as follows:

- INIT($\mathcal{D}$): Create an empty structure $\mathcal{D}$.
- INSERT($\mathcal{D}, x$): Insert $x$ in $\mathcal{D}$.
- DELETE($\mathcal{D}, x$): Delete $x$ from $\mathcal{D}$.
- FIND($\mathcal{D}, x$): Return pointer to $x$ in $\mathcal{D}$.
- COUNT($\mathcal{D}, x$): Return number of elements larger than $x$ in $\mathcal{D}$.

Describe how to modify a standard red-black tree in order to implement $\mathcal{D}$ such that INIT is supported in $O(1)$ time and INSERT, DELETE, FIND, and COUNT are supported in $O(\log n)$ time.
Problem 3:

A pharmacist has $W$ pills and $n$ empty bottles. Let \{\(p_1, p_2, \ldots, p_n\)\} denote the number of pills that each bottle can hold.

a) Describe a greedy algorithm, which, given $W$ and \{\(p_1, p_2, \ldots, p_n\)\}, determines the fewest number of bottles needed to store the pills. Prove that your algorithm is correct (that is, prove that the first bottle chosen by your algorithm will be in some optimal solution).
b) How would you modify your algorithm if each bottle also has an associated cost \( c_i \), and you want to minimize the total cost of the bottles used to store all the pills?

Give a recursive formulation of this problem (formula is enough). You do not need to prove correctness.

(Hint: Let \( \text{MinPill}[i, j] \) be the minimum cost obtainable when storing \( j \) pills using bottles among 1 through \( i \). Thinking of the 0-1 knapsack problem formulation may help.)

c) Describe briefly how you would design an algorithm for it using dynamic programming and analyse its running time.
Problem 4:

In this problem we look at the amortized cost of insertion in a dynamic table. Initially the size of the table is 1. The cost of insertion is 1 if the table is not full. When an item is inserted into a full table, it first expands the table and then inserts the item in the new table. The expansion is done by allocating a table of size 3 times larger than the old one and copying all the elements of the old table into the new table.

a) What is the cost of the $i$-th insertion?

b) Using the accounting method, prove that the amortized cost of an insert in a sequence of $n$ inserts starting with an empty table is $O(1)$. 
b) Prove the same amortized cost by defining an appropriate potential function. You can use the standard notation $\text{num}(T)$ for the number of elements in the table $T$ and $\text{size}(T)$ for the total number of slots (maximum size) of the table.