CS 2420 – Written 2
10 Points
Due Sept 22, 2010 **DUE AT CLASSTIME**

Turn in your written homework in class by folding it in half lengthwise and putting your names on the outside or submit the homework through Eagle.

Note, these exercises may be done in groups of one, two, or three. If more than one person is involved, list all the names on one set of answers. Groups may change throughout the semester. Answers should not be compared with others not in your group. You will learn much more by working in a group than you will learn working by yourself.

1. The terminology \( f(n) \) is \( O(n^2) \) is equivalent to saying \( f(n) \) is of order \( n^2 \) or saying \( f(n) \) is of complexity \( n^2 \). For each program segment (a-d) shown below:
   a. Find the complexity
   b. Select an appropriate picture from the list below (A-F) (or draw one of your own) to justify your answer.

   a. for (int k = 0; k<n;k++)
      for (int j = 0; j < k; j++) // Notice j ends at k (not n)
      cout << k << j;

   b. void doit(int n)
      { if (n <=0) return;
         doit(n/2);
         doit(n/2);
         for (int i=0;i<n;i++)
            cout << n << i;
      }

   ![Diagram A]
   ![Diagram B]
   ![Diagram C]
   ![Diagram D]
   ![Diagram E]
   ![Diagram F]
c.    void doit(int n)
     {    if (n <=0) return;
          doit(n/2);
          doit(n/2);
          cout << n;
     }

d.    int doit(int look,int beg, int end)
     {    if (beg > end) return -1;
          mid = (beg + end)/2;
          if (A[mid] == look)
              return mid;
          if (look < A[mid])
              return doit(look, beg,mid-1);
          return doit(look, mid+1,end);
     }

2. Consider the following sample data. For each set of timing information, indicate the complexity of the algorithm.

   a.
   \[ \begin{array}{|c|c|}
   \hline
   n & T(n) \\
   \hline
   2 & 5 \\
   4 & 5 \\
   8 & 5 \\
   16 & 6 \\
   32 & 5 \\
   \hline
   \end{array} \]

   b.
   \[ \begin{array}{|c|c|}
   \hline
   n & T(n) \\
   \hline
   2 & 5 \\
   4 & 10 \\
   8 & 20 \\
   16 & 40 \\
   32 & 80 \\
   \hline
   \end{array} \]

   c.
   \[ \begin{array}{|c|c|}
   \hline
   n & T(n) \\
   \hline
   2 & 3 \\
   4 & 10 \\
   8 & 15 \\
   16 & 20 \\
   32 & 25 \\
   \hline
   \end{array} \]
The next questions (3,4,5,6) deal with csilm.usu.edu. From CS3, go to Complexity. In this ILM, we are trying to show a visual representation of work by letting a cube represent a certain amount of work – as we are pretty good at estimating the volume of something. While it is interesting to say, "We expect this algorithm to take 27 steps," a more general solution gives expected work in terms of the size of the problem.

3. Consider the Complexity of Loops. Pick values for SIZE1, SIZE2, and SIZE3. If the Loop1 index = SIZE1, the Loop2 index = SIZE2, and the Loop3 index = SIZE3, what is the complexity of the loop code?

4. Consider Recursive Complexity (activity 2). With recursion, several things contribute to the complexity: the total number calls made and average amount of work done in each call. If we knew those two things, we could multiply to get total work. However, in recursion, we don’t just say, "Do it twelve times." List the items that control the number of times a recursive piece of code is called. Experiment with the ILM enough so that you can predict both what picture is drawn and the total complexity.

5. Consider Recursive Complexity (activity 2) but select the "formula" analysis. Mathematicians have found a formula which describes the complexity for a variety of simple cases. From the formula analysis, you can see another way of finding complexity. Your answer will be the same as the one you compute via the pictorial analysis. The parameters to the formula are a bit odd. Students want to define the parameters themselves, but life is not so simple. The mathematicians have decided what is useful. You must give them what they want (not what you want).
a: is the number of recursive calls you make from each single call
b: is the number of pieces the original piece is divided into
k: is the exponent on n that defines the work done in a single call (disregarding the recursive calls)

Using the "Let Me Try " feature, experiment with various settings for the recursive algorithm. Identify a, b, k, and the resultant Big O. Verify that you are right.

Set the parameters (work for a single call, recursive calls, recursive parameter, initial n value) to create the pseudo code below. What are the values of a, b, and k? What is the complexity?

```cpp
void doit (int n)
    { if (n < 1)
        return;
     int x = 0;
    for (int i=0; i < n; i ++)
        x++;
    doit (n/3);
    doit (n/3);
    }
```

6. Go to Verify Complexity from Runtime Data (exercise 7). In this example, we are trying to verify the complexity of an algorithm from test data. By clicking "Generate" the system will generate runtime data for various sized problems. By clicking "Custom", the system allows you to generate your own runtime data. Check your results in problem 2 of the assignment by entering in the data shown in your assignment. From the plotted data, you are to estimate the complexity. Remember with complexity, the constant multiplier has been removed. So when we say O(n) we may mean it varies as 5n or 10n or .5n. From this ILM, you can test your complexity match for the data by selecting both the complexity and the multiplier. The vertical green line shows the point on the x axis for which all large x values yield a time BENEATH your estimate. In other words, you have found an upper bound from that point on.

Using this ILM, (i) what is the complexity and (ii) what is the multiplier for the following data:

(a)

<table>
<thead>
<tr>
<th>Size</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
</tr>
<tr>
<td>16</td>
<td>78</td>
</tr>
<tr>
<td>32</td>
<td>316</td>
</tr>
<tr>
<td>48</td>
<td>657</td>
</tr>
<tr>
<td>64</td>
<td>1162</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Size</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>32</td>
<td>46</td>
</tr>
<tr>
<td>48</td>
<td>77</td>
</tr>
</tbody>
</table>
7. For the AVL tree below, which of the following are correct? Which are incorrect? (Test your answers by going to csilm.usu.edu CS3/Binary Search Trees. Delete the whole tree and then add nodes on each level at a time.)
   a. If a key with value 42, 29, or 27 is inserted in the tree, it does not lose the AVL property.
   b. After the insertion of 20, a single rotation is necessary to restore the AVL property of the tree.
   c. A double rotation is necessary to restore the AVL property after the insertion of 20. This makes 16 the new root of the tree.
   d. A single rotation is needed to restore the AVL property if any node with value less than 14 is added to the present tree.

```
    28
   /   \
  14    40
 /     /
12     16
```

8. An AVL tree of height 4 must have at least ______ nodes
   a. 21
   b. 7
   c. 14
   d. 12
   e. None of the above.

9. Using csilm.usu.edu (CS3/Binary Search Trees). Create a tree which has the AVL tree property and is of height 4 (a single node is of height 1) using the minimal number of nodes. Grab a screen shot of the tree you create and turn it in with your assignment. How many nodes are required? Note the use of Undo to revert back to an earlier state.

10. Consider the following AVL tree. Show the tree after insertion of 17 followed by deletion of 28.

```
    28
   /   \
  14    42
 /     /
5      40
```

11. Show the results of inserting items 1 through 8 in order in an initially empty AVL tree. (Test your answer by going to csilm.usu.edu CS3/Binary Search Trees)

12. In the following AVL tree, show the tree after deletion of 15.
13. Show an AVL tree for which a deletion will cause multiple rotations. Note, we count a double rotation as one rotation not two (it’s just fancier). Can you create a tree in which an addition causes multiple rotations?