Text Terminology:

Problem - question we seek an answer for, e.g., "What is the largest item in an array?"

Parameters - variables with unspecified values

Problem instance - assignment of values to parameters, i.e., the specific input to the problem

```
array S: 0 1 2 3 4 5 6
        5 10 2 15 20 1 11

n: 7
```

(Number of elements)

Algorithm - step-by-step procedure for producing a solution

Basic operation - fundamental operation in the algorithm (i.e., operation done the most)

Generally, we want to derive a function for the number of times that the basic operation is performed related to the problem size.

Problem size - input size. For algorithms involving lists/arrays, the problem size is the number of elements ("n").

Text Notation: text uses a C++ (Java) pseudo code

Variable and parameter names must be declared before usage with a specified type:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>keytype</td>
<td>items from an ordered set (we can compare two items for =, &lt;, &gt;, etc.)</td>
</tr>
<tr>
<td>index</td>
<td>integer used as an array index</td>
</tr>
<tr>
<td>number</td>
<td>either integer or real numbers</td>
</tr>
<tr>
<td>bool</td>
<td>a Boolean that can be assigned either &quot;true&quot; or &quot;false&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>C++ symbol</th>
<th>Comparison Operator</th>
<th>C++ syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>&amp;&amp;</td>
<td>x = y</td>
<td>(x == y)</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not</td>
<td>!</td>
<td>x ≤ y</td>
<td>(x &lt;= y)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x ≥ y</td>
<td>(x &gt;= y)</td>
</tr>
</tbody>
</table>

Control Structures:

```
for (i = 1; i <= n; i++) { while (j > 1) { if (score >= 90) {
    // loop body                              // loop body
    cout << 'A';
} else if (score >= 80) {
    cout << 'B';
} else if (score >= 70) {
    cout << 'C';
} else if (score >= 60) {
    cout << 'D';
} else {
    cout << 'F';
}
```

<table>
<thead>
<tr>
<th>Meaning</th>
<th>C++ syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>assignment</td>
<td>x = 4;</td>
</tr>
<tr>
<td>increment: x = x + 1</td>
<td>x++</td>
</tr>
<tr>
<td>decrement: x = x - 1</td>
<td>x--</td>
</tr>
<tr>
<td>x = x + 4</td>
<td>x += 4</td>
</tr>
</tbody>
</table>
1. The recursive definition of the Fibonacci sequence (0, 1, 2, 3, 5, 8, 13, ...) is:
   \[ f_0 = 0 \]
   \[ f_1 = 1 \]
   \[ f_n = f_{n-1} + f_{n-2} \quad \text{for } n \geq 2 \]
Complete the recursive algorithm that calculates the \( n \)th element in the sequence.

   ```c
   int fib ( int n ) {
       if (            ) {
           return
       } else if (            ) {
           return
       } else {
           return
       } // end if
   } // end fib
   ```

2. Complete the recursion tree showing the calls for \( \text{fib}(5) \).

   ![Recursion Tree]

3. For the \( \text{fib}(5) \) recursion tree, what would be the maximum number of \( \text{fib} \) "call-frames" on the "run-time stack" during execution.

4. In the \( \text{fib}(5) \) recursion tree, number the order in which the calls are performed.
5. Let $T(n)$ represent the time to perform the call to $\text{fib}(5)$. Complete the recurrence relation for the fibonacci.

$$T(n) =$$

6. Let's investigate how fast does $T(n)$ grows with respect to $n$'s growth?

Complete the inequalities:

(a) \hspace{1cm} T(n) \leq 2 \times T(n-2)

(b) \hspace{1cm} T(n) \leq 2 \times T(n-1)

7. What makes the recursive Fibonacci calculation so slow?
8. Can you think of a nonrecursive algorithm that would be faster?

9. Let $T(n)$ represent the time to perform the call to your faster algorithm with input $n$. How fast does $T(n)$ grows with respect to $n$'s growth?