Introduction
Instead of closely following our text in chapter 3, we take a different approach based on the industrial experience of your instructor. First, we narrow the application of algorithms to algorithms that can be implemented as computer programs. Second, we amplify our methods of demonstrating that statements about an algorithm are true to include testing on an actual computer.

**Definition 1**
An **algorithm** is a finite set of precise instructions for performing a computation or for solving a problem. Algorithms are expressed as computer programs. We will use the C++ programming language is this class to express algorithms.

**Definition 2**
Algorithms are measured in terms of **time complexity**. The time complexity of an algorithm is a measure of how much time is required to execute an algorithm. We measure the time in terms of the number of operations performed to execute an algorithm. A function $T(n)$ is created for every algorithm. The function $T(n)$ is a measure of the time required to execute an algorithm. The parameter $n$ is the size of the problem.

**Definition 3**
The time complexity of the statements of a computer program are given by a set of heuristics or rules.

**Time Complexity Rule 1**
The cost of subprogram prologs and epilogs is said to be zero. The cost of calling and returning from a subprogram is zero.

**Example 1**
Find the time complexity of the program in Figure 1.

```c++
int main()
{
    return 0;
}
```

Figure 1. Program for Example 1.

Solution: Complete the following steps.
1. Create a three column table putting line numbers in the first column, statements of the program in the second column, and the cost of each statement in the third column.
2. Sum the cost of the individual statements to find the cost of the program.

<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>int main()</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>{ return 0;</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>}</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

Explanation:
1. The statement of line 1 was assigned a cost of zero (0) because the function header is translated to the subprogram prolog and by Time Complexity Rule 1 subprogram prologs are said to cost nothing.
2. The statement on line 2 was assigned a cost of zero (0) because a function epilog is said to cost nothing and a return-statement is translated to a function epilog.
3. The part of the algorithm on line 3 is the closing curly brace. No cost is associated with either an opening curly brace or a closing curly brace.
The total is zero.

**Example 2**

Find the time complexity of the program in Figure 2.

```c
void DoNothing()
{
}

int main()
{
    DoNothing();
    return 0;
}
```

Figure 2. Program for Example 2.

**Solution:** Complete the following steps.

3. Create a three column table putting line numbers in the first column, statements of the program in the second column, and the cost of each statement in the third column.

4. Sum the cost of the individual statements to find the cost of the program.

<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>void DoNothing()</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>{</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>}</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>int main()</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>{    DoNothing();</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>return 0;</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>}</td>
<td>0</td>
</tr>
</tbody>
</table>

**Total**

**Explanation:**

1. The statement of line 1 was assigned a cost of zero (0) because a subprogram prolog is said to cost nothing by Time Complexity Rule 1.
2. The statement on line 2 was assigned a cost of zero (0) because the cost of a curly brace is zero.
3. The curly brace on line 3 marks the end of the function and is translated to the subprogram epilog. The subprogram epilog is assigned a cost of zero (0) by Time Complexity Rule 1.
4. The statement on line 4 is a subprogram prolog and therefore is assigned a cost of zero (0).
5. The statement on line 5 is translated to a subprogram epilog and is assigned a cost of zero (0).
6. The statement on line 6 is a closing curly brace and incurs no cost.

The total is zero.

**Time Complexity Rule 2**

Declarations cost nothing.
Example 3
Find the time complexity of the program in Figure 3.

```c
int main()
{
    int a;
    return 0;
}
```

Figure 3. Program for Example 3.

**Solution:** Complete the following steps.
1. Create a three column table putting line numbers in the first column, statements of the program in the second column, and the cost of each statement in the third column.
2. Sum the cost of the individual statements to find the cost of the program.

<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>int main()</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>{ int a;</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>return 0;</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

**Explanation:**
1. The statement of line 1 was assigned a cost of zero (0) because the function header is translated to the subprogram prolog and by Time Complexity Rule 1 subprogram prologs are said to cost nothing.
2. The statement on line 4 is a declaration and by Time Complexity Rule 2, the cost is zero (0).
3. The statement on line 3 was assigned a cost of zero (0) because a function epilog is said to cost nothing and a return-statement is translated to a function epilog.
4. The part of the algorithm on line 4 is the closing curly brace. No cost is associated with either an opening curly brace or a closing curly brace.

The total is zero.

**Time Complexity Rule 3**
Unit cost is assigned to all basic operations including arithmetic, logic, relational, and assignment operations.

Example 4
Find the time complexity of the program in Figure 4.

```c
int main()
{
    int a;
    a=1;
    a=a+1;
    return 0;
}
```

Figure 4. Program for Example 4.

**Solution:** Complete the following steps.
1. Create a three column table putting line numbers in the first column, statements of the program in the second column, and the cost of each statement in the third column.
2. Sum the cost of the individual statements to find the cost of the program.
Line | Statement | Cost
--- | --- | ---
1 | int main() | 0
2 | { int a; | 0
3 | a=1; | 1
4 | a=a+1; | 2
5 | return 0; | 0
6 | } | 0
Total | 3

**Explanation:**
1. The statement of line 1 was assigned a cost of zero (0) because the function header is translated to the subprogram prolog and by Time Complexity Rule 1 subprogram prologs are said to cost nothing.
2. The statement on line 4 is a declaration and by Time Complexity Rule 2, the cost is zero (0).
3. The statement on line 3 was assigned a cost of 1 because of the assignment operation. Basic operations including assignment are assigned unit cost by Time Complexity Rule 3.
4. The statement on line 4 was assigned a cost of 2 because there are two basic operations on that line, an arithmetic operation, addition, +, and an assignment operation. Unit cost is assigned to both operations making the total for line 4 equal to two (2).
5. The statement on line 5 was assigned a cost of zero (0) because a function epilog is said to cost nothing and a return-statement is translated to a function epilog.
6. The part of the algorithm on line 6 is the closing curly brace. No cost is associated with either an opening curly brace or a closing curly brace.
The total now comes to 3 adding the non-zero values on lines 3 and 4.

**Time Complexity Rule 4**
The cost of the operations executed in the test of a while-loop is accounted in two places. The cost of the test *when it is true* is accounted on the same line as the test. The cost of the test *when it is false* is accounted on the closing curly brace of the while-loop. The bodies of all while-loops are enclosed in curly braces.

**Example 5**
Find the time complexity of the code fragment in Figure 5.
```
int a=0;
while (a<10) {
    a++;
}
```
Figure 5. Code Fragment for Example 5.

**Solution:** Complete the following steps.
1. Create a three column table putting line numbers in the first column, statements of the program in the second column, and the cost of each statement in the third column.
2. Sum the cost of the individual statements to find the cost of the program.
### Line | Statement | Cost
--- | --- | ---
1 | int \( a=0; \) | 1
2 | while (\( a<10 \)) { | \( k = \sum_{a=0}^{9} 1 = 10 \) \( k = 10 \)
3 | \( a++; \) | 1
4 | } | 22

**Total**

**Explanation:**

1. The statement of line 1 is assigned a cost of one (1). The declaration costs nothing but the assignment operator has unit cost.

2. The operations on line 2 are assigned a cost of 10. The test on line 2 is executed eleven (11) times. The relational operator is eleven times. The first ten times the test is executed, the result is true. The last time it is executed, the eleventh time, it is false. We make the cost of line 2 equal to the cost of the test only when it is true. Since the result of the test is true ten times, we assign a cost of 10 to the statement, one for each time the relational operator was executed.

   We assign the cost of the test, when it is false, to the closing curly brace on the loop – to line 4.

3. The post-increment operator is assigned unit cost. Since it is executed as many times as the while-loop test is executed when it is true, we multiply the number of times the statement is executed by the cost of the statement, making the total ten (10).

4. We assign the cost of the test, when it is false, to the closing curly brace on the while-loop. There is no cost given to the curly brace ordinarily but we assign the cost of the test, when it is false, to the closing curly brace on a while-loop.

The total is 21.
Time Complexity

Rule 5

Computing the number of operations in a for-loop is too complicated. We, therefore, transform a for-loop into statements and a while-loop as shown in the example below.

Example 6

Split the for-loop in the figure below into its component parts and compute the time complexity of the code fragment.

\[
\text{for (int } a=0; a<10; a++)
\]

Figure 6. Code Fragment for Example 6.

Solution: Complete the following steps.

<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>int (a=0);</td>
<td>Place the loop initialization before the loop test.</td>
</tr>
<tr>
<td>2</td>
<td>while ((a&lt;10)) {</td>
<td>Replace the for-loop test with a while-loop test.</td>
</tr>
<tr>
<td>3</td>
<td>(a++);</td>
<td>Place the loop increment/decrement just before the end of the loop body.</td>
</tr>
<tr>
<td>4</td>
<td>}</td>
<td>Be sure to make the loop body a compound statement enclosed in curly braces.</td>
</tr>
</tbody>
</table>

Total

22

Since this is the same loop as shown in example 5, it has the same cost.
Time Complexity
Rule 6

Always select the alternative having the largest time complexity value.

Example 7

Find the time complexity of the code fragment in Figure 7.

```plaintext
if (a<n) {
    a=n*(a-1)+(n-1)*(a-2);
} else {
    a++;
}
```

Figure 7. Program for Example 7.

**Solution:** Complete the following steps.

<table>
<thead>
<tr>
<th>Line</th>
<th>Statement</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (a&lt;n) {</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>a=n*(a-1)+(n-1)*(a-2)</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>} else {</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>a++;</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>}</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>8</td>
</tr>
</tbody>
</table>

**Explanation:**

1. The statement on line 1 has unit cost because of the single comparison operator. The comparison is executed regardless of which alternative is selected.
2. The operations on line 2 are assigned a cost of 7 because of the 7 operations. Of the two alternatives, we always select the more costly. We desire the most conservative and costly computation of time complexity. Thus, any individual execution of this algorithm will never exceed our expression for its time complexity.
3. Line 3 has no operation on it and we assign no cost to it.
4. We have a single operation on line 4 and assign the cost accordingly.
5. There is no cost associated with the closing curly brace.

The total is 8, one for the comparison on line 1 and 7 for the maximum of the two alternatives.