Time Complexity of Loops

We analyze a number of fragments of C++ code. In each fragment I have inserted a variable `kount` which is not really part of the loop, but a counter which can be used to estimate its time complexity.

1. Here is a simple example. The code outputs n, and the asymptotic time complexity is $\Theta(n)$.

```
int kount = 0;
for(int = 0; i < n; i+++) kount++;
cout << kount << endl;
```

2. Here is an example with nested loops:

```
int kount = 0;
for(int i = 0; i < n; i++)
 for(int j = 0; j < n; j++) kount++;
cout << kount << endl;
```

By the result of Example 1, substituting i for n, the inner loop itertes i times for each iteration of the outer loop. Thus, the time complexity of Example 2 is $\Theta(n^2)$.

3. Here's a slightly trickier example:

```
int kount = 0;
for(int i = 0; i*i < n; i++) kount++;
cout << kount << endl;
```

Since $i * i < n$ is equivalent to $i < \sqrt{n}$, we can rewrite the code:

```
for(int i = 0; i < sqrt(n); i++) kount++;
```

We see that the number of iterations is $\Theta(\sqrt{n})$.

4. Here's an example that involves logarithms.

```
int kount = 0;
for(int i = 1; i < n; i=2*i) kount++;
cout << kount << endl;
```

What is the value of i after t iterations of the loop have executed? If $t = 1$, $i = 2$, If $t = 2$, $i = 4$, if $t = 3$, $i = 8$, and if $t = 4$, $i = 16$. We realize that $i = 2^t$, which means that $t = \log_2 i$. Thus, $i < n$, $t < \log_2 n$. The number of iterations is approximately $\log_2 n$, which is $\Theta(\log n)$.

And if we repeatedly halve the index until we get to 1, the number of iterations is also logarithmic.

5.
```
int kount = 0;
for(int i = n; i > 1; i = i/2) kount++;
cout << kount << endl;
```

The time complexity of Example 5 is $\Theta(\log n)$.

Substitution

You've used substitution to evaluate integrals in calculus. The technique can also be applied to asymptotic analysis. We solve Example 4 by using the substitution $k = \log i$. Note that $\log(2i) = \log i + \log 2 = \log i + 1 = k + 1$. The condition $i < n$ becomes $\log i < \log n$, or $k < \log n$.

Making the substituion, Example 4 becomes

6. ```
   int kount = 0;
   for(int k = 0; k < log n; k++); kount++;
   cout << kount << endl;
   ```

and we have another proof that the time complexity is $\Theta(\log n)$. The same substitution can be used to prove that Example 5 has time complexity $\Theta(\log n)$.

Here is a more interesting example, which you can solve using substitution twice.

7. ```
   int kount = 0;
   for(int i = 2; i < n; i = i*i); kount++;
   cout << kount << endl;
   ```

Let $i = 2^k$, or $k = \log i$, and $m = \log n$. Our code becomes

8. ```
   int kount = 0;
   for(int k = 1; k < m; k = 2*k) kount++;
   cout << kount << endl;
   ```

   Now substitute $k = 2^\ell$, or $\ell = \log k$. We get

9. ```
   int kount = 0;
   for int ell = 0; ell < log m; ell++) kount++;
   cout << kount << endl;
   ```

The time complexity is thus $\Theta(\log m) = \Theta(\log \log n)$.

Nested Loops

Computing the time complexity of nested loops is easy if the loops are independent:

10. ```
    int kount = 0;
    for(int i = 0; i < n; i++)
     for(int j = 0; j < log n; j++) kount++;
    cout << kount << endl;
    ```

The complexity is $\Theta(n \log n)$.
    ```

Interactive Nested Loops

The problem is harder if the bounds in the inner loop depend on the index of the outer loop. In the examples below, one loop is linear and the other is logarithmic, implying that the time complexity is $O(n \log n)$. However, it may not be $\Theta(n \log n)$, rather $\Theta(n)$. We need to distinguish those two cases.

11.
```
int kount = 0;
for(int i = 1; i < n; i++)
 for(int j = 1; j < i; j=2*j) kount++;
cout << kount << endl;
```

12.
```
int kount = 0;
for(int i = 1; i < n; i++)
 for(int j = i; j > 0; j=j/2) kount++;
cout << kount << endl;
```

In examples 11 and 12, we use the results for Examples 4 and 5 to show that the inner loop executes in $\Theta(\log i)$ time. Thus, both code fragments are equivalent to

```
int kount = 0;
 for(int i = 1; i < n; i++)
  kount = kount + log(i);
cout << kount << endl;
```

Since $\sum_{i=1}^{n} \log i = \Theta(n \log n)$, each example has time complexity $\Theta(n \log n)$.

13.
```
int kount = 0;
 for(int i = 1; i < n; i++)
  for(int j = i; j < n; j = 2*j) kount++;
 cout << kount << endl;
```

Note the difference between examples 11 and 13. Let $k = j/i$. Then $k = 1$ when $j = i$, $k = n/i$ when $j = n$, and $2k = 2j/i$. Substituting, we have

```
int kount = 0;
for(int i = 1; i < n; i++)
 for(int k = 1; k < n/i; k = 2*k) kount++;
cout << kount << endl;
```

Since $\log(n/i) = \log n - \log i$, we can rewrite the code

```
int kount = 0;
for(int i = 1; i < n; i++)
 kount = kount + log(n) - log(i);
cout << kount << endl;
```

We now use an integeral to approximate the sum.

$$\sum_{i=1}^{n}(\log n - \log i) \approx \int_{x=1}^{n}(\ln n - \ln x)dx = (x\ln n - x\ln x + x)\Big|_{x=1}^{x=n} = n\ln n - n\ln n + n - \ln n - 1 = \Theta(n)$$

The time complexity of the loop is thus $\Theta(n)$.

14.
```
int kount = 0;
for(int i = 1; i <= n; i=2*i)
 for(int j = 1; j <= i; j++)
cout << kount << endl;
```

Let $k = \log i$, that is $i = 2^k$.

```
int kount = 0;
for(int k = 0; k <= log n; k++)
 for(int j = 1; j <= 2^k; j++)
cout << kount << endl;
```

Both loops are now linear. The inner loop iterates 2^k times. We can rewrite the code as

```
int kount = 0;
for(int k = 0; k <= log n; k++)
 kount = kount + 2^k;
cout << kount << endl;
```

The problem is now summation of a geometric series:

$$\sum_{k=0}^{\log n} 2^k = 2^{\log n+1} - 1 = 2n - 1 = \Theta(n)$$

15.
```
int kount = 0;
for(int i = 1; i <= n; i=2*i)
 for(int j = i; j <= n; j++)
cout << kount << endl;
```

We use the same substitution: $k = \log i$. The inner loop iterates $n = 2^k$ times, and the time complexity is approximately

$$\sum_{k=0}^{\log n}(n - 2^k) = n\log n - (2^{\log n+1} - 1) = n\log n - 2n + 1 = \Theta(n\log n)$$