Analyzing Complexity of Loops and Nested Loops

We first analyze code fragments consisting of a single loop. We assume that everyone understands the time complexity of the basic linear loop given below to be $\Theta(n)$.

```
for(int i = 0; i < n; i = i+1)
```

A C++ for loop has four parts: the initializer, which initializes the iterator, the condition, the incrementer, and the body. In this document, we assume the body, which we do not write, takes one time unit to execute. The initializer and incrementer are really assignments, not equalities. In C++, "=" is used for assignment, while "==" means equality. But "=" usually denotes equality. To avoid confustion, we will use the equal sign for equality, and the stardard assignment symbol[1] "$\leftarrow$" for assignment. Since we are not actually writing a program, we need not mention the type of the iterator. Using our modified notation, the basic linear loop now:

for(i $\leftarrow$ 0; i < n; i $\leftarrow$ i+1)

We now use substitution to find the complexity of a number of single loop examples.

1. for(i $\leftarrow$ n; i > 0; i $\leftarrow$ i−1)

   Of course, we know that the answer is $\Theta(n)$. But as a warm-up, we use the substitution method anyway.

   We substitute j = n−1, which implies i = n−j. Thus the assingment i $\leftarrow$ n means j $\leftarrow$ n-i = n-n = 0. The condition $i > 0$ translates to n−j > 0, or $j < n$. The incrementer sets j = n−i to n−(i−1) = n−i+1 = j+1. We now write the loop:

   for(j $\leftarrow$ 0; j < n; j $\leftarrow$ j+1)

   which is the basic linear loop, with complexity $\Theta(n)$.

2. We work one more simple, but important, example.

   for(i $\leftarrow$ m; i < n; i $\leftarrow$ i+1) where m and n are given.

   We the loop iterates n−m times, but we'll use substitution anyway. Let j = i−m. For the initializer, subtract m from both sides, giving us j = i−m $\leftarrow$ m−m = 0. The condition becomes i−m < n−m, that is, j < n−m, and the incrementer i $\leftarrow$ i+1 becomes i−m $\leftarrow$ i+1−m, that is, j $\leftarrow$ j+1. We obtain:

   for(j = 0; j < n−m; j $\leftarrow$ j+1)

   Once again, this is the basic linear loop, and the complexity is Θ(n−m).

3. We now tackle some more complex examples. Consider the loop

   for(i $\leftarrow$ 1; i < n; i $\leftarrow$ 2∗i)

   This loop is logarithmic, as we shall see. Substitute j = log i and m = log n. Taking the log of both sides, the condition becomes log i < log n, and the incrementer becomes log i $\leftarrow$ log(2i) = log i + 1, that is, j $\leftarrow$ j+1. The loop is now

   for (j $\leftarrow$ 0; j < m; j $\leftarrow$ j+1)

   Which is, once again, our basic linear loop, whose complexity is Θ(m) = Θ(log n).

[1] Many programming languages use "=" for assignment, but that is not the normal way to express assignment in text.

4. for(i ← 0; i∗i < n; i ← i+1)

   Substitute m = $\sqrt{n}$, or n = m^2. The condition i∗i < n becomes i∗i < m∗m, equivalent to i < m. The loop is now

   for(i ← 0; i < m; i ← i+1)

   Which is, once again, our basic linear loop. Its complexity is $\Theta(m) = \Theta(\sqrt{n})$.

5. Now, for a somewhat trickier example.

   for(i ← 2; i < n; i ← i∗i)

   We initialize the iterator at 2, since if we used 0 or 1, the loop would be infinite. We subsitute j = log i and m = log n. We take the log of both sides for all three parameters of the loop. The initializer becomes j ← log 2 = 1. The condition becomes log i < log n), equivalent to $j < m$. The incrementer assigns log(i∗i) = 2log i to log i, which means that j is assigned the value 2j. We have:

   for(j ← 1; j < m; j ← 2∗j)

   This is equivalent to the loop for problem 3, and thus has complexity $\Theta(\log m) = \Theta(\log \log n)$.

We now consider nested loops. Here are two examples.

```
code(a): for( i ← 1; i < n; i ← i+1)
             for( j ← 1; j < i; j ← 2*j)

code(b): for( i ← 1; i < n; i ← i+1)
             for( j ← i; j < n; j ← 2∗j)
```

The complexity of code(a) is $\Theta(n \log n)$, while the complexity of code(b) is $\Theta(n)$. We use two different methods for calculating those complexities.

The first method is by reduction to integration. For each $0 < i < n$, the number of iterations of the inner loop of code(a) during the i^{th} iteration of the outer loop is $\Theta(\log i)$. Complexity of code(a) is thus $\Theta\left(\sum_{i=1}^{n-1} \log i\right)$, approximated by $\int_{x=1}^{n} \ln x \, dx = [x \ln x - x + 1]_1^n = n \ln n - n = \Theta(n \log n)$.

For each $0 < i < n$, the number of iterations of the inner loop of code(b) during the i^{th} iteration of the outer loop is $\Theta(\log n - \log i)$. The complexity of the code is thus $\Theta\left(\sum_{i=1}^{n-1} \log n - \log i\right)$, approximated by $\int_{x=1}^{n} (\ln n - \ln x) \, dx = [x \ln n - x \ln x + x - 1]_1^n = \Theta(n)$.

We now analyze the code fragments by a different method, which does not use calculus. Note that the time complexity of each code fragments is simultaneously $O(n \log n)$ and $\Omega(n)$. We show that the complexity of code(a) is $\Omega(n \log n)$, and the complexity of code(b) is $O(n)$, proving that the complexities are $\Theta(n \log n)$ and $\Theta(n)$, respectively.

We measure time complexity by counting the iterations of the inner loop. At each such iteration, there is an *index pair* (i,j), consisting of the current indices of the outer and inner loops. We could display the index pair by inserting the line

```
cout << "(" << i << "," << j << ")" << endl;
```

into each code fragment. Figures 1 and 2 show those pairs in the case n = 40. In each figure, the values of i are written above the matrix, and the values of j during the i^{th} iteration of the outer loop are written under i. The index pairs of code(a) consist of all (i,j) such that 0 < i < n and j is a power of 2 which is less than i.

code(c): for(i $\leftarrow \lceil n/2 \rceil$; i < n; i $\leftarrow$ i + 1)
for(j $\leftarrow$ 1; j < $\lceil n/2 \rceil$; j $\leftarrow$ 2$*$j)

code(d): for(i $\leftarrow \lceil n/2 \rceil$; i < n; i $\leftarrow$ i + 1)
for(k $\leftarrow$ 0; k+1 $\leq$ log(n$-$1); k $\leftarrow$ k+1)

Claim I: If k+1 $\leq$ log(n-1), then $2^k < \lceil n/2 \rceil$.
Proof of Claim I:
k+1 $\leq$ log(n$-$1), hence $2^{k+1} \leq n-1$, hence $2^k \leq \frac{n-1}{2} < \frac{n}{2} \leq \lceil \frac{n}{2} \rceil$

The time complexity of code(d) is simply the product of the numbers of times each loop iterates, namely $\log(n-1)(n-\lceil n/2 \rceil) = \Theta(n \log n)$, since the loops are independent. Letting $j = 2^k$, we see, by Claim I, that the complexity of code(c) is at least as great as the complexity of code(d). Since every index pair of code(c) is also an index pair of code(a), we can conclude that the complexity of code(a) is $\Omega(n \log n)$.

By Claim I, and substituting $j = 2^k$, we observe that the index pairs of code(c) form a subset of the index pairs of code(a), indicated by the red rectangle in Figure 1. Therefore the complexity of code(a) is $\Omega(n \log n)$. We conclude that code(a) has complexity $\Theta(n \log n)$.

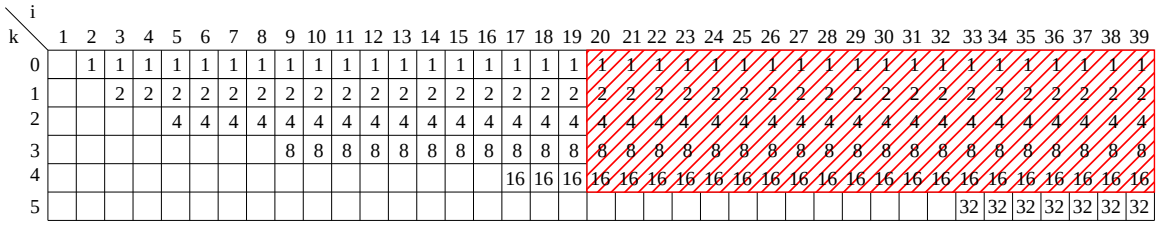

Figure 1: The Index pairs of code(c), indicated by the red rectangle, form a subset of the index pairs of code(a). Each cell contains the value of $j = 2^k$ for the index pair (i,j) of code(a).

We now prove that code(b) is $\Theta(n)$,

code(e): for(i $\leftarrow$ 1; i < n; i $\leftarrow$ i + 1)
for(k $\leftarrow$ 0; i$*2^k <$ n ; k $\leftarrow$ k+1)

code(f): for(k $\leftarrow$ 0; $2^k <$ n; k $\leftarrow$ k+1)
for(i $\leftarrow$ 1; i$*2^k <$ n; i $\leftarrow$ i + 1)

Claim II: Code(b) has the same complexity as code(e).
Proof of Claim:
Starting with code(e), substitute j for $i*2^k$. The initializer becomes $j \leftarrow i*2^0 = i$, the condition becomes j < n, and the incrementer becomes $j = i*2^k \leftarrow i*2^{k+1} = 2*j$, which yields code(b).

Claim III: Code(e) has the same complexity as code(f).
Proof of Claim:
Each iterates with exactly the same values of i and k.

Finally, for each value of k, the number of iterations of the inner loop of code(f) does not exceed $n*2^{-k}$, hence the total time complexity of (e) is bounded by the convergent geometric series $\sum_{textrmk=0}^{\infty} n2^{-k} = 2n = O(n)$. We conclude that code(b) is $\Theta(n)$.

k \ i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
1	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38																				
2	4	8	12	16	20	24	28	32	36																														
3	8	16	24	32																																			
4	16	32																																					
5	32																																						

Figure 2: Each cell contains the value of $j = i*2^k$ for the index pair (i,j) of code(b).

Try to find the complexity of these nested loop code fragments. When I say "easy," I mean no harder than the problems above.

6. "easy"

   for(i $\leftarrow$ n; i $>$ 0; i $\leftarrow$ i/2)

   for(j $\leftarrow$ 1; j $<$ i; j $\leftarrow$ j+1)

7. "easy"

   for(i $\leftarrow$ n; i $>$ 0; i $\leftarrow$ i/2)

   for(j $\leftarrow$ n; j $>$ i; j $\leftarrow$ j$-$1)

8. "easy"

   for(i $\leftarrow$ 1; i $<$ n; i $\leftarrow$ i+1)

   for(j $\leftarrow$ i; j $>$ 0; j $\leftarrow$ j/2)

9. "medium"

   for(i $\leftarrow$ 1; i$*$i $<$ n; i $\leftarrow$ i+1)

   for(j $\leftarrow$ i; j $>$ 0; j $\leftarrow$ j/2)

10. "medium"

    for(i $\leftarrow$ 1; i $<$ n; i $\leftarrow$ 2$*$i)

    for(j $\leftarrow$ 2; j $<$ i; j $\leftarrow$ j$*$j)

11. "medium"

    for(i $\leftarrow$ 1; i $<$ n; i $\leftarrow$ i+1)

    for(j $\leftarrow$ 1; j $<$ i$*$i; j $\leftarrow$ 2$*$j)

12. "hard"

    for(i $\leftarrow$ 1; i $<$ n; i $\leftarrow$ 2$*$i)

    for(j $\leftarrow$ 1; i $<$ i; j $\leftarrow$ 2$*$j)

13. "even harder"

    for(i $\leftarrow$ 1; i $<$ n; i $\leftarrow$ 2$*$i)

    for(j $\leftarrow$ n; j $>$ i; j $\leftarrow$)/2