Code Examples

For Loops

Consider the following pseudo-code:
Function $f_1(n)$:
x = 0
for i=1 to n do
    for j=1 to n do
        x = x + (i-j)
end

First, we need to agree upon a "model of computation" in which to do our analysis.

When we talk about the running time of this code, we basically mean "what is the total number of operations that will need to be performed to run this program in our model of computation?".

This model informs us on how to approximate how many operations each instruction takes, and in particular which instructions take constant time and which don't. We typically do not formally define the model of computation, it is instead a loosely agreed upon thing.

In any reasonable model, doing an assignment like "x=0" or "x=1" would be viewed as taking a fixed number of operations.

But how many operations does it take to execute a statement like
"x = x + (i-j)"?

That depends on the sizes of x, i, j, in particular it depends on the number of digits.

So a mathematician would NOT consider this to happen with a constant amount of operations, but a computer scientist could; our program are typically written using primitive datatypes like int, float, double that have a fixed

number of digits.

To exactly capture the running time of f_1 in our model of computation, we would need to include constants for the different amount of time each line takes when running.

So the exact running time E_n would look something like this

$$E_n = c' + \sum_{a=1}^{n} \left(c'' + \sum_{b=1}^{n} (c''' + c'''') \right)$$

where c', c'', c''', c'''' are non-zero constants that capture the time to execute "x=0", maintain the for loops, and execute "x = x + (i-j)".

Function $f_1(n)$:
x = 0
for i=1 to n do
    for j=1 to n do
        x = x + (i-j)
end

We will instead express the running time (T_n) of this code as

$$T_n = \sum_{a=1}^{n} \sum_{b=1}^{n} c,$$

where $c > 0$ is some constant.

Of course, this does not exactly capture the running time of this code. However, we can notice there are some constants $c_1, c_2 > 0$ such that

$$c_1 T_n \leq E_n \leq c_2 T_n,$$

and therefore since we are doing an asymptotic analysis, analyzing T_n is sufficient for our purposes.
This is one of the first "shortcuts" asymptotic analysis allows us to take.

Therefore, we have

$$T_n = \sum_{a=1}^{n}\sum_{b=1}^{n} c = c\sum_{a=1}^{n}\sum_{b=1}^{n} 1 = c\sum_{a=1}^{n} n = cn\sum_{a=1}^{n} 1 = cn^2$$

and therefore $T_n = \Theta(n^2)$.

You'll notice here that the value of c did not matter; in the next examples I will just use 1 instead of c.

This is another "shortcut" available to us. But be careful with this shortcut, where the constant appears determines if it matters or if it doesn't; until you get some experience and learn to spot the difference it is best to include the constants.

Next, consider the following code:
Function $f_2(n)$:
x = 0
for i=1 to n do
    for j=1 to i do
        x = x + (i-j)
end
We can express the running time (T_n) of this code as

$$T_n = \sum_{a=1}^{n}\left(\sum_{b=1}^{a} 1\right) = \sum_{a=1}^{n} a = \frac{n(n+1)}{2} = 0.5n^2 + 0.5n$$

and therefore $T_n = \Theta(n^2)$.

$$\sum_{a=1}^{n^2} a^2$$

Lets try something a little more complicated:
Function $f_3(n)$:
x = 0
for i=1 to n do
    for j=1 to $\sqrt{n}$ do
        x = x + (i-j)
end
We can express the running time (T_n) of this code as

$$T_n = \sum_{a=1}^{n} \sum_{b=1}^{\sqrt{n}} 1 = \sum_{a=1}^{n} \sqrt{n} = \sqrt{n} \sum_{a=1}^{n} 1 = n\sqrt{n} = n^{1.5}.$$

and therefore $T_n = \Theta(n^{1.5})$.

Lets try something a little more accurate:
Function $f_4(n)$:
x = 0
for i=1 to n do
    for j=1 to $\lfloor\sqrt{n}\rfloor$ do
        x = x + (i-j)
end

Note that here the $\lfloor\sqrt{n}\rfloor$ indicates we are rounding down the square root of n (to round up, we would use the notation $\lceil\sqrt{n}\rceil$). We can express the running time (T_n) of this code as

$$T_n = \sum_{a=1}^{n} \sum_{b=1}^{\lfloor\sqrt{n}\rfloor} 1 = \sum_{a=1}^{n} \lfloor\sqrt{n}\rfloor = n\lfloor\sqrt{n}\rfloor.$$

This is not a nice function, so we will want to simplify it further.
Observe that

$$n\lfloor\sqrt{n}\rfloor \leq n\sqrt{n}$$

and so $n\lfloor\sqrt{n}\rfloor = O(n^{1.5})$.
We can also observe that

$$n\lfloor\sqrt{n}\rfloor \geq n(\sqrt{n}-1) = n\sqrt{n} - n = \frac{1}{2}n\sqrt{n} + \frac{1}{2}n\sqrt{n} - n \geq \frac{1}{2}n\sqrt{n}$$

for $n \geq 4$.
This then tells us that $n\lfloor\sqrt{n}\rfloor = \Omega(n^{1.5})$.
Therefore $T_n = \Theta(n^{1.5})$.

Next examples:
Function $f_5(n)$:
x = 0
for i=1 to n^2 do
    for j=1 to i^3 do
        x = x + j!!!!
end

$$T_n = \sum_{a=1}^{n^2} \sum_{b=1}^{a^3} 1$$

How long does it take to compute j!!!!?
Not really a fair question at this point. Lets just skip this one.

Next example:
Function $f_6(n)$:
x = 0
for i=2n to $2n^2 + 5n$ do
    for j=1 to $i^3 + i^2$ do
        x = x + j
end
Our running time can be expressed as

$$T_n = \sum_{a=2n}^{2n^2+5n} \left(\sum_{b=1}^{a^3+a^2} 1 \right) = \sum_{a=2n}^{2n^2+5n} (a^3 + a^2)$$

Give up on finding equality, but don't give up on finding upper and lower bounds.
In this next part I will be using Nicomachus's Theorem, which tell us that

$$\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i \right)^2$$

Upper bound:

$$
\begin{aligned}
\sum_{a=2n}^{2n^2+5n}(a^3+a^2) &\le \sum_{a=1}^{7n^2}(a^3+a^2) \\
&\le \sum_{a=1}^{7n^2} 2a^3 \\
&= 2\sum_{a=1}^{7n^2} a^3 \\
&= 2\left(\sum_{a=1}^{7n^2} a\right)^2 && \text{applying Nichomachus's Theorem} \\
&= 2\left(\frac{7n^2(7n^2+1)}{2}\right)^2 \\
&= \text{polynomial with degree 8} \\
&\le cn^8 && \text{for some constant } c>0
\end{aligned}
$$

Therefore, $T_n = O(n^8)$.
Lower bound:

$$\begin{aligned}
\sum_{a=2n}^{2n^2+5n} (a^3+a^2) &\geq \sum_{a=2n}^{2n^2+5n} a^3 \\
&\geq \sum_{a=1}^{2n^2+3n} a^3 \\
&\geq \sum_{a=1}^{n^2} a^3 \\
&= \left(\sum_{a=1}^{n^2} a\right)^2 && \text{Nichomachus's again} \\
&= \left(\frac{n^2(n^2+1)}{2}\right)^2 \\
&= \text{polynomial with degree 8}
\end{aligned}$$

Therefore, $T_n = \Omega(n^8)$.
Since $T_n = O(n^8)$ and $T_n = \Omega(n^8)$, we can conclude that $T_n = \Theta(n^8)$.

While Loops

Consider the following code:
Function $w_1(n)$:
$x = 0$
$i = 7$
while $i \leq n$ do
    $x = (x+1)^2$
    $i = i + 1$
end
This while loop behaves very much like a for loop, in that it runs until the counter i reaches n.

We can express the running time (T_n) of this code as

$$T_n = \sum_{a=7}^{n} 1 = n - 7 + 1,$$

and since $\lim_{n\to\infty} \frac{n-6}{n} = 1$ we have $T_n = \Theta(n)$.

Next, consider the following code:
Function $w_2(n)$:
$x = 0$
$i = 1$
while $i < n$ do
    $x = (x + 1)^2$
    $i = i + 3$
end
This while loop also behaves very much like a for loop, in that it runs until the counter i reaches n, but this time the counter increases by 3 each time. We can express the running time (T_n) of this code as

$$T_n = \sum_{a=1}^{\lfloor n/3 \rfloor} 1 = \lfloor n/3 \rfloor,$$

and therefore $T_n = \Theta(n)$.

The next example is much more interesting:
Function $w_3(n)$:
$x = 0$
$i = 1$
while $i < n$ do
    $x = (x + 1)^2$
    $i = 2i$
end
To find the running time, we need to understand how many times the while loops runs.
This while loop runs as long as $i < n$, but i is doubling with each iteration.

Therefore, if k is the number of iterations of the while loops, the while loop runs while we have

$$i = 1 \cdot 2^k < n.$$

We can solve for k by taking the $\log_2$ of each side of the inequality:

$$\log_2(2^k) < \log_2(n).$$

and so $k < \log_2(n)$.
We can therefore express the running time (T_n) of this code as

$$T_n = \sum_{a=1}^{\lfloor \log_2(n) \rfloor} 1 = \lfloor \log_2(n) \rfloor,$$

and therefore $T_n = \Theta(\log(n))$.

Combinations of for loops and while loops

Function $fw_1(n)$:
$x = 0$
for $i = 1$ to n do
    $j = 1$
    while $j < n$ do
        $x = (x + 1)^2$
        $j = 2j$
end

First, lets figure out the number of iterations the while loop will do.

After k iterations, we will have $j = 2^k$.
We can keep iterating as long as $j = 2^k < n$.
Solving for k, we have that there will be $\lfloor \log_2(n) \rfloor$ iterations.

Therefore,

$$T_n = \sum_{a=1}^{n} \sum_{b=1}^{\lfloor \log_2(n) \rfloor} 1 = \sum_{a=1}^{n} \lfloor \log_2(n) \rfloor = n \lfloor \log_2(n) \rfloor$$

and so $T_n = \Theta(n \log(n))$.

Making a small modification, lets analyze this:
Function $fw_2(n)$:
$x = 0$
for $i = 1$ to n do
    $j = 1$
    while $j < i$ do
        $x = (x + 1)^2$
        $j = 2j$
end

After k iterations, we will have $j = 2^k$.
We can keep iterating as long as $j = 2^k < i$.
Solving for k, we have that there will be $\lfloor \log_2(i) \rfloor$ iterations.

Then

$$T_n = \sum_{a=1}^{n} \sum_{b=1}^{\lfloor \log_2(a) \rfloor} 1 = \sum_{a=1}^{n} \lfloor \log_2(a) \rfloor = \lfloor \log_2(1) \rfloor + \lfloor \log_2(2) \rfloor + \ldots \lfloor \log_2(n) \rfloor$$

This is a bit tough, lets give up on equality and find an upper bound:

$$\lfloor \log_2(1) \rfloor + \lfloor \log_2(2) \rfloor + \ldots \lfloor \log_2(n) \rfloor \leq \log_2(1) + \log_2(2) + \ldots \log_2(n) = \log_2(n!)$$

$$\log_2(A) + \log_2(B) = \log_2(AB)$$

Observe that $n! \leq n^n$, so

$$\log_2(n!) \leq \log_2(n^n) = n \log_2(n)$$

which then tells us that $T_n = O(n \log(n))$.

Can we show that $T_n = \Omega(n \log(n))$?

I will leave this as an exercise for you.