Summations, Running Time, Asymptotic Notation

Recitation Week 1

What is the value returned by the following function? Express your answer as a function of n. From this, deduce the worst-case running time using Big Oh notation. For full credit, provide a lower bound as well, thus leading to a bound with Θ notation.

function mystery(n)
$r := 0$
for $i := 1$ ***to*** $n - 1$ **do**
    for $j := i + 1$ ***to*** n **do**
        for $k := 1$ ***to*** j **do**
            $r := r + 1$
return r

```
function mystery(n)
r := 0
for i := 1 to n − 1 do
    for j := i + 1 to n do
        for k := 1 to j do
            r := r + 1
return r
```

The upper bound

Outer loop runs fewer than *n* times

Second loop runs fewer than *n* times

Inner loop runs fewer than *n* times

Therefore, the body of the loop runs at most $O(n^3)$ times and the running time is therefore $O(n^3)$.

```
function mystery(n)
r := 0
for i := n/4 to 3n/4 do
    for j := i + 1 to n do
        for k := 1 to j do
            r := r + 1
return r
```

The lower bound

Let's think what would happen if the outer loop only ran from i = n/4 to 3n/4

Outer loop runs at least *n/2* times

Second loop runs at least *n/4* times

Inner loop runs at least *n/4* times

Therefore, the body of our smaller program runs at least $n^3/32 = \Omega(n^3)$ times. So the running time of our original function is $\Omega(n^3)$.

FINDING THE RETURN VALUE

- Based on the code, finding the return value would also imply the running time
- We represent the return value as a summation

$$r = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{j} 1$$

```
function mystery(n)
r := 0
for i := 1 to n − 1 do
    for j := i + 1 to n do
        for k := 1 to j do
            r := r + 1
return r
```

SOLVING THE SUMMATION

$$
\begin{aligned}
r &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{j} 1 \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} j \\
&= \sum_{i=1}^{n-1} (n + (i+1)) \cdot \frac{n - (i+1) + 1}{2} \\
&= \frac{1}{2} \sum_{i=1}^{n-1} (n-i)(n+i+1) \\
&= \frac{1}{2} \sum_{i=1}^{n-1} n^2 + in + n - in - i^2 - i \\
&= \frac{1}{2}(n^2+n)(n-1) - \frac{1}{2} \sum_{i=1}^{n-1} i - \frac{1}{2} \sum_{i=1}^{n-1} i^2
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{(n^2+n)(n-1)}{2} - \frac{n(n-1)}{2 \cdot 2} - \frac{1}{2} \sum_{i=1}^{n-1} i^2 \\
&= \frac{(2n^2+n)(n-1)}{4} - \frac{(n-1)(n)(2n-1)}{2 \cdot 6} \\
&= \frac{(6n^2+3n-2n^2+n)(n-1)}{12} \\
&= \frac{(4n^2+4n)(n-1)}{12} \\
&= \frac{4n^3+4n^2-4n^2-4n}{12} \\
&= \frac{n^3-n}{3}
\end{aligned}
$$

What is the value returned by the following function? Express your answer as a function of n. From this, deduce the worst-case running time using Big Oh notation. For full credit, provide a lower bound as well, thus leading to a bound with Θ notation.

```
function mystery3(n)
r := 0
for i := 1 to n do
    for j := 1 to n do
        for k := j to n − j do
            r := r + 1
return r
```

```
function mystery3(n)
r := 0
for i := 1 to n do
    for j := 1 to n do
        for k := j to n − j do
            r := r + 1
return r
```

The upper bound

Outer loop runs fewer than *n* times

Second loop runs fewer than *n* times

Inner loop runs fewer than *n* times

Therefore, the body of the loop runs at most $O(n^3)$ times and the running time is therefore $O(n^3)$.

```
function mystery3(n)
r := 0
for i := 1 to n/4 do
    for j := 1 to n/4 do
        for k := j to n − j do
            r := r + 1
return r
```

The lower bound

Let’s think what would happen if the first loop ran from i = 1 to n/4 and the second loop only ran from j = 1 to n/4

Outer loop runs at least *n/4* times

Second loop runs at least *n/4* times

Inner loop runs at least *n/2* times

Therefore, the body of our smaller program runs at least $n^3/32 = \Omega(n^3)$ times. So the running time of our original function is $\Omega(n^3)$.

Finding the Return Value

- Based on the code, finding the return value would also imply the running time
- We represent the return value as a summation

$$r = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=j}^{n-j} 1$$

```
function mystery3(n)
r := 0
for i := 1 to n do
    for j := 1 to n do
        for k := j to n − j do
            r := r + 1
return r
```

SOLVING THE SUMMATION

$$
\begin{aligned}
r &= \sum_{i=1}^{n} \sum_{j=1}^{n/2} \sum_{k=j}^{n-j} 1 \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n/2} n - 2j + 1 \\
&= \sum_{i=1}^{n} n^2/2 - n/2 \cdot (n/2 + 1) + n/2 \\
&= \sum_{i=1}^{n} n^2/4 \\
&= \frac{n^3}{4}
\end{aligned}
$$

Is $\max(f(n), g(n)) = \Theta(f(n)+g(n))$ for all nonnegative functions $f(n)$ and $g(n)$ of n? Formally justify your answer.

- RECALL THAT F(N) = Θ(N) IS EQUIVALENT TO F(N) = Ω(G(N)) AND F(N) = O(G(N))
- DO WE THINK THE EQUALITY IS TRUE?
- WHICH IS THE "EASIER" OF THE TWO?

$$\max(f(n), g(n)) = O(f(n) + g(n))$$

Suppose $\max(f(n), g(n)) = g(n)$ (the other case will be symmetric). Then, since $g(n) \leq 1 \cdot (f(n)+g(n))$, $\max(f(n), g(n)) = O(f(n) + g(n))$.

$$\max(f(n), g(n)) = \Omega(f(n) + g(n))$$

Suppose $\max(f(n), g(n)) = g(n)$ (the other case will be symmetric). Then, $\frac{1}{2}(f(n)+g(n)) \leq \frac{1}{2}(2g(n)) \leq g(n)$. Finally, this means that $\max(f(n), g(n)) = \Omega(f(n) + g(n))$.

Is $\min(f(n), g(n)) = \Theta(f(n) + g(n))$ for all nonnegative functions $f(n)$ and $g(n)$ of n? Formally justify your answer.

- RECALL THAT F(N) = Θ(N) IS EQUIVALENT TO F(N) = Ω(G(N)) AND F(N) = O(G(N))
- DO WE THINK THE EQUALITY IS TRUE?
- PROOF BY CONTRADICTION

We will show that $\min(f(n), g(n)) = \Omega(f(n) + g(n))$ does not hold by contradiction. Suppose $\min(f(n), g(n)) = \Omega(f(n) + g(n))$ for all $f(n)$ and $g(n)$. Let $f(n) = n$ and $g(n) = 1$. Then we have that $\min(f(n), g(n)) = g(n) = 1$. By the definition of Ω, there must exist some constants c and n_0 such that $n' + 1 \leq c \cdot 1$ for all $n' > n_0$. Take $n_0 = c$. But then we have $n' + 1 > c + 1 > c$, a contradiction.

Show that $\Theta(\cdot)$ is transitive. That is, suppose $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$. Show that $f(n) = \Theta(h(n))$.

- Let's just begin with Big Oh
- Recall the definitions, all we need to do is find constants!

Suppose $f(n) = O(g(n))$ and $g(n) = O(h(n))$. Then, by definition, there exist c_1, c_2, n_0, n_1 such that $0 \leq f(n') \leq c_1 g(n')$ for all $n' > n_0$ and $0 \leq g(n'') \leq c_2 h(n'')$ for all $n'' > n_1$. We want to find some c_3 and n_2 such that $0 \leq f(n''') \leq c_3 h(n''')$ for all $n''' > n_2$.

Let $n_2 = \max(n_0, n_1)$. Since our inequalities hold for all sufficiently large n, $0 \leq f(n') \leq c_1 g(n')$ for all $n' > n_2$ and $0 \leq g(n'') \leq c_2 h(n'')$ for all $n'' > n_2$.

Let $c_3 = c_1 \cdot c_2$. We know that $0 \leq g(n'') \leq c_2 h(n'')$ for all $n'' > n_2$ so multiplying both sides by c_1 we have $0 \leq c_1 g(n'') \leq c_3 h(n'')$ for all $n'' > n_2$. We also know that $0 \leq f(n') \leq c_1 g(n')$ for all $n' > n_2$, so combining inequalities we have $0 \leq f(n') \leq c_3 h(n')$ for all $n' > n_2$, completing the proof.