CS 61B
Spring 2025

Asymptotics II

Exam-Level 05: February 24, 2025

1 Finish the Runtimes

Below we see the standard nested for loop, but with missing pieces!

```
for (int i = 1; i < ______; i = ______) {
    for (int j = 1; j < ______; j = ______) {
        System.out.println("Circle is the best TA");
    }
}
```

For each part below, **some** of the blanks will be filled in, and a desired runtime will be given. Fill in the remaining blanks to achieve the desired runtime! There may be more than one correct answer.

Hint: You may find **Math.pow** helpful.

(a) Desired runtime: $\Theta(N^2)$

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < i; j = ______) {
        System.out.println("This is one is low key hard");
    }
}
```

Solution:

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < i; j = j + 1) {
        System.out.println("This is one is low key hard");
    }
}
```

Explanation: Remember the arithmetic series $1 + 2 + 3 + 4 + ... + N = \Theta(N^2)$. We get this series by incrementing j by 1 per inner loop.

(b) Desired runtime: $\Theta(\log(N))$

```
for (int i = 1; i < N; i = i * 2) {
    for (int j = 1; j < ______; j = j * 2) {
        System.out.println("This is one is mid key hard");
    }
}
```

Solution: Any constant would work here, 2 was chosen arbitrarily.

```
for (int i = 1; i < N; i = i * 2) {
    for (int j = 1; j < 2; j = j * 2) {
        System.out.println("This is one is mid key hard");
    }
}
```

Explanation: The outer loop already runs $\log n$ times, since i doubles each time. This means the inner loop must do constant work (so any constant `j < k` would work).

(c) Desired runtime: $\Theta(2^N)$. $\frac{2^N}{N}$ is a valid answer, could you think of another?

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < ______; j = j + 1) {
        System.out.println("This is one is high key hard");
    }
}
```

Solution:

```
for (int i = 1; i < N; i = i + 1) {
    for (int j = 1; j < Math.pow(2, i); j = j + 1) {
        System.out.println("This is one is high key hard");
    }
}
```

Explanation: Remember the geometric series $1 + 2 + 4 + ... + 2^N = \Theta(2^N)$. We notice that i increments by 1 each time, so in order to achieve this 2^N runtime, we must run the inner loop 2^i times per outer loop iteration.

(d) Desired runtime: $\Theta(N^3)$

```
for (int i = 1; i < ______; i = i * 2) {
    for (int j = 1; j < N * N; j = ______) {
        System.out.println("yikes");
    }
}
```

```
for (int i = 1; i < Math.pow(2, N); i = i * 2) {
    for (int j = 1; j < N * N; j = j + 1) {
        System.out.println("yikes");
    }
}
```

Explanation: One way to get N^3 runtime is to have the outer loop run N times, and the inner loop run N^2 times per outer loop iteration. To make the outer loop run N times, we need stop after multiplying `i = i * 2` N times, giving us the condition `i < Math.pow(2, N)`. To make the inner loop run N^2 times, we can simply increment by 1 each time.

2 Asymptotics is Fun!

(a) Using the function `g` defined below, what is the runtime of the following function calls? Write each answer in terms of `N`. Feel free to draw out the recursion tree if it helps.

```
void g(int N, int x) {
    if (N == 0) {
        return;
    }
    for (int i = 1; i <= x; i++) {
        g(N - 1, i);
    }
}
```

`g(N, 1)`: Θ(____)

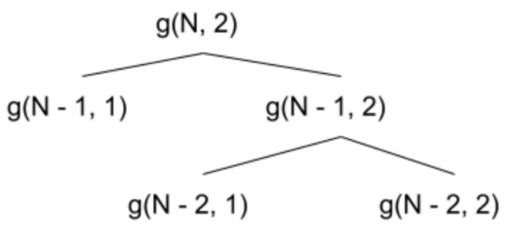
`g(N, 1)`: $\Theta(N)$

Explanation: When `x` is 1, the loop gets executed once and makes a single recursive call to `g(N - 1)`. The recursion goes `g(N)`, `g(N - 1)`, `g(N - 2)`, and so on. This is a total of `N` recursive calls, each doing constant work.

`g(N, 2)`: Θ(____)

`g(N, 2)`: $\Theta(N^2)$

Explanation: When `x` is 2, the loop gets executed twice. This means a call to `g(N)` makes 2 recursive calls to `g(N - 1, 1)` and `g(N - 1, 2)`. The recursion tree looks like this:



From the first part, we know `g(..., 1)` does linear work. Thus, this is a recursion tree with N levels, and the total work is $(N-1)+(N-2)+...+1=\Theta(N^2)$ work.

(b) Suppose we change line 6 to `g(N - 1, x)` and change the stopping condition in the for loop to `i <= f(x)` where `f` returns a random number between 1 and `x`, inclusive. For the following function calls, find the tightest Ω and big O bounds. Feel free to draw out the recursion tree if it helps.

```
void g(int N, int x) {
    if (N == 0) {
        return;
    }
    for (int i = 1; i <= f(x); i++) {
        g(N - 1, x);
    }
}
```

`g(N, 2)`: Ω`(____)`, O`(____)`

`g(N, N)`: Ω`(____)`, O`(____)`

`g(N, 2)`: $\Omega(N)$, $O(2^N)$

`g(N, N)`: $\Omega(N)$, $O(N^N)$

Explanation: Suppose `f(x)` always returns 1. Then, this is the same as case 1 from (a), resulting in a linear runtime.
On the other hand, suppose `f(x)` always returns x. Then `g(N, x)` makes `x` recursive calls to `g(N - 1, x)`, each of which makes `x` recursive calls to `g(N - 2, x)`, and so on, so the recursion tree has 1, x, x^2 ... nodes per level. Outside of the recursion, the function `g` does `x` work per node. Thus, the overall work is $x * 1 + x * x + x * x^2 + ... + x * x^{N-1} = x(1 + x + x^2 + ... + x^{N-1})$.
Plug in `x = 2` to get $2(1 + 2 + 2^2 + ... + 2^{N-1}) = O(2^N)$ for our first upper bound. Plug in `x = N` to get $N(1 + N + N^2 + ... + N^{N-1}) = O(N^N)$ (ignoring lower-order terms).

3 Asymptotics Proofs

As a reminder, the formal definitions of Ω, Θ, and O are provided below:

Let f, g be real-valued functions. Then:

$f(x) \in \Theta(g(x))$ if there exists $a, b, N_0 > 0$ such that for all $N > N_0$, $|ag(N)| \leq |f(N)| \leq |bg(N)|$.

$f(x) \in O(g(x))$ if there exists $b, N_0 > 0$ such that for all $N > N_0$, $|f(N)| \leq |bg(N)|$.

$f(x) \in \Omega(g(x))$ if there exists $a, N_0 > 0$ such that for all $N > N_0$, $|ag(N)| \leq |f(N)|$.

Informally, we say that $f(x) \in O(g(x))$ approximately means that $f(x) \leq g(x)$, and similarly, $f(x) \in \Theta(g(x))$ means $f(x) = g(x)$ and $f(x) \in \Omega(g(x))$ means $f(x) \geq g(x)$. This problem will explore why we can make this informal statement, by showing that the O relation shares many properties with the $\leq$ relation.

For this problem, let f, g, and h be real-valued functions, and let x, y, and z be real numbers. You won't be expected to write full proofs on exams, but this thinking style will be helpful on exams and especially in later classes.

(a) If $x \leq y$, then $y \geq x$. Show that if $f(x) \in O(g(x))$, then $g(x) \in \Omega(f(x))$

Solution: Assume x is large enough that the inequalities from all O bounds hold. From the definitions above we know that there exists $b > 0$ such that $|f(x)| \leq |bg(x)|$. Dividing both sides of the inequality by b, we get $|\frac{1}{b}f(x)| \leq |g(x)|$. If we define a to be $\frac{1}{b}$, this is the definition of $g(x) \in \Omega(f(x))$. Therefore if $f(x) \in O(g(x))$, $g(x) \in \Omega(f(x))$.

(b) If $x \leq y$ and $y \leq x$, then $x = y$. Show that if $f(x) \in O(g(x))$ and $g(x) \in O(f(x))$, then $f(x) \in \Theta(g(x))$

Solution: Assume x is large enough that the inequalities from all O bounds hold. Since $f(x) \in O(g(x))$, there must exist a value $b > 0$ such that $|f(x)| \leq |bg(x)|$. Additionally, since $g(x) \in O(f(x))$, there must exist a value $b' > 0$ such that $|g(x)| \leq |b'f(x)|$. Dividing both sides of the inequality by b', we get $|\frac{1}{b'}g(x)| \leq |f(x)|$. We then join the two inequalities on the shared $|f(x)|$ term to get $|\frac{1}{b'}g(x)| \leq |f(x)| \leq |bg(x)|$. If we define a to be $\frac{1}{b'}$, this is the definition of $f(x) \in \Theta(g(x))$.

(c) For any real number, $x \leq x$. Show that for any function, $f(x) \in O(f(x))$.

Solution: Assume x is large enough that the inequalities from all O bounds hold. From the definition of O, $f(x) \in O(g(x))$ if there exists $b, N_0 > 0$ such that $|f(x)| \leq |bg(x)|$ for all $x > N_0$. Let $g(x) = f(x)$. We see that for $|b| \geq 1$, $|f(x)| \leq |bf(x)|$ holds true regardless of the value of x. Since we were able to find $b, N_0 > 0$ such that the inequality was true, $f(x) \in O(f(x))$.

(d) If $x \leq y$ and $y \leq z$, then $x \leq z$. Show that if $f(x) \in O(g(x))$ and $g(x) \in O(h(x))$, then $f(x) \in O(h(x))$

Solution: Assume x is large enough that the inequalities from all O bounds hold. If $f(x) \in O(g(x))$, there exists $b > 0$ such that $|f(x) \leq |bg(x)|$. If $g(x) \in O(h(x))$, there exists b' such that $|g(x) \leq |b'h(x)|$. Substituting the second inequality into the first, we get that $|f(x) \leq |bb'h(x)|$. This is the definition of $f(x) \in O(h(x))$.

(e) For any pair of real numbers x and y, either $x < y$, $x = y$, or $x > y$. Show that this is NOT a property of O; that is, find functions f and g such that $f(x) \notin O(g(x))$ and $g(x) \notin O(f(x))$.

Solution: Any pair of functions that cannot consistently upper-bound each other would work here. An example of such a pair is $x * (1 + \sin(x))$ and $x * (1 + \cos(x))$.