Modeling Complex Algorithms

Data Structures and Algorithms

Warm Up

Construct a mathematical function modeling the worst-case runtime of the following method. Your function should be written in terms of n, the provided input.

Assume each println takes some constant time c to run.

```
public void mystery(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n * n; j++) {
            System.out.println("Hello"); +1     n²
        }                                            n
        for (int j = 0; j < 10; j++) {
            System.out.println("world"); +1     10
        }
    }
}
```

Remember: work outside in

Solution: $T(n) = n(n^2 + 10) = n^3 + 10n$

Socrative:

www.socrative.com

Room Name: CSE373

Please enter your name as: Last, First

Project 1 Out

Project 1 out – Part 1 due Friday April 13th

- Pair Programming
- Navigating the Project
- How it will be graded
- Need a partner? Come see me after class or email me ASAP

HW 1 due tonight!

- please make sure your legal name is associated with your practice-it account

Class Survey due tonight

Modeling Complex Loops

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!");
    }
}
```

+1 n n $f(n) = n^2$

Keep an eye on loop bounds!

Modeling Complex Loops

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!"); +c
    }
}
```

0 + 1 + 2 + 3 +…+ n-1

n

Summation
1 + 2 + 3 + 4 +… + n = $\sum_{i=1}^{n} i$

Definition: Summation

$\sum_{i=a}^{b} f(i)$ = f(a) + f(a + 1) + f(a + 2) + … + f(b-2) + f(b-1) + f(b)

T(n) = $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c$

Simplifying Summations

```
for (int i = 0; i < n; i++) {
   for (int j = 0; j < i; j++) {
      System.out.println("Hello!"); +c
   }
}
```

$$\underbrace{\underbrace{+c}_{0 + 1 + 2 + 3 + \ldots + n-1}}_{n}$$

(0c + 1c + 2c + 3c + … + i-1c)
+ (0c + 1c + 2c + 3c + … + i-1c)
+ (0c + 1c + 2c + 3c + … + i-1c)
+ repeat n times

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci$$ Summation of a constant

$$= c\sum_{i=0}^{n-1} i$$ Factoring out a constant

$$= c\frac{n(n-1)}{2}$$ Gauss's Identity

$$= \frac{c}{2}n^2 - \frac{c}{2}n$$ $O(n^2)$

Function Modeling: Recursion

```
public int factorial(int n) {
    if (n == 0 || n == 1) {    +3
        return 1;  +1                  } +c
    } else {
        return n * factorial(n - 1);  +????
}
```

Function Modeling: Recursion

```
public int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
}
```

$+c_1$

$+T(n-1)$

$+c_2$

$$T(n) = \begin{cases} C_1 & \text{when } n = 0 \text{ or } 1 \\ C_2 + T(n-1) & \text{otherwise} \end{cases}$$

Definition: Recurrence

Mathematical equivalent of an if/else statement
f(n) =

Unfolding Method

$$T(n) = \begin{cases} C_1 & \text{when } n = 0 \text{ or } 1 \\ C_2 + T(n-1) & \text{otherwise} \end{cases}$$

$$T(3) = C_2 + T(3-1) = C_2 + (C_2 + T(2-1)) = C_2 + (C_2 + (C_1)) = 2C_2 + C_1$$

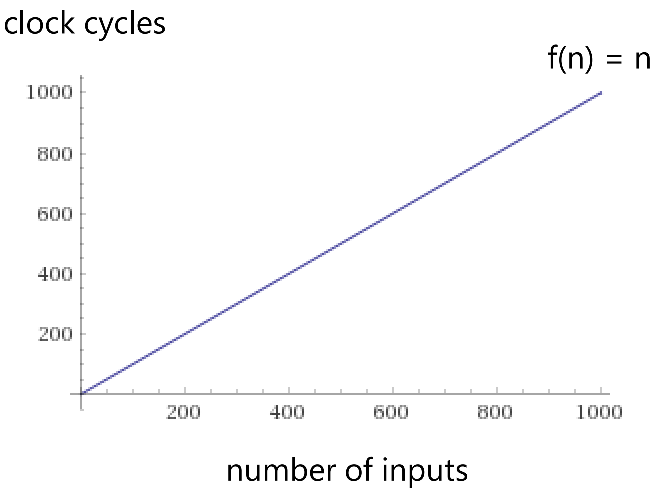
$$T(n) = C_1 + \sum_{i=0}^{n-1} C_2$$

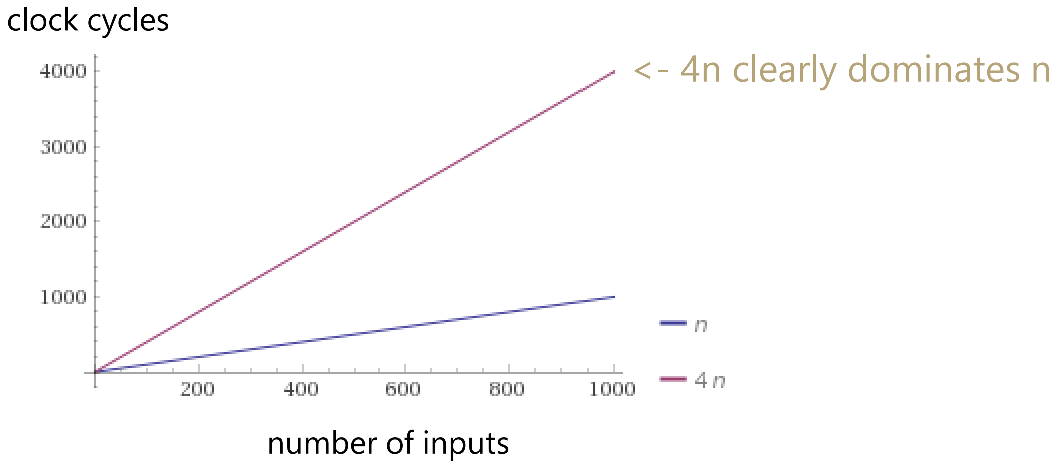
Summation of a constant

$$T(n) = C_1 + (n-1)C_2$$

Asymptotic Analysis

Asymptotic Analysis



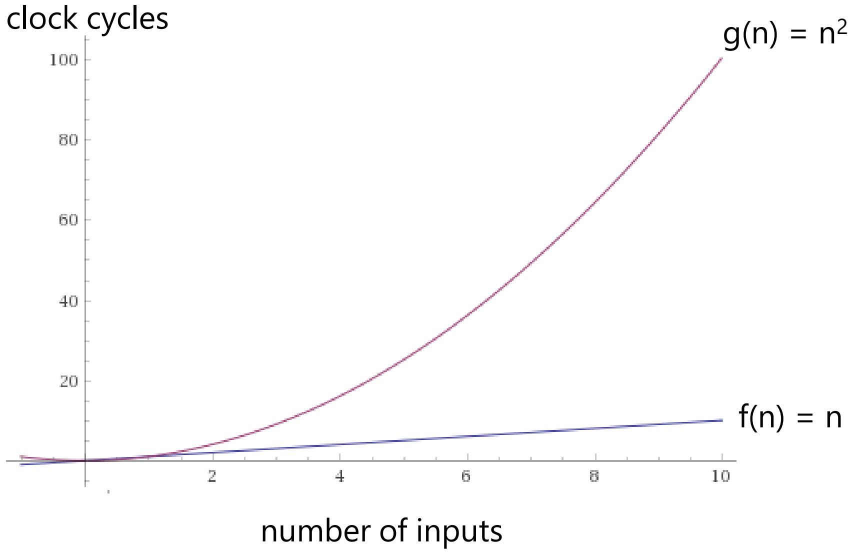


Definition: Domination

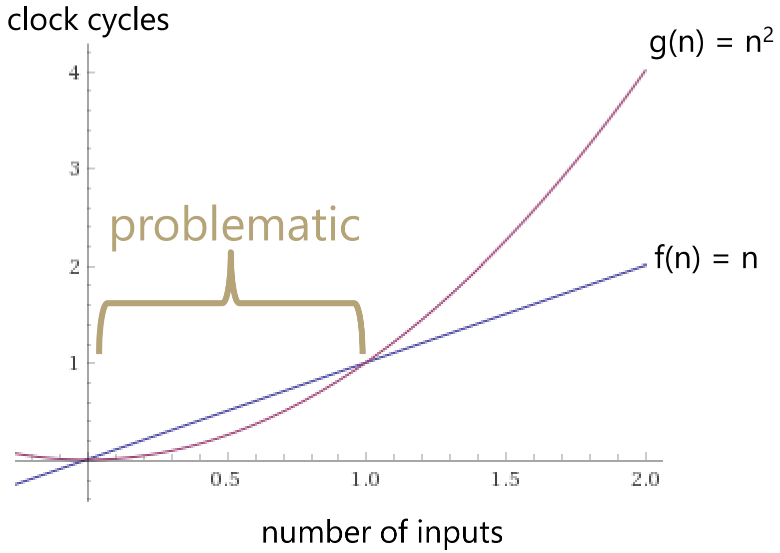
A function f(n) is **dominated** by g(n) when…
There exists two constants c > 0 and $n_0 > 0$
Such that for all values of $n \geq n_0$
$f(n) \leq c * g(n)$

Can we say that n "dominates" 4n?
Yes!
c = 4 or more
$n_0 = 1$ (because definition requires $n_0 > 0$)

Asymptotic Analysis



Zoom in ->



Clearly n^2 dominates n, right?

We can find a c & n_0 that satisfy the definition of domination
c = 1
n_0 = 1

Definition: Big O

***O*(f(n))** is the "family" or "set" of all functions that are dominated by f(n)

∈ Element Of

f(n) is less than or equal to g(n)

“f(n) is dominated by g(n)”

“f(n) is contained inside O(g(n))”

$f(n) \in O(g(n))$

Practice

$5n + 3 \in O(n)$ True

$n \in O(5n + 3)$ True

$5n + 3 = O(n)$ True

$O(5n + 3) = O(n)$ True

$O(n^2) = O(n)$ False

$n^2 \in O(1)$ False

$n^2 \in O(n)$ False

$n^2 \in O(n^2)$ True

$n^2 \in O(n^3)$ True

$n^2 \in O(n^{100})$ True

Definition: ∈

"element of" mathematical symbol

Definition: Big O

***O*(f(n))** is the "family" or "set" of <u>all</u> functions that are <u>dominated by</u> f(n)

Definitions: Big Ω

Definition: Big O

O(f(n)) is the "family" or "set" of all functions that <u>are dominated by</u> f(n)
O(f(n)) is less than or equal to f(n)
$O(f(n)) \leq f(n)$
O(f(n)) is the lower bound of f(n)'s asymptotic analysis

Do we have a name for a set of functions that are the dominators?

Definition: Big Ω

Ω(f(n)) is the family of all functions that <u>dominate</u> f(n)
Ω(f(n)) is greater than or equal to f(n)
$\Omega(f(n)) \geq f(n)$
Ω(f(n)) is the upper bound of f(n)'s asymptotic analysis

Examples

$4n^2 \in \Omega(1)$

true

$4n^2 \in \Omega(n)$

true

$4n^2 \in \Omega(n^2)$

true

$4n^2 \in \Omega(n^3)$

false

$4n^2 \in \Omega(n^4)$

false

$4n^2 \in O(1)$

false

$4n^2 \in O(n)$

false

$4n^2 \in O(n^2)$

true

$4n^2 \in O(n^3)$

true

$4n^2 \in O(n^4)$

true

Definition: Big O

O(f(n)) is the "family" or "set" of all functions that are dominated by f(n)

Definition: Big Ω

Ω(f(n)) is the family of all functions that dominates f(n)

Definitions: Big Θ

We say $f(n) \in \Theta(g(n))$ when both

$f(n) \in O(g(n))$ and $f(n) \in \Omega\ (g(n))$ are true

Which is only when $f(n) = g(n)$

Definition: Big Θ
$\Theta(f(n))$ is the family of functions that are equivalent to $f(n)$

Industry uses “Big Θ” and “Big O” interchangeably

Summary

$O(f(n)) \leq f(n) == \Theta(f(n)) \leq \Omega(f(n))$

	Dominated by $f(n) \in O(g(n))$	Dominates $f(n) \in \Omega(g(n))$
$f(n)$		
$O(1)$		
$O(\log n)$		
$O(n)$		
$O(n^2)$		
$O(n^3)$		