Asymptotics

CS61BL: It's all about efficiency

An engineer will do for a dime what any fool will do for a dollar.

- Development Cost: How long does it take you to write your program? Can you speed up the process by using IDEs, unit testing, and efficient debugging skills?
- Computational Cost
  - Program runtime: How long does your program take to run?
  - Program memory: How much memory does your program use to run?
- Your job as a computer scientist: Look at a problem and choose the right data structures and algorithms to minimize cost

A Tale of Efficiency: Convert String[] into one String

- Option A: Use String concatenation. (e.g. “hello ” + “world”)

```
private static String stringConcat(String[] strings) {
    String result = "";
    for (String str : strings) {
        result += str;
    }
    return result;
}
```

- Option B: Use a StringBuilder. It has an append(String str) method.

```
private static String stringBuilderConcat(String[] strings) {
    StringBuilder result = new StringBuilder();
    for (String str : strings) {
        result.append(str);
    }
    return result.toString();
}
```

Methods for Measuring Program Runtime

1. Time how long a program takes to execute.
    - Unix `time` command
    - Various Java classes that can be used to create timers
    - Pros:
    - Cons:

Methods for Measuring Program Runtime

1. Time how long a program takes to execute.
    - Unix `time` command
    - Various Java classes that can be used to create timers
    - Pros: Easy to measure, intuitive
    - Cons: Varies depending on computer, compiler, cosmic rays, etc

Methods for Measuring Program Runtime

1. Time how long a program takes to execute.
    - Unix `time` command
    - Various Java classes that can be used to create timers
    - Pros: Easy to measure, intuitive
    - Cons: Varies depending on computer, compiler, cosmic rays, etc
2. Count number of operations program makes.
    - Insert counters into code (e.g. How many add operations?)
    - Pros:
    - Cons:

Methods for Measuring Program Runtime

1. Time how long a program takes to execute.
    - Unix `time` command
    - Various Java classes that can be used to create timers
    - Pros: Easy to measure, intuitive
    - Cons: Varies depending on computer, compiler, cosmic rays, etc
2. Count number of operations program makes.
    - Insert counters into code (e.g. How many add operations?)
    - Pros: Doesn't depend on computer specs
    - Cons: Not the actual runtime, varies on input

Methods for Measuring Program Runtime

1. Time how long a program takes to execute.
    - Unix `time` command
    - Various Java classes that can be used to create timers
    - Pros: Easy to measure, intuitive
    - Cons: Varies depending on computer, compiler, cosmic rays, etc
2. Count number of operations program makes.
    - Insert counters into code (e.g. How many add operations?)
    - Pros: Doesn't depend on computer specs
    - Cons: Not the actual runtime, varies on input
3. Symbolize execution as a function
    - Analyze code line by line and give an estimate of runtime in terms of input
    - Pros:
    - Cons:

Methods for Measuring Program Runtime

1. Time how long a program takes to execute.
    - Unix `time` command
    - Various Java classes that can be used to create timers
    - Pros: Easy to measure, intuitive
    - Cons: Varies depending on computer, compiler, cosmic rays, etc
2. Count number of operations program makes.
    - Insert counters into code (e.g. How many add operations?)
    - Pros: Doesn't depend on computer specs
    - Cons: Not the actual runtime, varies on input
3. Symbolize execution as a function
    - Analyze code line by line and give an estimate of runtime in terms of input
    - Pros: Doesn't depend on computer specs or inputs. Tells how program scales
    - Cons: Doesn't tell you actual times

A Tale of Efficiency: Convert String[] into one String

- Option A: Use String concatenation. (e.g. “hello ” + “world”)

```
private static String stringConcat(String[] strings) {
    String result = "";
    for (String str : strings) {
        result += str;
    }
    return result;
}
```

- Option B: Use a StringBuilder. It has an append(String str) method.

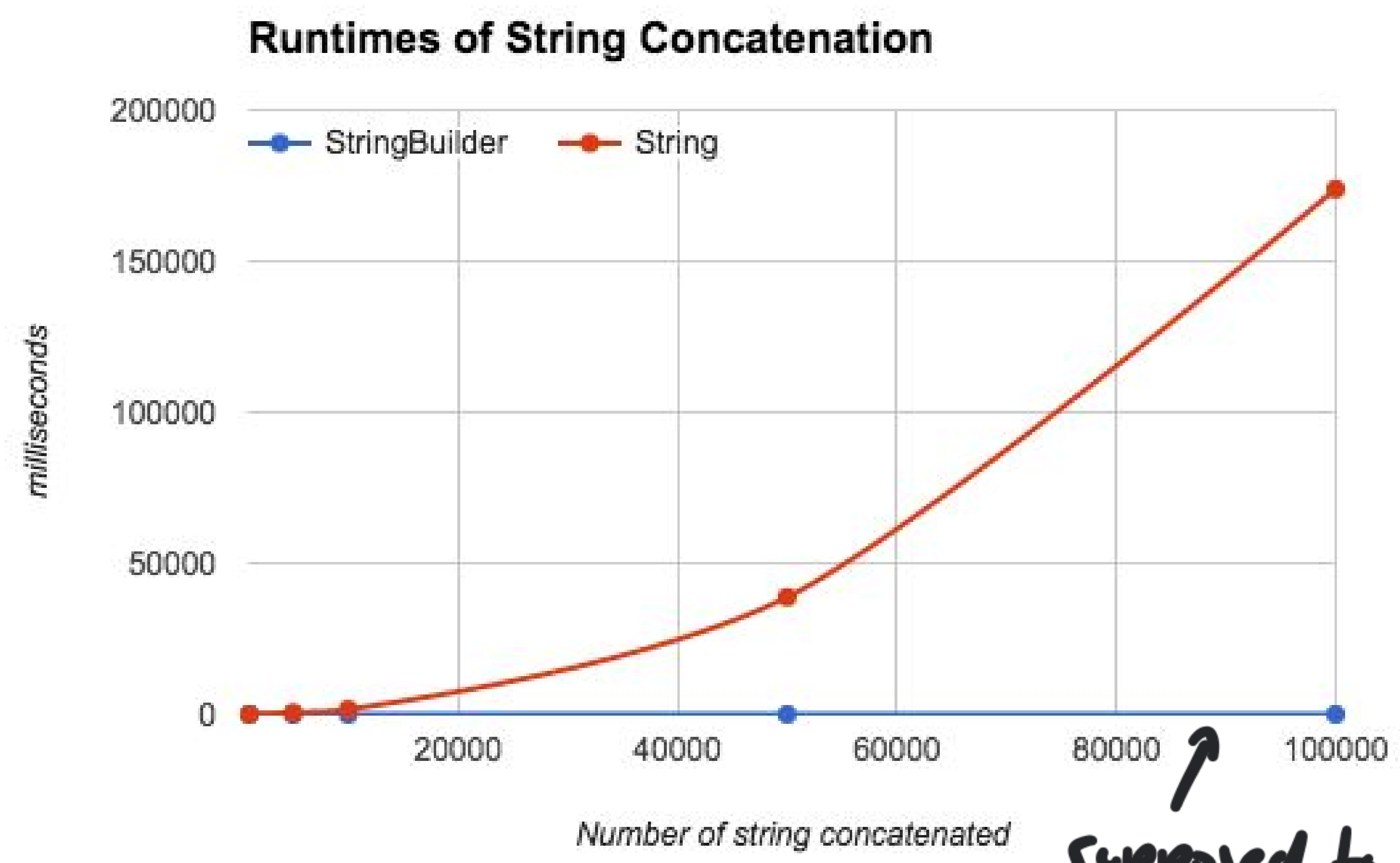
```
private static String stringBuilderConcat(String[] strings) {
    StringBuilder result = new StringBuilder();
    for (String str : strings) {
        result.append(str);
    }
    return result.toString();
}
```

A Tale of Efficiency: Convert String[] into one String

```
Number of strings concatenated: 1000
String concatenation: 34 ms
StringBuilder concatenation: 0 ms
===========
Number of strings concatenated: 5000
String concatenation: 582 ms
StringBuilder concatenation: 1 ms
===========
Number of strings concatenated: 10000
String concatenation: 1654 ms
StringBuilder concatenation: 1 ms
===========
Number of strings concatenated: 50000
String concatenation: 38812 ms
StringBuilder concatenation: 4 ms
===========
Number of strings concatenated: 100000
String concatenation: 173798 ms
StringBuilder concatenation: 12 ms
```

A Tale of Efficiency: Convert String[] into one String

```
Number of strings concatenated: 1000
String concatenation: 34 ms
StringBuilder concatenation: 0 ms
===========
Number of strings concatenated: 5000
String concatenation: 582 ms
StringBuilder concatenation: 1 ms
===========
Number of strings concatenated: 10000
String concatenation: 1654 ms
StringBuilder concatenation: 1 ms
===========
Number of strings concatenated: 50000
String concatenation: 38812 ms
StringBuilder concatenation: 4 ms
===========
Number of strings concatenated: 100000
String concatenation: 173798 ms
StringBuilder concatenation: 12 ms
```



supposed to be linear (bad scale)

Asymptotic Notation

- Cost function: how runtime (or space usage) changes with input size
- Order of growth: “shape” of the cost function
- Asymptotic notation describes orders of growth by providing bounds.

Formal Definitions

Formal Definitions of Big-O

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

positive

there exist k, N such that

$f(n) \leq k \cdot g(n) \quad \forall n > N$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

Formal Definition of Big-O in English

https://www.desmos.com/calculator/yitz7onvuj

Think of it this way: if you're trying to prove that one function is asymptotically bounded by another [f(n) is in O(g(n))], you're allowed to multiply them by positive constants in an attempt to stuff one underneath the other. You're also allowed to move the vertical line (N) as far to the right as you like (to get all the crossings onto the left side). We're only interested in how the functions behave as n shoots off toward infinity.

Big-O Challenge

Suppose we have some performance measurement $F(n)$, where n is the size of our problem.

- Suppose F(n) = 2n + 1. Find a simple g(n) and corresponding k and N

Formal Definitions of Big-Ω - Lower Bound

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

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

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$$

Formal Definition of Θ · tight bound

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

$$\Rightarrow f(n) \in O(g(n)) \text{ AND }$$
$$\in \Omega(g(n))$$

$$0 < \lim_{n \to} \frac{f(n)}{g(n)} < \infty$$ (combination of prev. definitions)

Some Tips

- In general, we do not care about constant factors
  - $O(2n) \in O(n)$
- $n^3 + 10n^2 + 20n + 7 \in O(n^3)$
  - Because we are looking at limiting behavior with a really big n, we typically only include the dominating term (i.e. the largest and slowest term)
- $n \in O(n^3)$
  - Big-O can be misleading as it is just an upper bound

Table of Important Big-O Sets

Function	Common Name
$O(1)$	Constant time
$\subset O(\lg N)$	logarithmic
$\subset O(N)$	linear
$\subset O(N \lg N)$	N lg N or linearithmic
$\subset O(N^2)$	Quadratic
$\subset O(2^N)$	Exponential

Analyzing Code Samples

Nested For Loops (http://shoutkey.com/shall)

Find a simple f(N) such that the runtime $R(N) \in \Theta(f(N))$. By simple, we mean there should be no unnecessary multiplicative constants or additive terms.

```
public static void printParty(int n) {
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j < i; j++) {
            System.out.println("free");
        }
    }
}
```

A. 1

B. lg N

C. N

D. N lg N

E. $\boxed{N^2}$

F. Something else

Nested For Loops

```java
public static void printParty(int n) {
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j < i; j++) {
            System.out.println("free");
        }
    }
}
```

$\Theta(N^2)$

i	j	prints
1	1	1
2	2	2
3	3	3
⋮		
N	N	N

$1+2+3+\ldots+N = \frac{N(N+1)}{2}$

$\frac{N^2}{2}$

work = total area

Nested For Loops: The Sequel (http://shoutkey.com/rust)

Find a simple f(N) such that the runtime R(N) ∈ Θ(f(N)). By simple, we mean there should be no unnecessary multiplicative constants or additive terms.

```
public static void printParty2(int n) {
    for (int i = 1; i <= n; i = i * 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("kekistan");
        }
    }
}
```

A. 1

B. lg N

C. N

D. N lg N

E. N^2

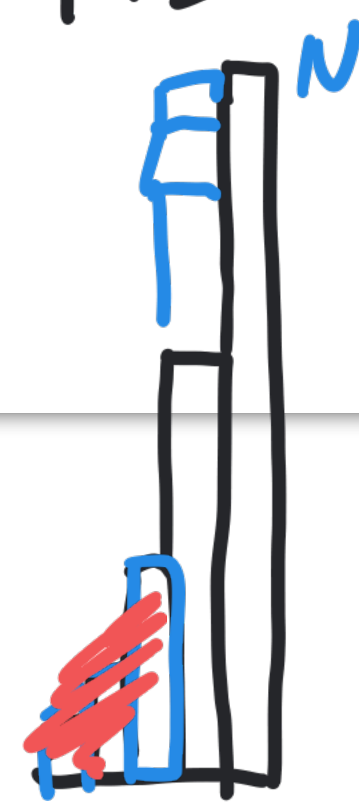
F. Something else

Nested For Loops: The Sequel

```java
public static void printParty2(int n) {
    for (int i = 1; i <= n; i = i * 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("kekistan");
        }
    }
}
```

$\Theta(N)$

i	j	prints
1	1	1
2	2	2
4	4	4
8	8	8
⋮	⋮	⋮
N	N	N



$1 + 2 + 4 + 8 + \ldots + N$

$= \sum_{i=0}^{\lg N} 2^i$

$= 2^{\lg N} - 1$

$N - 1$

Recursion (http://shoutkey.com/just)

Find a simple f(N) such that the runtime $R(N) \in \Theta(f(N))$

Using your intuition, bound the runtime of this code as a function of N?

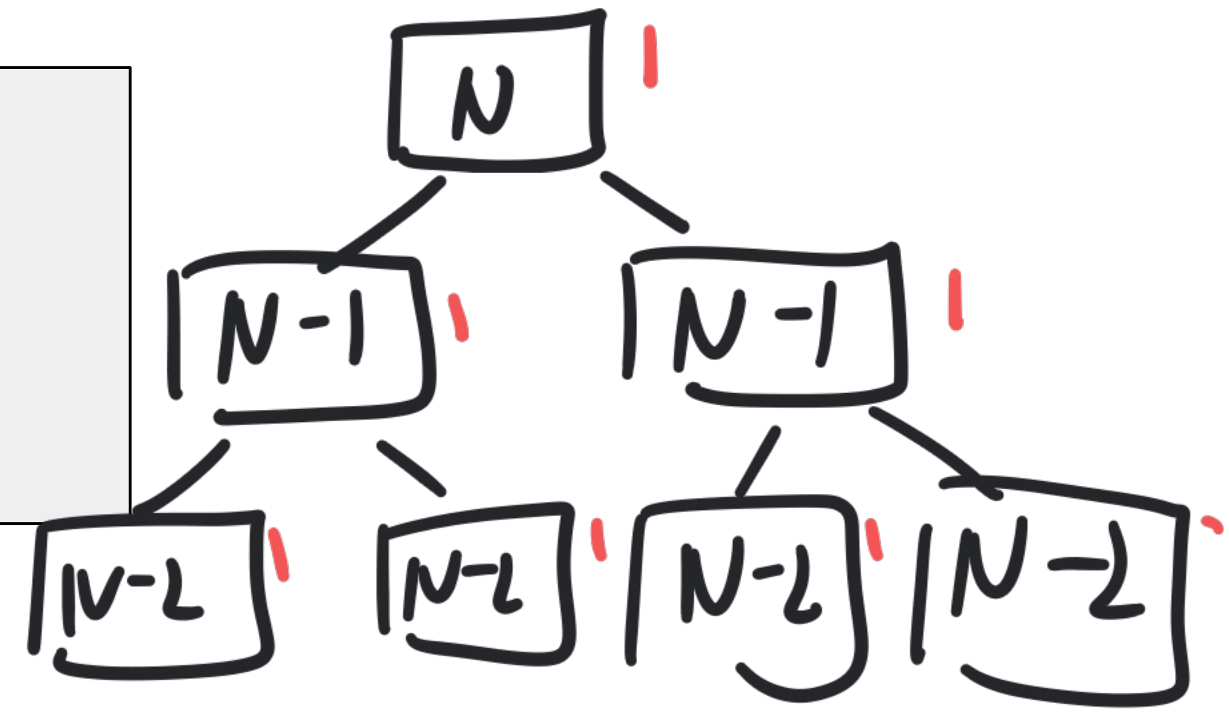
A. 1
B. log N
C. N
D. N^2
E. 2^N

```
public static int f3(int n) {
   if (n <= 1)
       return 1;
   return f3(n-1) + f3(n-1)
}
```

Recursion

2^N

```java
public static int f3(int n) {
    if (n <= 1)
        return 1;
    return f3(n-1) + f3(n-1)
}
```



$$\sum_{\text{layer}} \frac{\text{node}}{\text{layer}} \cdot \frac{\text{work}}{\text{node}} \qquad N$$

$$\sum_{i=0}^{N} 2^i (1) \in \Theta(2^N)$$

$\boxed{1}$

A Trick Question (http://shoutkey.com/peanut)

Let R(N) be the runtime of the code below as a function of N.

- What is the order of growth of R(N)?
  - A. $R(N) \in \Theta(1)$
  - B. $R(N) \in \Theta(N)$
  - C. $R(N) \in \Theta(N^2)$
  - D. Something else.

```
public boolean dupFinder(int[] a) {
    int N = a.length;
    for (int i = 0; i < N; i += 1) {
        for (int j = i + 1; j < N; j += 1) {
            if (a[i] == a[j]) {
                return true;
            }
        }
    }
    return false;
}
```

A Trick Question

Let R(N) be the runtime of the code below as a function of N.

- What is the order of growth of R(N)?
  - A. $R(N) \in \Theta(1)$
  - B. $R(N) \in \Theta(N)$
  - C. $R(N) \in \Theta(N^2)$
  - **D. Something else (depends on input)**

```java
public boolean dupFinder(int[] a) {
    int N = a.length;
    for (int i = 0; i < N; i += 1) {
        for (int j = i + 1; j < N; j += 1) {
            if (a[i] == a[j]) {
                return true;
            }
        }
    }
    return false;
}
```

/

A Trick Question

Let R(N) be the runtime of the code below as a function of N.

- What is the order of growth of R(N)?
  - It depends! In the **worst case**, $R(N) \in \Theta(N^2)$
  - BUT, for an array of all equal elements, $R(N) \in \Theta(1)$

```
public boolean dupFinder(int[] a) {
    int N = a.length;
    for (int i = 0; i < N; i += 1) {
        for (int j = i + 1; j < N; j += 1) {
            if (a[i] == a[j]) {
                return true;
            }
        }
    }
    return false;
}
```

Rumour Has It

~~Big O is for worst case.~~
~~Big Ω is for best case.~~
~~Big Θ is for average case.~~

Rumour Has It So Wrong

Big O is for worst case.

Big Ω is for best case.

Big Θ is for average case.

The Truth of the Matter: Repeat After Me

- You can use asymptotic notation (Big O, Big Ω, and Big ϴ) to express the best case, worst case, or average case.

The Truth of the Matter

- You can use asymptotic notation (Big O, Big Ω, and Big Θ) to express the best case, worst case, or average case.
- Asymptotic notation is just a *bound* on functions. **IT DOES NOT SAY WHAT THE FUNCTIONS MEAN.**

Question (http://shoutkey.com/chives)

Which statement gives you more information about the neighborhood?

A. Every house in the neighborhood is worth less than $1,000,000.
B. The most expensive house in the neighborhood is worth $1,000,000.

Question

Which statement gives you more information?

A. Every house in the neighborhood is worth less than $1,000,000.

B. The most expensive house in the neighborhood is worth $1,000,000.

Question (http://shoutkey.com/yellow)

Which statement gives you more information about the function R(N)?

A. $R(N) \in O(N^2)$.
B. In the worst case, $R(N) \in \Theta(N^2)$.

Answer

Which statement gives you more information about the function R(N)?

A. $R(N) \in O(N^2)$.
B. In the worst case, $R(N) \in \Theta(N^2)$.

Note: Even though B is a stronger statement than A, for convenience, people usually just say A.

Example:

- Runtime of Selection Sort is $O(N^2)$.
- This statement is true, but runtime is also $O(N^5)$, and $O(2^N)$.
- Stronger (but wordier) statement: In the worst case, runtime of Selection Sort is $\Theta(N^2)$.

Warning! Common Fallacies Ahead

1) ~~The best case is when N=1 (Base case)~~

$\Rightarrow \cancel{\Theta(1)}$

$\boxed{\text{N has to be big,}}$ we care about $N \Rightarrow \infty$

2) $\cancel{n^2 \in O(n) \quad k} = n$

k is a constant

3) $\cancel{e^{3N} \in O(e^{2N})}$

Repeat After Me

Repeat After Me

There is no magic shortcut for these problems (well… usually)

- Runtime analysis often requires careful thought.
- This is not a math class, though we'll expect you to know these:
  - $1 + 2 + 3 + \ldots + N = N(N+1)/2 = \Theta(N^2)$ ← Sum of First N Numbers
  - $1 + 2 + 4 + 8 + \ldots + N = 2N - 1 = \Theta(N)$ ← Sum of First N Powers of 2
- Strategies
  - Write out examples (use charts)
  - Draw pictures (recursion trees)

Citations

- Encapsulation anecdote by Jonathan Shewchuck.
- Most slides based on Josh Hug’s offering of 61B.