Data Structures

Giri Narasimhan
Office: ECS 254A
Phone: x-3748
giri@cs.fiu.edu

Hello!

- What are we here for?
  - COP 3530
  - It is a required course, but a pivotal one for BS-CS
- What is this course about?
  - Data Structures & Algorithm Analysis
    - Teaches you to plan before you act
    - Helps prior to coding to design the algorithm in a disciplined, systematic manner
    - Helps analyze and compare algorithms before implementation
    - Helps implement the most efficient programs
    - Simple to increasingly complex data structures & algorithms

General Information

- Course Website: https://users.cs.fiu.edu/~giri/teach/3530Fall2016.html
- Moodle Site: Soon!
- See Course website for
  - Syllabus
  - course objectives and learning outcomes
  - prerequisites and co-requisites
  - Required text
  - All policies, rules and regulations, including
    - Assignment Submission Policy
    - Cheating Policy
    - attendance standards

Evaluation

Programming Assignments	40%
In-class quizzes	15%
Exams	40%
Class Participation	5%

Pseudocode: Prelude to Code

- Structured, indented code
- Free format
- Skip formal syntax, declarations, and other details

Pseudocode 1:
Find kth largest among N numbers

```
SELECT-KTH-LARGEST (k, A)
    Sort(A) in decreasing order
    Report A[k]
```

- 3 considerations: Correctness? Time? Memory?

Pseudocode 2:
Report all Prime numbers between 1 and N

```
ReportAllPrimes (N)

For j = 2 to N do
    For k = 2 to j-1 do
        If (k is a factor of j) then
            Report j is not a prime and process next j
    Report j is a prime
```

- A prime number has only two divisors: 1 and itself

- 3 considerations: Correctness? Time? Memory?

Pseudocode 3:
Max Contiguous Subsequence Sum

MaxSubseqSum(*A*)

For every possible subsequence of *A*

compute Sum of subsequence

keep track of highest subsequence sum

- 3 considerations: Correctness? Time? Memory?

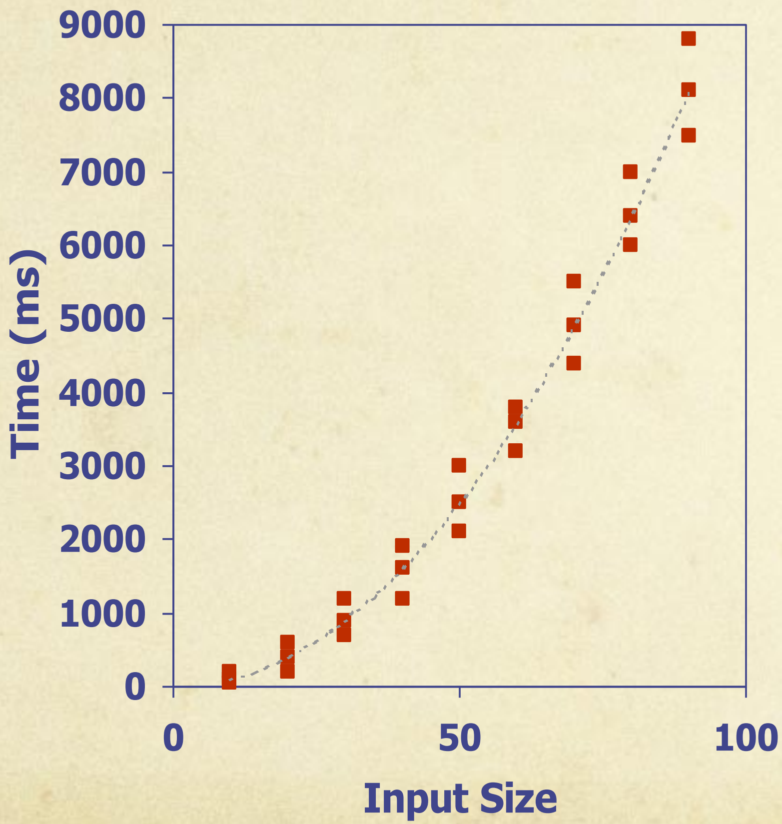
MaxSubseqSum(*A*)

```
Initialize maxSum to 0
N := size(A)
For i = 1 to N do
    For j = i to N do
        Initialize thisSum to 0
        for k = i to j do
            add A[k] to thisSum
        if (thisSum > maxSum) then
            update maxSum
```

Time Complexity Analysis

- Pseudocode is enough for prelim time & memory analysis
- Time Complexity Analysis can:
  - Give you rough estimate of time
  - Give you a sense of growth of time complexity with input size (**Asymptotics**)
  - Help you to compare two or more algorithms in a machine-independent manner
- Time Complexity Analysis cannot:
  - Inform you about correctness or memory usage
  - Account for machine-dependent, PL-dependent, and programmer-dependent differences

Why not do Timing experiments



- Implementations take time
- Experiments cannot test all inputs
- Average-Case vs Worst-Case Analysis
- Results may be machine-dependent

Asymptotic Running Time

- To compute **asymptotic running time**,
  - Consider the worst-case scenario
  - Consider the worst-case scenario & count number of steps as a function of length of input
  - Eliminate all terms except the dominant term(s)
  - Eliminate constants where possible
  - Simplify expression where possible
  - What remains is typically the asymptotic running time in **big-Oh notation**

Pseudocode 1:
Find kth largest among N numbers

SELECT-KTH-LARGEST (k, A)

Sort(A) in decreasing order

Report A[k]

- 3 considerations: Correctness? Time? Memory?
- Time ≈ time for sorting

Pseudocode 2:
Report all Prime numbers between 1 and N

- A prime number has only two divisors: 1 and itself

REPORTALLPRIMES (N)

For j = 2 to N do
    For k = 2 to j-1 do
        If (k is a factor of j) then
            Report j is not a prime and process next j
    Report j is not a prime

- 3 considerations: Correctness? Time? Memory?
- Time ≈ N^2

Pseudocode 3:
Max Contiguous Subsequence Sum

MaxSubseqSum(*A*)

For every possible subsequence of *A*

compute Sum of subsequence

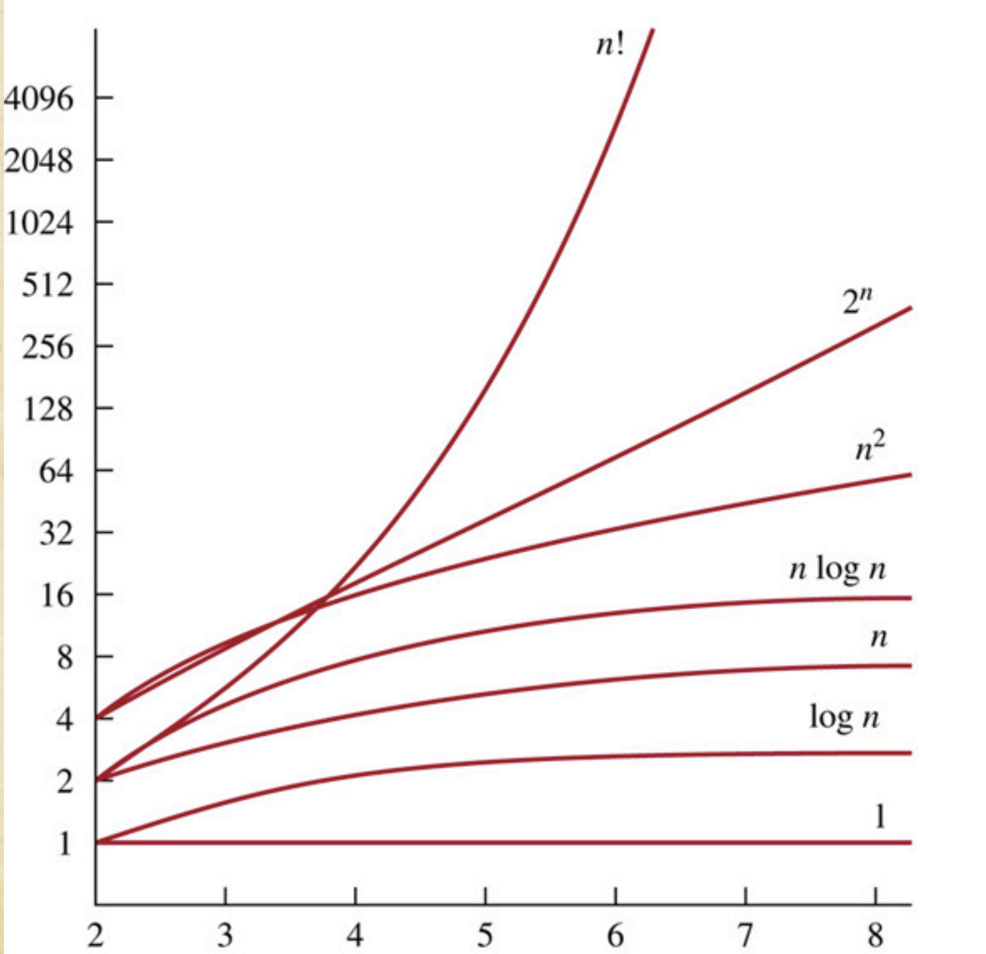
keep track of highest subsequence sum

- Time ≈ N^3

MaxSubseqSum(*A*)

```
Initialize maxSum to 0
For i = 1 to N-1 do
    For j = i to N-1 do
        Initialize thisSum to 0
        for k = i to j do
            add A[k] to thisSum
        if (thisSum > maxSum) then
            update maxSum
```

Some Growth Rates



1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n-log-n
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$	[*see page 47*]	*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`    if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$	[*see* ALGORITHM 2.4]	*divide and conquer*	*mergesort*

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`    for (int j = i+1; j < N; j++)`
		`        if (a[i] + a[j] == 0)`
		`            cnt++;`
			double loop	*check all pairs*

cubic	N^3	`for (int i = 0; i < N; i++)`
		`    for (int j = i+1; j < N; j++)`
		`        for (int k = j+1; k < N; k++)`
		`            if (a[i] + a[j] + a[k] == 0)`
		`                cnt++;`
			triple loop	*check all triples*

exponential	2^N	[*see* CHAPTER 6]	*exhasutive search*	*check all subsets*

Review your exponents, logs, series

Exponents

- $X^A X^B = X^{A+B}$
- $X^A \div X^B = X^{A-B}$
- $(X^A)^B = X^{AB}$
- Fine points
  - $X^N + X^N = 2X^N \neq X^{2N}$
  - $X^N * X^N = X^{2N}$
  - $2^N + 2^N = 2^{N+1}$

Logarithms

- $\log x^y = y \log x$
- $\log xy = \log x + \log y$
- $\log \log n = \log(\log n)$
- $\log^k n = (\log n)^k$
- $\log_y x = \log x - \log y$
- $\log_b x = \log_a x / \log_a b$

Advantages of Asymptotic Analysis & Big-Oh Notation

- Allows for rough measure of running time
- Abstracts main features of code without focusing on details of implementation or hardware or language or environment
- Tells us how time complexity scales with input size
- Allows for a quick high-level comparison of algorithms