Welcome to CIS 675!

Welcome to CIS 675!

- Course Topics
- Course personnel
- Blackboard
- Homework & Exams
- Make-up Policy
- Extra Credit
- Professionalism
- Academic Integrity
- Proof Writing

CIS 675: Course Topics

- Analyzing running time of algorithms:
  - Big-O analysis
  - Recurrence relations
- Designing algorithms
  - Divide-and-conquer
  - Greedy algorithms
  - Dynamic programming
  - Linear programming
- Complexity classes
  - P vs. NP
  - Reductions & NP-completeness

CIS 675: Course Personnel

- Instructor: Prof. Sucheta Soundarajan
  - E-mail: susounda@syr.edu
  - Office hours: Tuesdays, 12:30pm-1:30pm
- TA: Zeinab (Sara) Saghati Jalali
  - E-mail: zsaghati@syr.edu
  - Office hours: Fridays, 2pm-3pm
- TA: James Kotary
  - E-mail: jkotary@syr.edu
  - Office hours: Mondays, 1pm-2pm
- All office hours on Blackboard Collaborate

grading questions → TAs (e-mail is preferable)

CIS 675: Blackboard

- Blackboard will contain everything for this course: meetings, information, assignments, etc.
- Assignments must be submitted through Blackboard
- If you don’t have access, let me know ASAP!

CIS 675: Homework

- There will be 7 homework assignments during the semester
- I will drop the lowest scoring assignment
- Homeworks will typically contain 3-5 problems

CIS 675: Homework Policies

- Homeworks must be submitted through Blackboard as pdf files typed
  - This does not mean that you can handwrite it, take a picture, and convert to pdf
- I suggest that you use ~~LaTeX~~ to write your solutions! (http://www.latex-project.org/) overleaf.org .com? .tex
- I'll pick a few problems from each assignment to grade, but post solutions to all problems
- Homeworks are due at midnight "anywhere on Earth" (this works out to 7am the next day, Eastern time)

CIS 675: Homework Policies

- Collaboration and use of outside sources on homework is permitted!
- ***But you must write up your own answers!***
- If you copy answers from another student, the internet, etc., this is an academic integrity violation!

can't ask the Internet to solve for you

CIS 675: Exams

- 3 non-cumulative exams
- Exams will be oral exams
  - You will sign up for a 20-30 minute timeslot with an examiner (me or a TA)
  - You get 6-8 questions a week in advance (can use outside sources, but I recommend not)
  - The examiner picks 2, you must explain your solution, answer questions, and then explain how you'd modify your solution if the problem changed

CIS 675: Make-up Policy

- If you have a documented emergency, let me know as soon as practical!
- Without documentation, no make-ups
- With documentation, make-ups at the discretion of the instructor

CIS 675: Make-up Policy

- But even if you miss homeworks or exams, that is ok!
- I drop the lowest homework score
- I will drop the lowest of 6 exam problems

CIS 675: Extra credit

- There will be extra credit opportunities throughout the semester

CIS 675: First Extra Credit Assignment

- Due by midnight Friday, February 12
- Tell me about yourself!
  - Mathematical background?
  - Any algorithms course before?
  - What degree program are you in?
  - What do you hope to learn from this course?
  - Anything else I should know?
- See Blackboard for full assignment!
- Worth 0.5% of extra credit!

CIS 675: Second Extra Credit Assignment

- You can sign up to take notes for one class session during the semester
- Take good notes and write them up nicely (in pdf)
- Send me the notes and I will post them on Blackboard so that other students can see
- Worth up to 2% extra credit (depends on how good your notes are)
- If interested, send me the date that you'd like to take notes for! (Ok if multiple students pick same date)
- Must let me know by Sunday at midnight (February 14)

CIS 675: Professionalism

- Be respectful to the instructor, TA, and other students!
- When sending an e-mail, include the course name in the subject and a clear summary of what your e-mail is about
  - "Homework 3 for CIS 675"
- In your e-mails, include a proper salutation ("Hi Prof. Soundarajan") and use proper English.

CIS 675: Academic Integrity

- Syracuse University takes academic integrity very seriously!
- As graduate students, if you commit an academic integrity violation, you can be suspended from the university (even if it's your first violation!)
- ***Read the academic policy carefully!*** ***http://supolicies.syr.edu/ethics/acad_integrity.htm***

CIS 675: Academic Integrity

- What's allowed?
  - Working with others on homework
  - Looking at someone else's class notes outside of class
- What's not allowed? (These are just examples!)
  - Copying someone else's homework solutions
  - Directly copying homework solutions from the internet
  - Using notes during exams (allowed to refer to your solutions)
  - Using a cell phone to ask someone for answers to an exam
  - If unsure, ask the instructor!
- ***Don't cheat! I don't want you to get suspended!***

CIS 675: How to Write a Proof

- Many of your homework assignments will require you to prove something
- Clearly state what you are trying to prove, and include all necessary steps to demonstrate the claim that you are making
- If I can't understand what you are trying to prove, or your proof is missing steps, you may receive 0 points for that problem

CIS 675: How to Write a Proof

Example: Prove that the sum of two odd integers is even.

Claim: The sum of two odd integers is even.

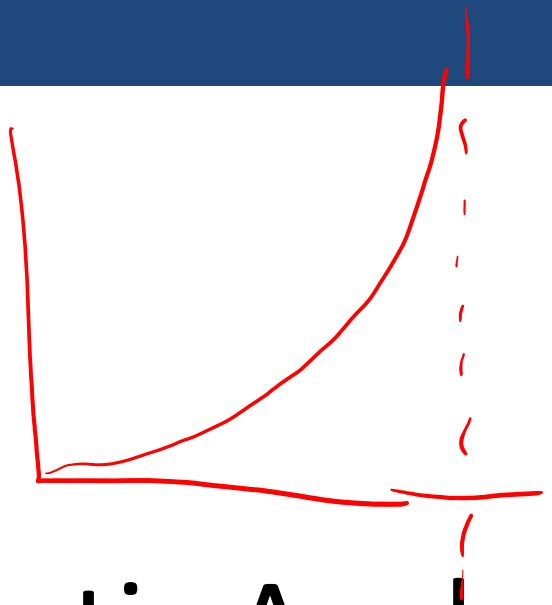
Proof: An odd integer x is one that can be written as $x = 2k+1$ for some integer k.

So suppose $x = 2k+1$, $y = 2j+1$ are both odd.

$x + y = 2k+1+2j+1 = 2(k+j+1)$. $k+j+1$ is an integer, so $x+y$ is 2 times an integer, which makes it even. □

CIS 675: How to Write a Proof

- Watch the supplemental video on Blackboard for more background on proofs
  - I highly recommend this if you don't have a lot of recent experience with proof-writing!



Asymptotic Analysis

Why Analyze Running Time?

- Suppose you've created a new algorithm
- How do you convince people to use it?
- Does it…
  - Run faster?
  - Use fewer resources?

Why Analyze Running Time?

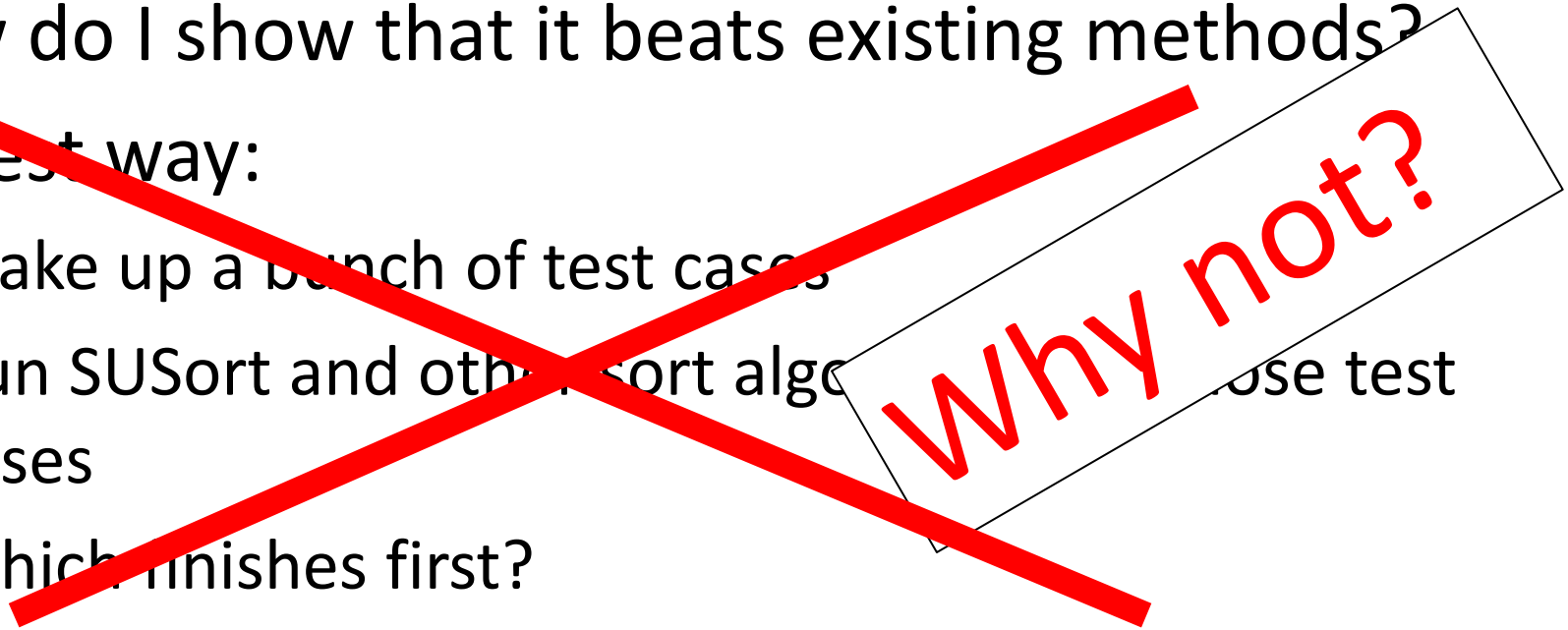
- Suppose I've invented a new sorting algorithm, **SUSort**
- How do I show that it beats existing methods?

Why Analyze Running Time?

- Suppose I've invented a new sorting algorithm, **SUSort**
- How do I show that it beats existing methods?
- Easiest way:
  - Make up a bunch of test cases
  - Run SUSort and other sort algorithms on those test cases
  - Which finishes first?

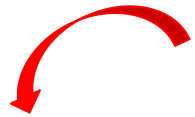
Why Analyze Running Time?

- Suppose I've invented a new sorting algorithm, **SUSort**
- How do I show that it beats existing methods?
- Easiest way:
  - Make up a bunch of test cases
  - Run SUSort and other sort algorithms on those test cases
  - Which finishes first?

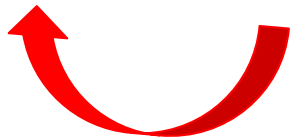


Why Analyze Running Time?

- Suppose I've invented a new sorting algorithm, **SUSort**
- How do I show that it beats existing methods?
- Easiest way:
  - Make up a bunch of test cases
  - Run SUSort and other sort algorithms on those test cases
  - Which finishes first?



How do you know they're the 'right' test cases?



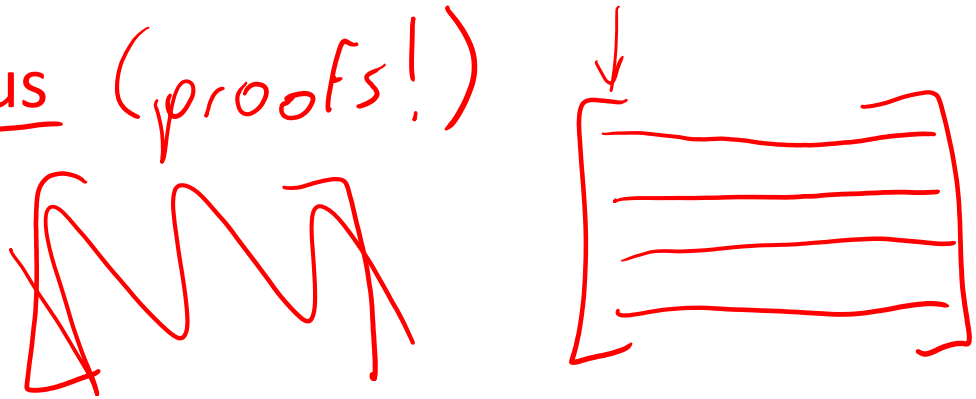
Computers get faster; data gets bigger… will the ordering change next year?

Why Analyze Running Time?

- Data sizes are doubling every year!
- Moore's Law: density of transistors doubles every 18-24 months: more processing speed!
- If your program takes 2 minutes to run today…how long will it take next year?

Key Observations

- Run-time analysis should be:
  - Independent of the platform (hardware, OS, language)
  - Independent of the programmer's skill
  - Independent of specific test cases (content and size!)
  - Theoretically rigorous (proofs!)



So How Can We Do Better?

- Theoretical analysis of algorithms is used to estimate the run-time of an algorithm as a function of the size of the input
- This run-time is given in terms of the number of primitive operations required (e.g., arithmetic operations) $O(1)$

So How Can We Do Better?

Run time = 5 seconds

Depends on input & machine set-up, compiler, etc.

vs.

Run time = 5000 primitive operations

Depends on input content and size

vs.

Run time = 500*n*, where *n* is length of specific type of input

Depends on input content

vs.

Worst-case run time = 900*n*, where *n* is length of input

Let's do an example

```
function sumArray(arr):
    tot_sum = 0
    tot_product_sum = 0

    for idx in range(len(arr)):
        tot_sum += arr[idx]

    for idx1 in range(len(arr)):
        for idx2 in range(len(arr)):
            tot_product_sum += arr[idx1] * arr[idx2]

    return tot_sum, tot_product_sum
```

N elements

$4 + 7n + 7n^2$

1

1

2n

n increments

n writes

n iterations

3 ops/iteration for a total of 3n

lookup

2n

$2n^2$

n^2 times

$5n^2$

2

Big-O Notation: Definition

$f(x) = O(g(x))$ means that

- There exists:
  - some positive constant M
  - some minimum x-value x_0
- Such that for all $x > x_0$:
  - $f(x) \leq M * g(x)$

Big-O Notation: Definition

$O(g(x)) = f(x)$

f(x) = O(g(x)) means that

- There exists:
  - some positive constant M
  - some minimum x-value x_0
- Such that for all $x > x_0$:
  - $f(x) \leq M * g(x)$

Important note:
O(g(x)) is actually a set!

When we say "f(x) = O(g(x))", we are actually saying that $f(x) \in O(g(x))$.

$4 + 7n + 7n^2 = O(n^2)$

$(M = 100, n_0 = 1000000)$

$100\,n^2$

for all $n > n_0$, $4 + 7n + 7n^2 \leq M \cdot n^2$

Big-O Notation: Notation Conventions

- O(g(x)) represents a set, so the = sign doesn't mean what it normally means
- We use "=" to represent set membership
- This means that "equality" is not symmetric!

We can say f(x) = O(g(x)), but not O(g(x)) = f(x)!

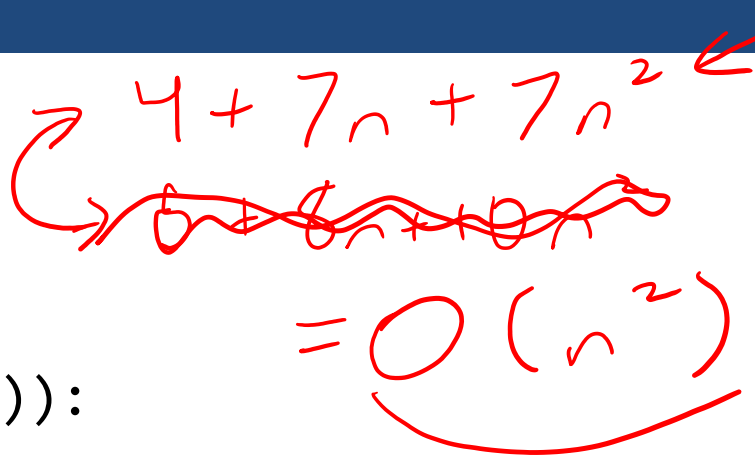
Big-O Notation: Example

```
function sumArray(arr):
    tot_sum = 0
    tot_product_sum = 0

    for idx in range(len(arr)):
        tot_sum += arr[idx]

    for idx1 in range(len(arr)):
        for idx2 in range(len(arr)):
            tot_product_sum += arr[idx1] * arr[idx2]

    return tot_sum, tot_product_sum
```



Other Types of Asymptotic Relationships

- Little-o notation: f(x) = o(g(x)) iff:
  - For every positive ε,
  - There exists a constant x_ε
  - Such that $f(x) \le \varepsilon g(x)$ for all $x \ge x_\varepsilon$

Informally, this means that g(x) grows much faster than f(x): for EVERY ε, no matter how small, we can find a place where g(x) is 1/ε-times bigger than f(x). In other words, even if we shrink g(x) by a factor of 100, 1000, 1,000,000, …, it is still going to be bigger than f(x).

Question

Q: What is the key difference between the big-O relationship and the little-o relationship?

Question

Q: What is the key difference between the big-O relationship and the little-o relationship?

$\varepsilon = \frac{1}{100}$ $\quad x \leq \frac{2x}{100}$ if x is large enough? NO

A: If f(x) = o(g(x)), it is also O(g(x)). But if f(x) = O(g(x)), it is not necessarily o(g(x))!

$f(x) = x \quad g(x) = 2x \quad f(x) \neq o(g(x))$

Can anyone think of an example of this?

$f(x) = x$
$g(x) = x^2$

$f(x) = O(g(x))$
$M = 1 \quad x_0 = 2$

Other Types of Asymptotic Relationships

- Big-Omega notation:

$$f(x) = \Omega(g(x)) \text{ iff } g(x) = O(f(x))$$

- Big-Theta notation:

$$f(x) = \Theta(g(x)) \text{ iff } f(x) = O(g(x)) \text{ and } g(x) = O(f(x))$$

↑ $f(x) = g(x)$?

Other Types of Asymptotic Relationships

- Big-Omega notation:

$$f(x) = \Omega(g(x)) \text{ iff } g(x) = O(f(x))$$

Interpretation: f(x) is bounded below by g(x)

- Big-Theta notation:

$$f(x) = \Theta(g(x)) \text{ iff } f(x) = O(g(x)) \text{ and } g(x) = O(f(x))$$

Interpretation: f(x) is bounded above and below by g(x)

Question

Q: If f(x) is Θ(g(x)), does that mean that f(x) and g(x) are the same function?

$f(x) = x$

$g(x) = 2x$

Summary

Notation	Interpretation
$f(x) = O(g(x))$	$f(x)$ is asymptotically bounded above by $g(x)$
$f(x) = o(g(x))$	$g(x)$ grows much faster than $f(x)$
$f(x) = \Theta(g(x))$	$f(x)$ and $g(x)$ grow at roughly the same rate
$f(x) = \Omega(g(x))$	$f(x)$ is bounded below by $g(x)$