

CS218: Design And Analysis Of Algorithms

Yan Gu

Welcome Back!



Class information

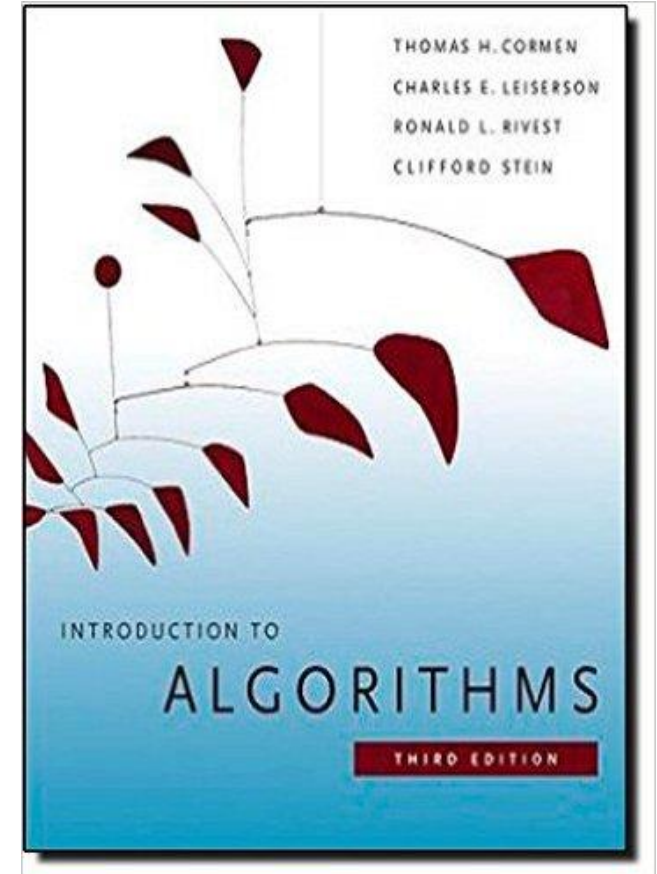
- **Classes:** TTh 12:30–1:50pm
- **Instructor:** Yan Gu
- **Email:** ygu@cs.ucr.edu
- **OH:** TTh 1:50-2:20pm (after class)
- **Website:** <https://www.cs.ucr.edu/~ygu/teaching/218/W24/index.html>
- **Useful links:** <https://campuswire.com/c/G40C1B47A/feed/2>
- **Policy:** <https://campuswire.com/c/G40C1B47A/feed/3>

Class information

- **Campuswire: offline Q&A**
- **GradeScope: submitting your homework assignments**
- **CodeForces: submit your programming solutions (yes, we have programming assignments!)**
- **Posting your questions on Campuswire is highly encouraged: let your questions help others!**
- **Please register ASAP!**

Textbook

- Introduction to Algorithms. Third Edition
- Cormen, Leiserson, Rivest, and Stein
- <https://ebookcentral.proquest.com/lib/ucr/detail.action?pq-origsite=primo&docID=3339142>
- But mostly based on our slides, the book will provide some useful readings and more details



Covered topics

- **Analysis of algorithms**
- **Divide-and-conquer algorithms**
- **Greedy algorithms**
- **Dynamic programming (DP)**
- **Graph algorithms**
- **Data structures**

A typical interview for IT companies

- **Round 1: Online assessment (OA): 2 **algorithm** questions**
 - Must pass all test cases that you cannot see
 - The recruiter will not review your resume if you do not pass the OA
- **Round 2: Technical phone interview (1 or 2): solve **algorithm** questions**
 - Write the code while explain correctness / time complexity to the interviewer
 - Some behavior questions but usually easier to prepare
- **Round 3: Onsite interviews (or similar online)**
 - 4-6 interviews. A lot of them are coding interviews about **algorithms**
 - Also some system design / leadership questions (usually lower bar for junior people)

What are the typical problems? ([link](#))

2

Google coding interview questions types



IGotAnOffer

#1	Graphs / Trees <i>E.g. Range sum of a binary search tree</i>	39%
#2	Arrays / Strings <i>E.g. Product of an array except self</i>	26%
#3	Dynamic Programming <i>E.g. Continuous subarray sum</i>	12%
#4	Recursion <i>E.g. Given a 2D board and a list of words, find all words in the board.</i>	12%
#5	Geometry / Math <i>E.g. Given a set of point on an xy-plane, find the rectangle with minimum area</i>	11%

Source: Glassdoor.com data for Google software engineer interview questions. Analysis by IGotAnOffer.

Why do companies emphasize algorithm design and programming so much?

- **Good algorithms means efficiency, effectiveness, good end-user experience, eco-friendliness, etc., brings up benefits immediately!**
 - Saves resource (e.g., power, time)
 - Provides high-quality solutions (e.g., friend/topic recommendation)
 - Provides real-time response to users (e.g., search engine, game rendering)
 - More up-to-date information (e.g., high-frequency trading)
- **They want to recruit people that are**
 - Skilled at using existing algorithms to solve simple problems (e.g., the interview problems)
 - Likely to be able to design efficient new algorithms needed in the future

Friend Recommendation

Find all my friends' friends?

- **Input: a social network and a certain user**
- **Output: a list of their friends' friends**
- **Model**
 - Social network => graph
 - User => vertex
 - Friend relationship => edges
- **Problem formalization: find all two-hop neighbors of a vertex**
- **Which algorithm can visit vertices in a graph based on the hop-distance from a vertex?**
- **Solution: Using Breadth-First Search for two hops (which should be covered in CS10C)**

DNA Sequences

Which DNAs are more similar?

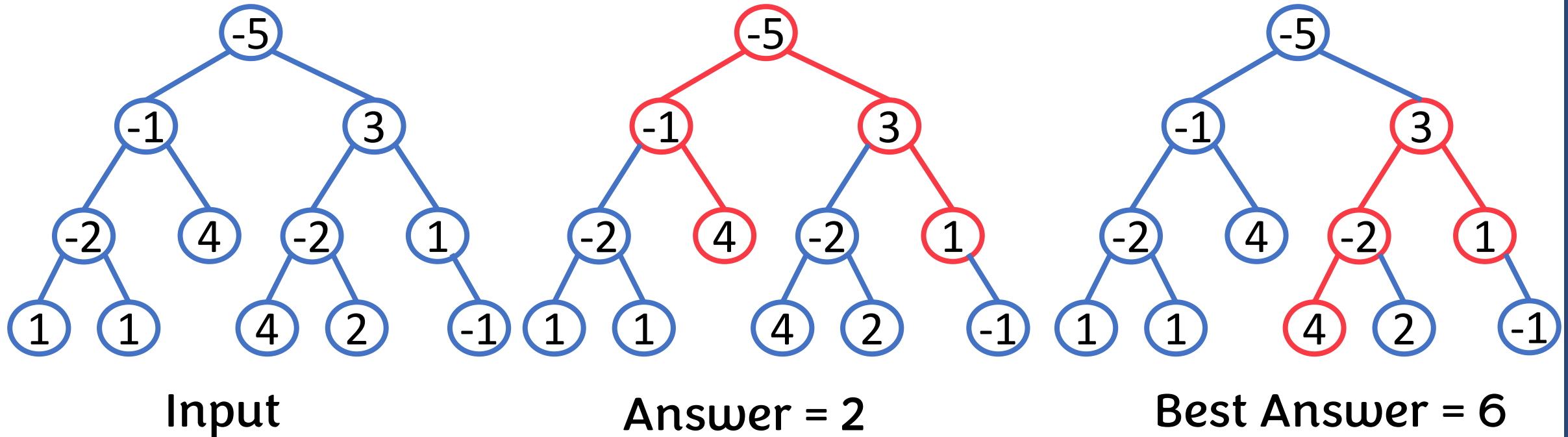
- **Input: Three DNA sequences A, B, C**
 - ATGCGAGGCTTAACAGCAAAAAGTGACTAGTGGT
 - ATGGTGGCTTAACCAGCAAAAAGTGACTAGTGGT
 - TTGGAGGGCTTAACAGCAAGAGTGACTAGTGTA
- **Output: which of B and C is more similar to A?**
- **Model**
 - DNA sequences => strings
 - Similarity => Longest common sequence / edit distances
- **Problem formalization: Compute the LCS or edit distance between A and B, and between A and C**
- **How to compute LCS or edit distance?**
- **Solution: Dynamic programming (which should be covered by CS141)**

What are the typical problems?

- Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree.
- Find the minimum number required to insert into a word to make it a palindrome.
- Given a matrix of N rows and M columns. From $m[i][j]$, we can move to $m[i+1][j]$, if $m[i+1][j] > m[i][j]$, or can move to $m[i][j+1]$ if $m[i][j+1] > m[i][j]$. The task is print longest path length if we start from $(0, 0)$.
- Given a collection of intervals, merge all overlapping intervals

What are the typical problems?

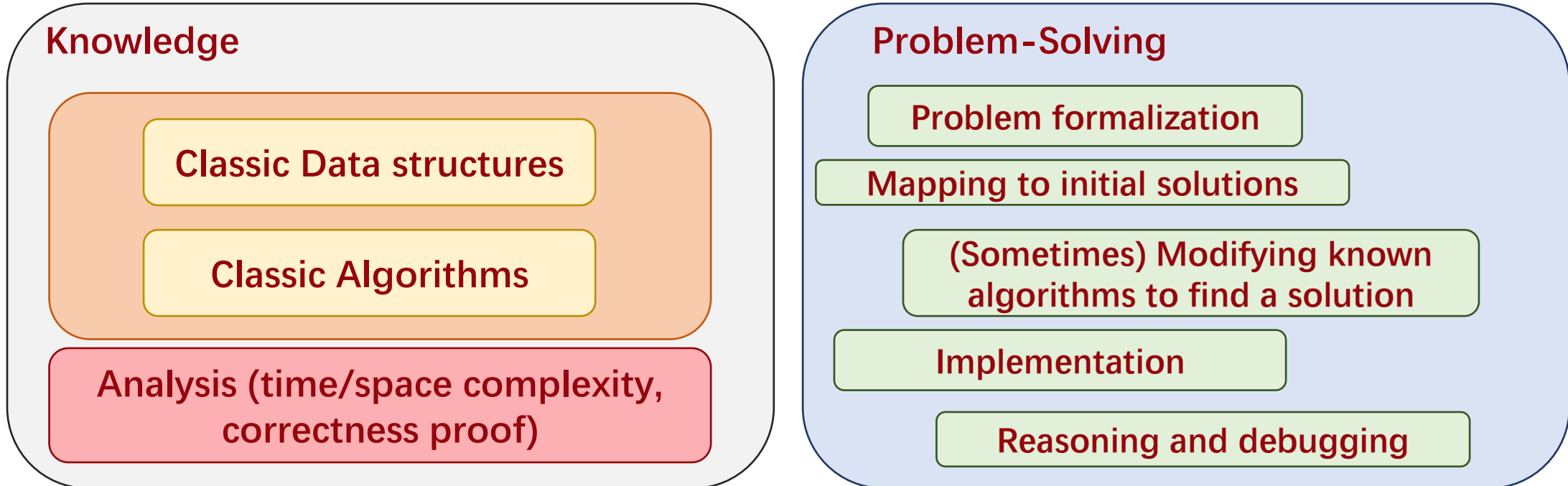
- Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree.



- Solution: Dynamic programming + Postprocessing (how?)
- What happens if it is an arbitrary tree instead of a binary tree?

We learn algorithms because we want to **solve problems**

- Real world problems may look fancy, but they are just algorithm problems
- We need to learn how to model the problem and design an algorithm for it
- We need to learn how to combine different algorithms to solve a more complicated problem / modify an algorithm to address special needs
- Learning algorithm is not only to memorize the details, but to create new ones on needs



Why are algorithms important?



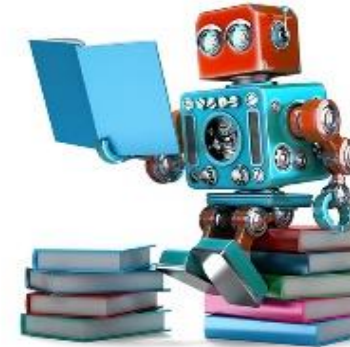
**Database /
Data warehouses**



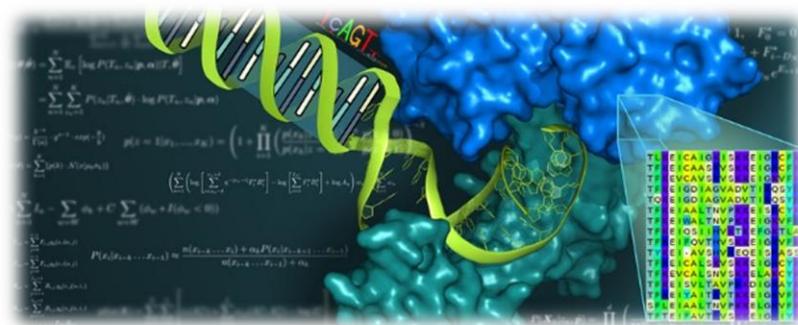
**Data mining /
Data science**



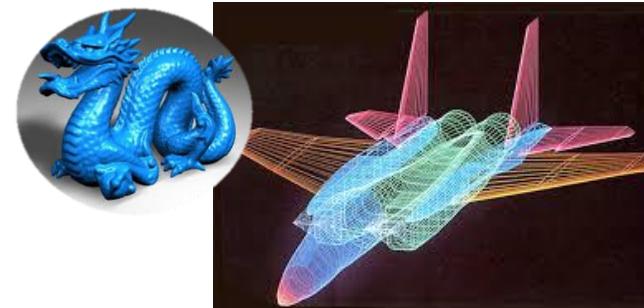
**Machine learning /
Artificial intelligence**



Computational biology

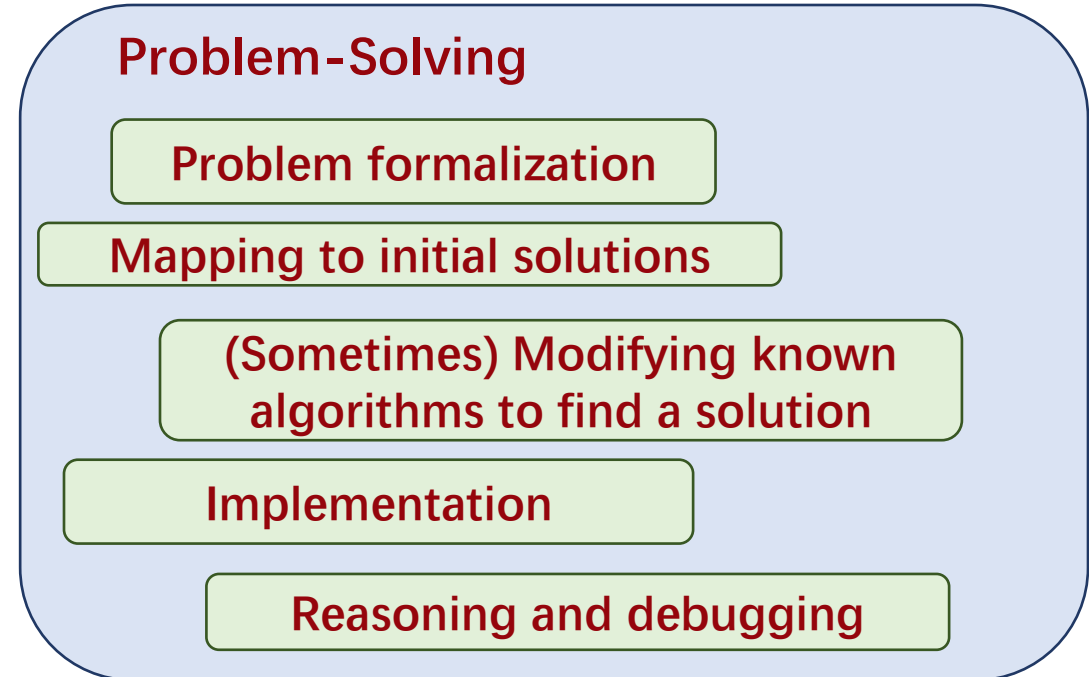
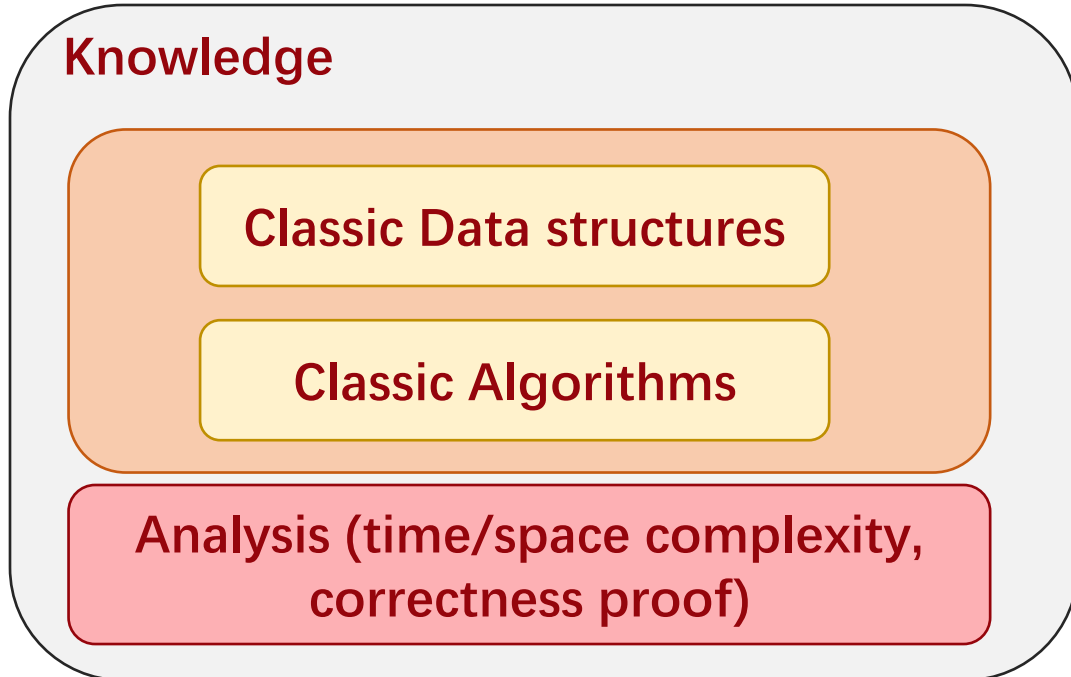


**Computer graphics /
computational geometry**



We learn algorithms because we want to **solve problems**

- Real world problems may look fancy, but they are just algorithm problems
- We need to learn how to model the problem and design an algorithm for it
- We need to learn how to combine different algorithms to solve a more complicated problem / modify an algorithm to address special needs
- Learning algorithm is not only to memorize the details, but to create new ones on needs



We learn algorithms to **solve problems**

- Not just to know some fancy names and simulate them on an input instance with 5 or 8 elements
- Hence, for my CS 218, we are not going to cover too many fancy algorithms since if you don't know how to implement them and use them in practice, that's equivalent to you don't know them
- **However, the course is still very challenging since:**
 - (1) You might not have known how to implement algorithms in your previous algorithm courses (e.g., Dijkstra's algorithm) and apply them to real-world problems
 - (2) We cannot teach you how to solve real-world problems: it is an ability that you can practice and I will help you practice, but it's more challenging than memorizing certain knowledge (that's why everyone interviews you using these problems)

Everyone has a different background / goal

- **8 of you are PhD students:**

- Some of you are working on algorithms or related problems with very good algorithmic background, and want to learn some more involved knowledge
- The other of you are PhDs but do not work closely on algorithms, so you want to learn something related to or helpful with your research
- Some PhDs from other departments who wants to know more about CS

- **Most of you are master students:**

- Most of you are aiming for a good job, and want this course to help you with your interviews
- Some of you are doing research and considering PhDs in the future

- **(Previously) About 10% of you are undergraduate students who want to learn more algorithms / try graduate courses / help your research**

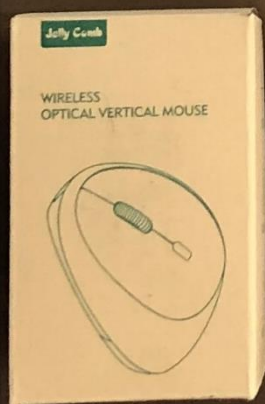
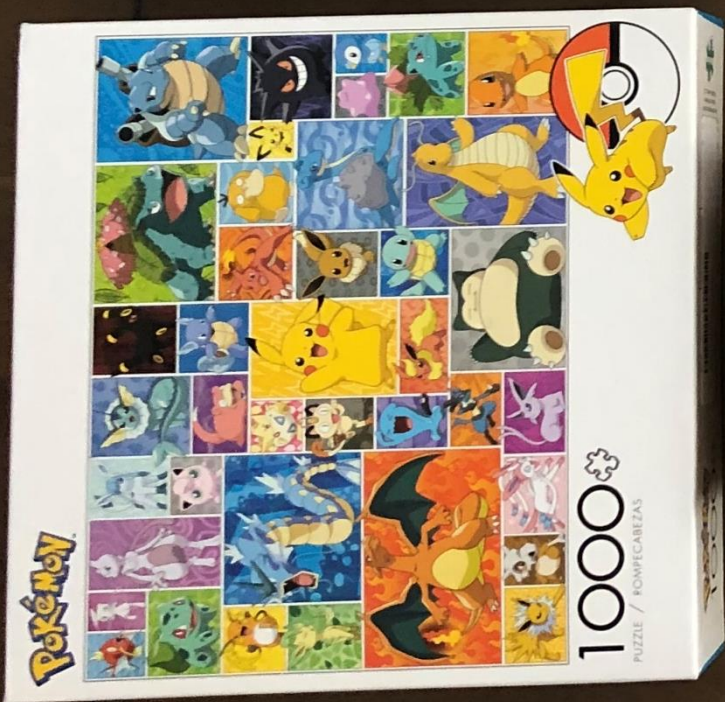
Grade

- **Final: 30% + bonus**
- **Midterm: 20% + bonus**
- **Homework: 44% + bonus**
- **Entrance exam: 5%**
- **Bonus:**
 - Class participation
 - Bonus questions in homework or exams
 - Solving challenge problems

Typical highest score 130-140
Usually 15% can achieve >100 => A+

Grade

- **Entrance exam: 5%**
- **Homework: 44%**
 - Five assignments (biweekly), but each has two deadlines
 - Weights: 6%, 8%, 9%, 9%, 12%
- **Midterm exam: 20%**
 - Feb 13 (Tuesday on Week 6)
- **Final exam: 30%**
 - Mar 20 (Wednesday on Week 11)
- **Bonus:**
 - Class participation
 - Solving certain hard problems in the homework assignments and exams



Homework (45% = 6% + 9% + 9% + 9% + 12%)

- The full mark of each assignment is more than the percentage

- Written assignment
 - Training programming problems
- ~8% for a 9% assignment
- Challenge problems: ~3 problems, 1~1.5% pts each
 - You can work on any one of them to earn the last 1%
 - You can work on multiple of them to earn more bonus points
 - If you totally solve any **entire question** (i.e., get full pts, for programming, you need to pass all tests), you will get 1 **candy**
 - Training programming problems will have an earlier deadline
 - For challenge problems, you can work on them beyond deadline. If you figure out the solutions before the last week of the quarter, you can earn half of the points (but no candies).

Written HW assignments

- Use LaTeX to prepare the solutions
 - <https://www.cs.ucr.edu/~yihans/teaching/218/W21/sample.zip>
- Exam problems will be similar to written HW assignments

Programming assignments

- On CodeForces
- See the guideline here:
 - <https://www.dropbox.com/s/coruzyy10h1jdvf/OJ.pdf?dl=0>
- You can submit multiple times
- You also need to submit a short report for any programming problem in the homework
 - The submission id
 - Description of your algorithm, correctness, time complexity
- Partial points allowed (say you can get 1.5% out of a 2% problem)
- Training programming part has deadline one week earlier!

Grace Days

- **For emergencies**

- Sickness, family issues, internet issues, or other reasons

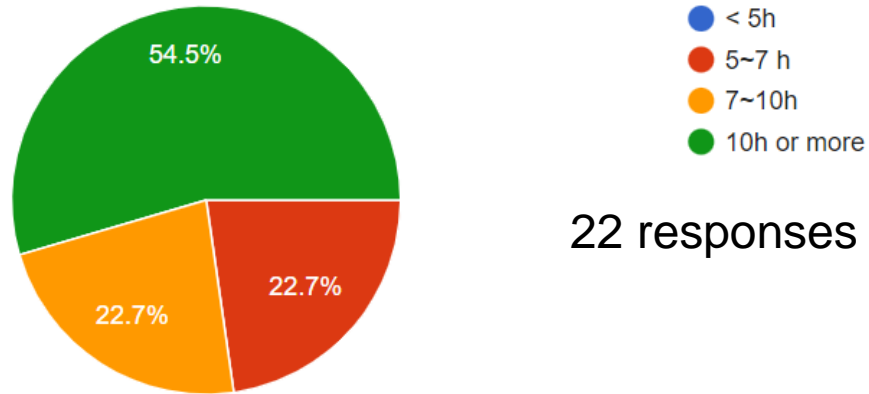
- **4 grace days to use for the **entire** quarter, 2 days maximum per deadline**

- You don't lose points
- The number of grace days used must be an integer (you can't use 0.25 grace days if you submit 6 hours late, it's still 1 grace day)

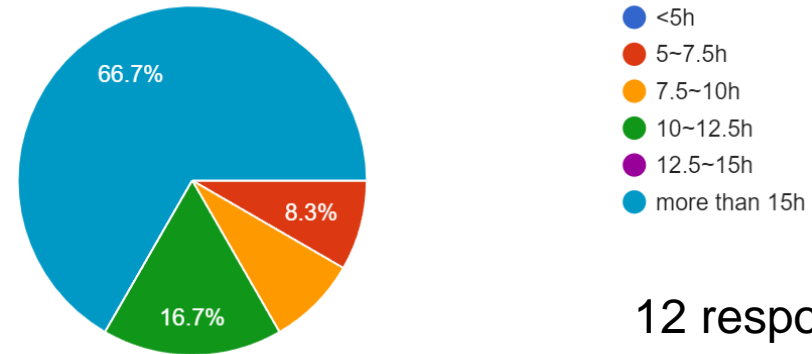
- **Use your grace days *wisely***

- If you got sick, power outage, ..., use your grace days
- **We won't accept other excuses for late submission**, except for very, very severe issues (and you need to provide some evidence for that)
- You can't use all grace days because you want to go to Disneyland, and later you ask for more grace days because you get a fever

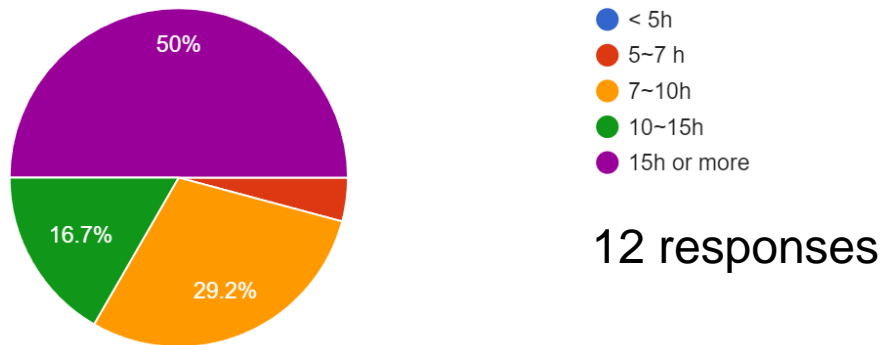
You should expect much work on this course



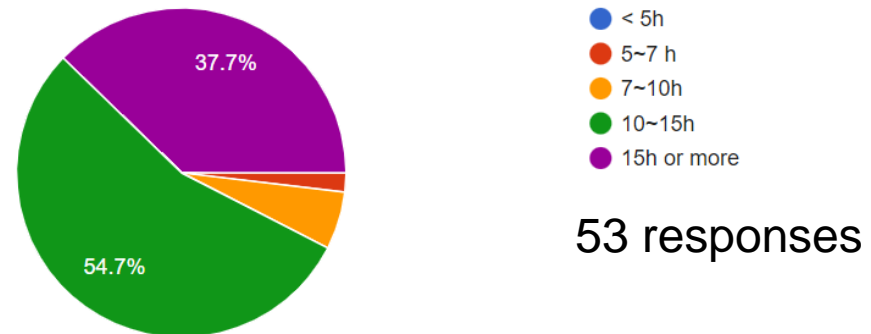
Winter 21



Spring 21



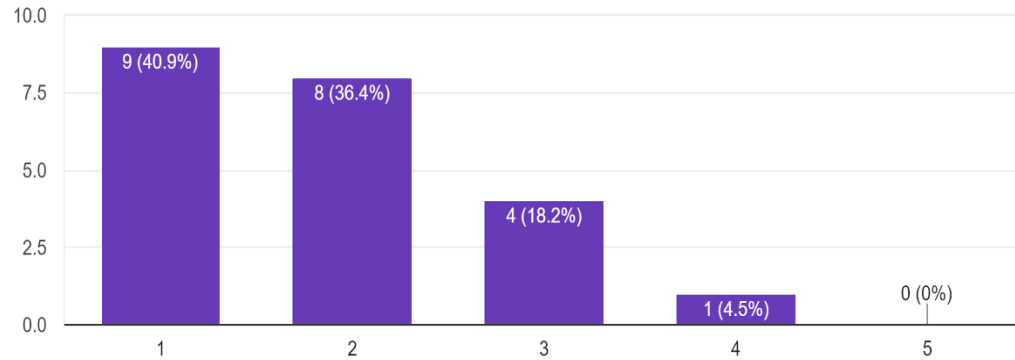
Spring 22



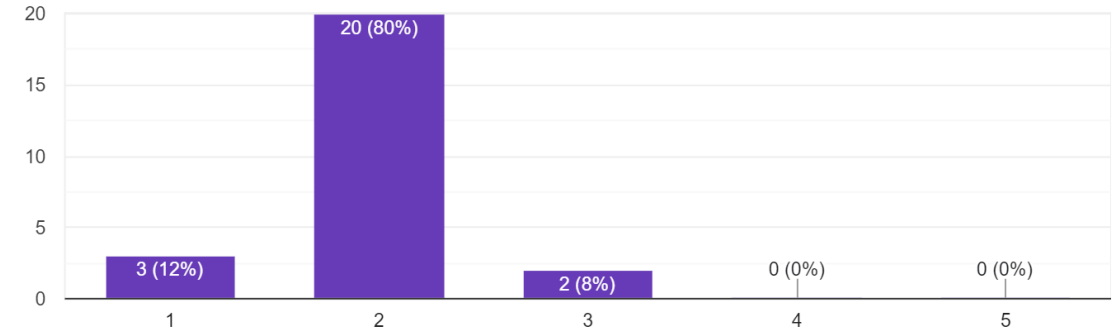
Fall 22

You should expect much work on this course

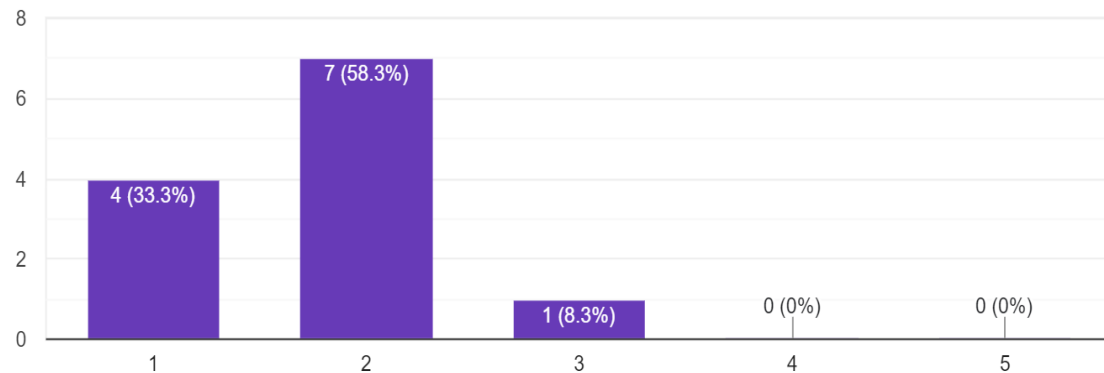
- (How hard is this course: very hard to very easy)



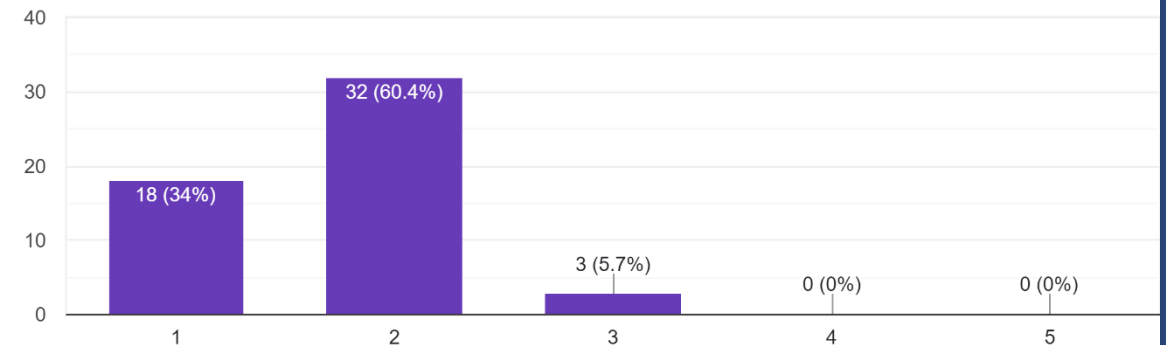
Winter 21



Spring 22



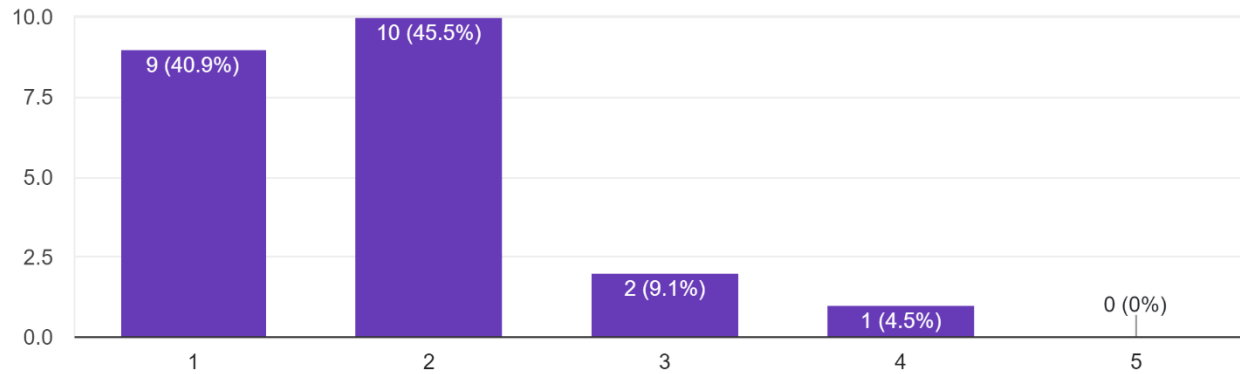
Spring 21



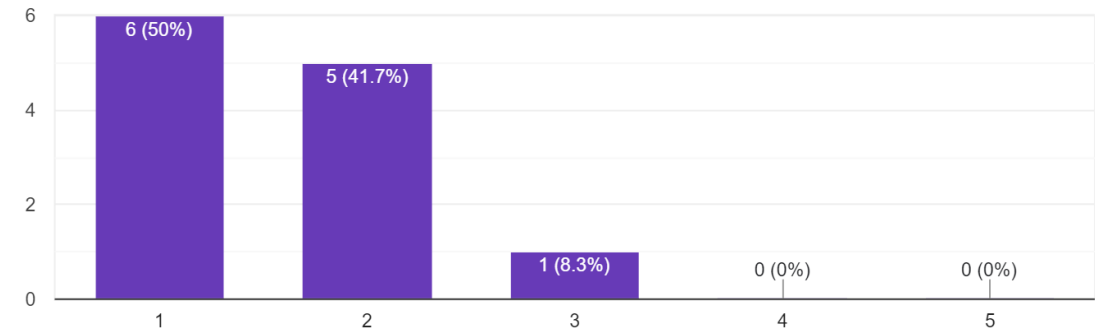
Fall 22

But it should also be an interesting experience

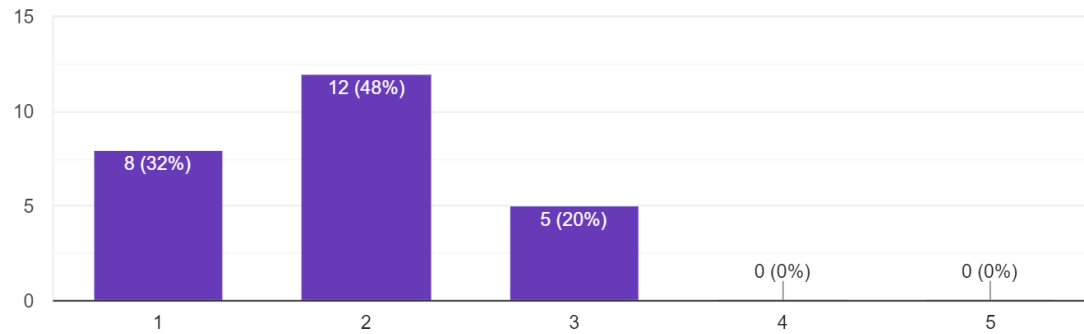
- (How interesting is this course: very interesting to very boring)



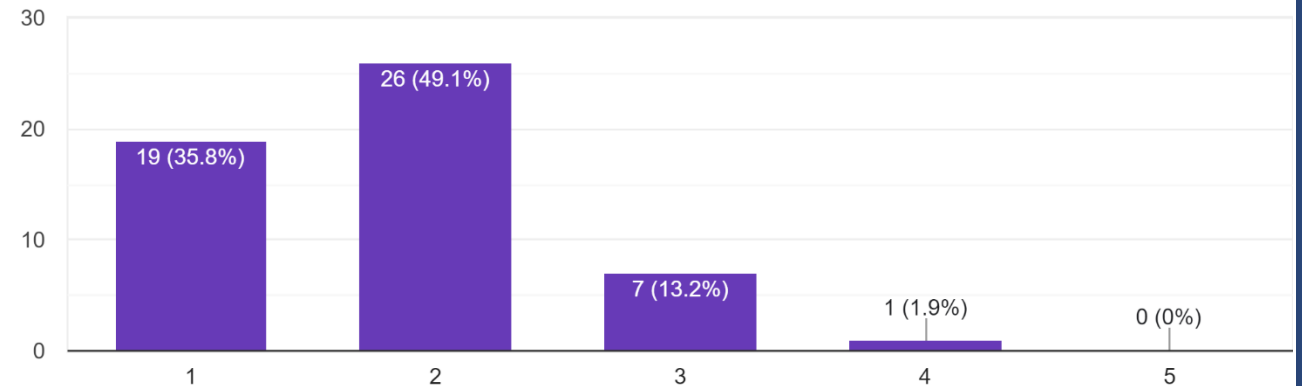
Winter 21



Spring 21



Spring 22



Fall 22

Tentative score-to-grade mapping

- **A+: >100%**
- **A: 90%**
- **A-: 85%**
- **B+/B/B-: 80%/75%/70%**
- **C/D: 65%/60%**
- **F: <60%**

Tentative score-to-grade mapping (from Spring 22)

- **A+: >100%, 16%**
- **A: 90%, 48%**
- **A-: 85%, 58%**
- **B+/B/B-: 80%/75%/70%, 94%**
- **C/D: 65%/60%**
- **F: <60%**

PLAGIARISM WARNING



- **Cheating or plagiarism will NOT be tolerated!!!**
- **Collaboration policy**
 - You can get help from me, textbook (or relevant books), or Internet. **Any source should be cited.**
 - You can discuss with your classmates, but must acknowledge them in your write-up
 - But you **cannot** copy anything. You **cannot** share your solutions/codes with others, and you cannot read others' solutions/codes.
 - When you write down your solution, it must be close-book.
 - All collaborations are “theoretically”, not “practically”
- **See UCR academic integrity for additional information:**
 - <https://conduct.ucr.edu/policies/academic-integrity-policies-and-procedures>

TA Team

- TAs:

- Xiangyun: xding047@ucr.edu
- Zijin: zwan018@ucr.edu

- Readers:

- Andy: ali164@ucr.edu
- Pierre: ptawa004@ucr.edu
- Shrey: ssinh017@ucr.edu
- Vineeth: vsuva001@ucr.edu

Office hours



- More information: <https://campuswire.com/c/G40C1B47A/feed/5>

Searching online or using ChatGPT

- Search engine/AI tools are allowed
- Searching online / find resources / collect information to solve problems is part of your life / future work, you need to get trainings for that
- Using AI tools is part of your life / future work, you need to get training for that, too
- But again, it will show you the useful information. You need to understand it and make it your own knowledge

Doctors: Googling stuff online does not make you a doctor.

Programmers:



Using ChatGPT

- Always think on your own – it's nothing to be happy with if ChatGPT can solve a problem but you cannot
- You take the course (and all other courses) because you want to be better than ChatGPT

**When I
realise ChatGPT
can do my job**



**When I
realise ChatGPT
can do my job**



Picture from <https://cheezburger.com/20737797/artificial-meme-telligence-18-clever-memes-inspired-by-chatgpt>

Entrance Exam

- You have 1 week to work on 6 programming problems [[link](#)]
- 5% in your final score
- They cover basic knowledge in undergraduate algorithm courses:
 - Basic programming and data structures
 - Greedy algorithm
 - Basic dynamic programming
 - Basic graph algorithms
- (most of them only needs ~20 lines of code)
- If you find them already hard / cannot finish them in a week: reconsider if you still want to take the course
- I would not recommend to take the course if you get <5 points

Don't feel frustrated if you cannot solve all problems / you cannot get an A / your classmates are doing better than you

- At graduate level, no one is better than others in everything, since you all have your own specialties and expertise
- You have very different backgrounds

Instead, think more about what you will learn from my CS 218

- Lots of practice on programming
- Problem-solving ability
- Some new algorithms and data structures, some are not covered in textbook but very good to use in practice

But if you feel like it's too hard

- **Maybe you need more background knowledge, and take it later**
 - **Take some prerequisite courses (CS 119L, CS 142)**
 - Join the programming team for more exposure to and practice on algorithms
 - Practice using online resources (Leetcode, CodeForces, other materials online)
- **You take the course for knowledge, not for the credit**
 - **It's important that you absorb the knowledge with full understanding**

	If you work hard and do everything properly	If you cheat
If your background is good	<ul style="list-style-type: none"> You will get lots of practice for your algorithm and programming skills You can solve bonus problems for advanced techniques/skills 	<ul style="list-style-type: none"> You don't benefit from this course and waste the opportunity You are very likely to fail this class
If your background is not so good	<ul style="list-style-type: none"> First take CS 142/119L that are easier courses we set up to prepare you better for CS 218 Take CS 218 in a later quarter You will have a decent chance to excel 	<ul style="list-style-type: none"> Your skills will still be limited after attending the current program You are very likely to fail this class

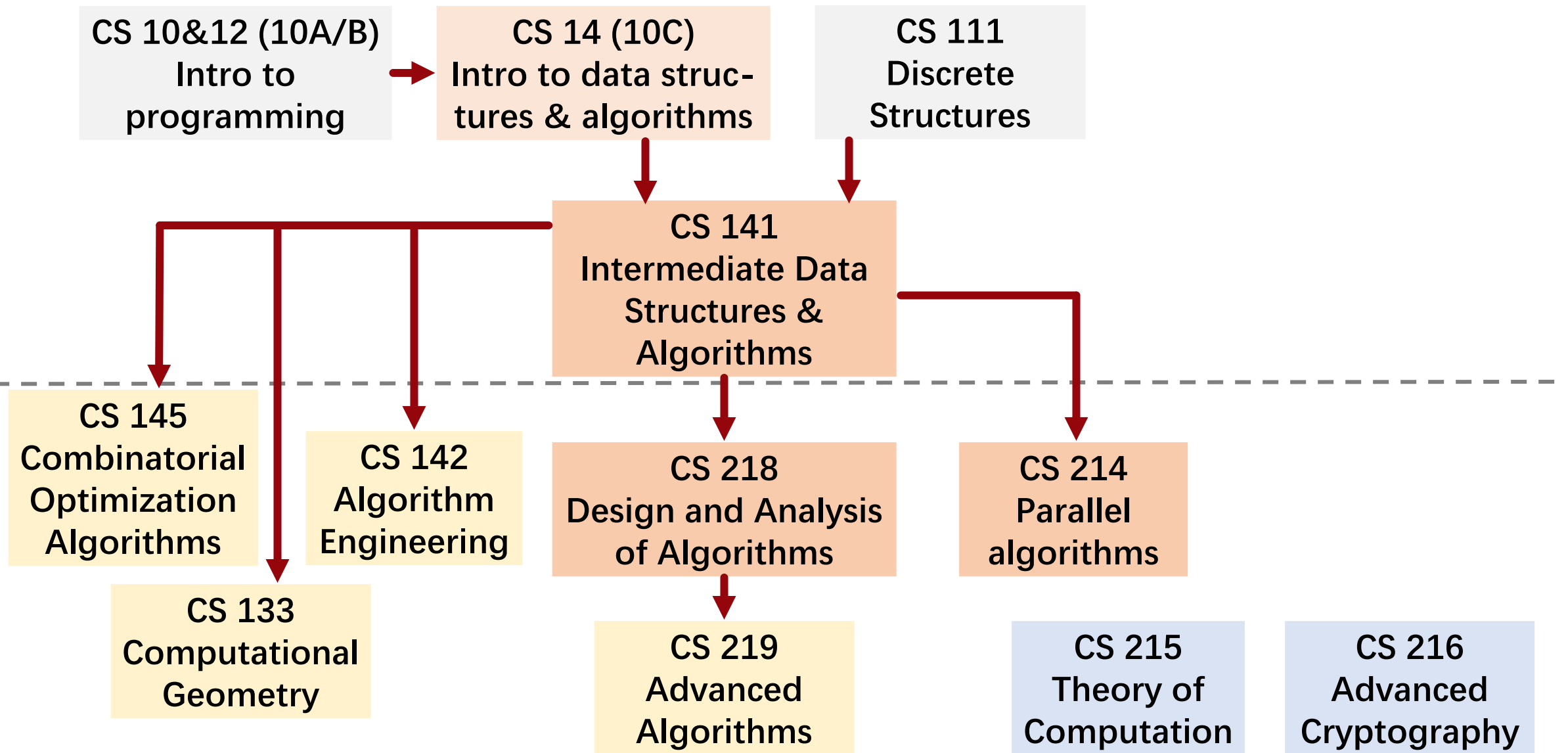
Background you need

- **Asymptotic analysis**
 - $O, \Omega, \Theta, o, \omega$ notations, growth of functions
- **Basic ideas about greedy algorithm**
 - Optimal substructure
 - Prove optimality
- **Basic ideas about dynamic programming**
 - DP recurrence, optimal substructure
- **Basic ideas about graph theory**
 - BFS, DFS, adjacency matrix
 - Shortest path (concept and algorithms), minimum spanning tree (concept and algorithms)

- **Basic ideas about data structures**
 - Priority queue, binary heap
 - Balanced binary tree
 - Hash table
- **Basic ideas about probability, linear algebra, discrete math**
 - Expectation, matrix
- **Basic programming knowledge**
 - Pointers, arrays, ...

Note:

There will be some lectures about reviewing these topics (when necessary).
But they are just “reminders”, not intended to teach you something new if you didn’t know them.



Summary

Algorithms are useful

- **For most code you write, algorithm is the core**
 - If you don't understand algorithms, your code might be ineffective or even incorrect
 - Research in different areas are designing algorithms with different focus
- **For finding jobs and interviews, the understanding of algorithms is decisive**
 - Due to many reasons, **important** positions do require good algorithm background
 - Many of the basic topics / core ideas are covered in the algorithm courses

Algorithms are useful, but also hard

- **You need creativity / intuition, solid background and rich experience in math / programming, and a lot of practice**
 - That's what makes your expertise / degree valuable!
 - That's why you deserve a good job and high salary if you have such abilities!

But if you feel like it's too hard

- **Maybe you need more background knowledge, and take it later**
 - Take some prerequisite courses (CS 141, CS 142)
 - Join the programming team for more exposure to and practice on algorithms
 - Practice using online resources (Leetcode, CodeForces, other materials online)
 - Unlike CS 141, CS 218 is not mandatory
- **You take the course for knowledge, not for the credit**
 - It's important that you absorb the knowledge with full understanding


Analyzing algorithms

Yan Gu

Analyzing algorithms

- Predict how your algorithm performs in practice

- **Criteria**

- Running time  Time complexity
- Space usage
- Cache I/O
- Disk I/O
- Network bandwidth
- Power consumption
- Lines of codes

What are good algorithms?

- Tale about Gauss



One day Gauss's teacher asked his class to add together all the numbers from 1 to 100, assuming that this task would occupy them for quite a while. He was shocked when young Gauss, after a few seconds thought, wrote down the answer 5050. (source: <https://nrich.maths.org/2478>)

Other students:

```
sum = 0;  
for (int i = 1; i <= n; i++)  
    sum += i;
```

$O(n)$ time complexity

Gauss:

$\text{sum} = (1+n)*n/2;$

$O(1)$ time complexity

$$\begin{array}{cccccccccccc} 1 & + & 2 & + & 3 & + & 4 & + & 5 & + & 6 & + & 7 & + & 8 & + & 9 & + & 10 & = & ? \\ \hline & & & & & & & & + & & & & & & & & & & & & \\ & & & & & & & & + & & & & & & & & & & & & \\ & & & & & & & & + & & & & & & & & & & & & \\ & & & & & & & & + & & & & & & & & & & & & \\ & & & & & & & & + & & & & & & & & & & & & \end{array}$$
$$(1+10) + (2+9) + (3+8) + (4+7) + (5+6) = ?$$

What is “time complexity”?

- To estimate the time needed for an algorithm
- To define the “cost” of an algorithm, we first need to define what “costs” time
- **Computational model**

What is a computational model?

- What resource do we have?
- What operations are we allowed to do?
- What is the cost of each operation?

Algorithm A

Cost bound $f(A)$

**Computational
Model**

Cost measure f

What is “time complexity”?

- Count the number of “instructions” in the algorithm
- Random Access Machine (RAM) model
 - We have an arbitrarily large memory
 - We can
 - do arithmetic calculation
 - Read/write to a memory location given the address
 - Every operation takes **unit time**

What is “time complexity”?

- To estimate the time needed for an algorithm
- To define the “cost” of an algorithm, we first need to define what “costs” time
- **Computational model**

What is a computational model?

- What resource do we have?
- What operations are we allowed to do?
- What is the cost of each operation?

Algorithm A

Cost bound $f(A)$

**Computational
Model**

Cost measure f

Further reading

- **Chapter 2 of CLRS, “Getting Started”**

What is “time complexity”?

- Count the number of “instructions” in the algorithm
- Random Access Machine (RAM) model
 - We have an arbitrarily large memory
 - We can
 - do arithmetic calculation
 - Read/write to a memory location given the address
 - Every operation takes **unit time**

```
sum = 0;  
for (int i = 1; i <= n; i++)  
    sum += i;
```

3n+2 operations?

```
sum = (1+n)*n/2;
```

3 operations?

Asymptotic notation

```
sum = 0;  
for (int i = 1; i <= n; i++)  
    sum += i;
```

$3n+2$ operations?

$O(n)$ operations

```
sum = (1+n)*n/2;
```

3 operations?

$O(1)$ operations

What is “time complexity”?

- Random Access Machine (RAM) model
 - Every operation, including memory access, arithmetic operations, etc., takes a **unit time**
- Ignored the “low-level parts”
- We only care about the **order of the cost**, e.g., we omit
 - Lower-order terms
 - Constants
- We care about **the main decisive factors of the input**
 - Usually the input size n
 - Can be other key factors such as #vertices/#edges
 - Parameterized algorithms / complexity

Asymptotic notations

- Big-O: asymptotically **smaller than or equal to** \leq
larger than or equal to \geq
smaller than $<$
larger than $>$
equal to $=$
- n is $O(n)$
- $3n + 2$ is $O(n)$
- $\log n + \sqrt{n} + 4$ is also $O(n)$
- What happens if we want to say other cases?

Analogy to real numbers

Functions	Real numbers
$f(n) = O(g(n))$	$a \leq b$
$f(n) = \Omega(g(n))$	$a \geq b$
$f(n) = \Theta(g(n))$	$a = b$
$f(n) = o(g(n))$	$a < b$
$f(n) = \omega(g(n))$	$a > b$

Popular Classes of Functions

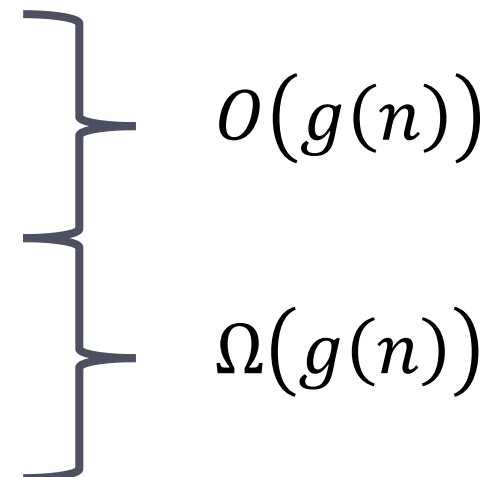
		Example
• Constant:	$f(n) = \Theta(1)$	1, 10, 100000000
• Logarithmic:	$f(n) = \Theta(\log(n))$	$\log n$, $\log n + 3 \log \log n$
• Poly-logarithmic:	$f(n) = O(\log^k n)$	$\log^2 n$, $\log^9 n + 8$
• Sublinear:	$f(n) = o(n)$	$\log n$, \sqrt{n} , $n^{1/5}$
• Linear:	$f(n) = \Theta(n)$	n , $5n + \log n$
• Super-linear:	$f(n) = \omega(n)$	n^3 , $n \log n$
• Quadratic:	$f(n) = \Theta(n^2)$	n^2 , $3n^2 + n$
• Polynomial:	$f(n) = O(n^k)$	$n^3 + 2n^2 + 4$, $4n^5$
• Exponential:	$f(n) = \Theta(k^n)$	2^n

($k \geq 0$ is a constant)

Compare two functions

- $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$

- 0: $f(n) = o(g(n))$
- **Constant** $c > 0$: $f(n) = \Theta(g(n))$
- ∞ : $f(n) = \omega(g(n))$



$O(g(n))$

$\Omega(g(n))$

Commonly-used functions

For large enough n , we have (asymptotically):

$$\begin{array}{ccccccc} c > 0 & & c_1 > 1 & 0 < c_2 < 1 & & c_3 > 1 & c_4 > 1 \\ \bullet \ c < \log \log n < \log n < \log^{c_1} n < n^{c_2} < n < n \log n < n^{c_3} < c_4^n < n^n \end{array}$$

Analyze running time

- **Random Access Machine (RAM) model**
 - Every operation, including memory access, arithmetic operations, etc., takes **unit time**
- **We only care about the **order of the cost** for simplicity, and omit**
 - Constants
 - Lower-order terms
- **We usually consider **worst-case** running time for **general input****
 - In some cases, we also analyze bounds with probabilistic guarantees (e.g., **average** running time for randomized algorithms)

Case study: selection sort

- Find the smallest element in the array
- Move it to the front
- For the rest of elements, find the smallest element and repeat

```
for (int i = 0; i < n; i++) {  
    for (int j = i+1; j < n; j++) {  
        if (a[j] < a[i]) swap(a[i], a[j]);  
    }  
}
```

$$(n - 1) + (n - 2) + \dots + 1 = n \times \frac{n - 1}{2} = \Theta(n^2)$$

Why we don't care about constants & lower-order terms?

- Why we only care about the order (main term)?

Quicksort: $O(n \log n)$ time in expectation

```
void qsort(data_type* A, int l, int r)
{
    int i = l, j = r-1;
    data_type x = A[random_number(l, r)];
    do {
        while (A[i] < x) i++;
        while (A[j] > x) j--;
        if (i <= j) {
            swap(A[i], A[j]);
            i++;
            j--;
        }
    } while (i <= j);

    if (i < r - 1) qsort(A, i, r);
    if (l < j) qsort(A, l, j+1);
}
```

Running time in ms:

	Quicksort	Selection sort
10	0.0098	0.0067
100	0.011	0.027
1,000	0.14	1.683
10,000	1.695	286.425
100,000	19.46	13799.5
1,000,000	226.443	1291300
10,000,000	1706.54	

Within 1s

About 20 min

Selection sort: $O(n^2)$ time:

```
for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
        if (a[i]>a[j]) { swap(A[i], A[j]); }
```

How large a problem you can solve?

- Assume your computer can do each operation in $0.1 \mu\text{s}$ (10^7 operations per second)

Complexity	Max problem size (n)		
	1 second	1 minute	1 hour
$5n$	2,000,000	120,000,000	7,200,000,000
$3n \log_2 n$	172,454	1,003,201	418,961,604
$2n^2$	2,236	17,320	134,164
n^3	215	843	3,301
n^4	56	156	435
2^n	23	29	35

If you are not familiar with these notations

- **Read CLRS Chapter 3: “Growth of Functions”**
 - Definitions of the asymptotic notation
 - How to compare the growth of two functions
 - What are the classic “classes” of functions

After this lecture...

- Join campuswire, gradescope, and codeforces
- Start to work on the entrance exam problems
- Finish course policy test

The next lecture...

- Lower-bound analysis: how hard a problem is?

Have fun!