

CS141: Intermediate Data Structures and Algorithms

Yan Gu

The situation in the last 18 months



Welcome back!



Outline for today's lecture

- **1. Course Logistics**
- **2. Covered Topics**
- **3. Background Test**
- **4. Miscellaneous**

Class information (CS 141 Session I)

- **Classes:** Tuesday & Thursday 2:00–3:20 PM
- **Instructor:** Yan Gu
- **Office hours:** see the calendar on course webpage
- **Email:** ygu@cs.ucr.edu
- **Website:**
<https://www.cs.ucr.edu/~yihans/teaching/141/F21/index.html>

Poll link

- <https://PollEv.com/yangu797>

Or

- Text YANGU797 to 37607 to join the session



People

- **Instructors:**

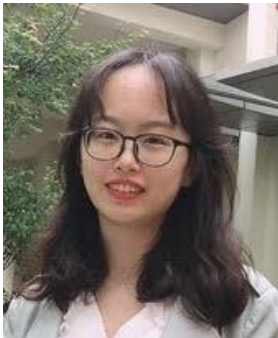


Yan



Yihan

- **TAs:**



Letong



Xiaojun



Yuta



Zachary



Boning



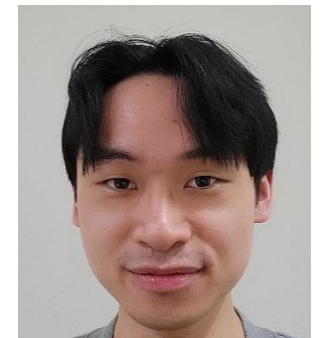
Carolyn



Michelle



Najmeh

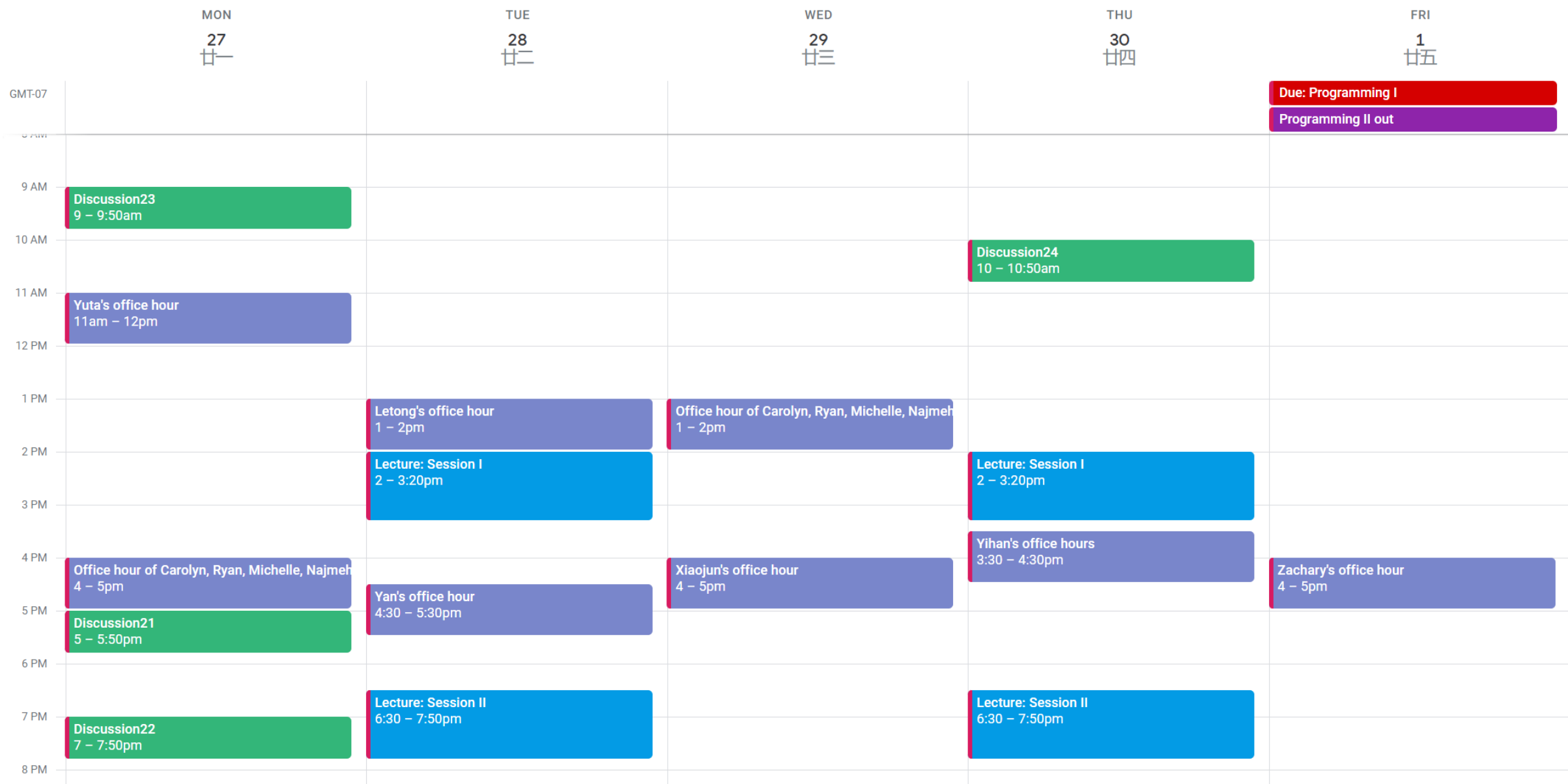


Ryan

The two sessions will align

- **Share the same discussion sessions, homework assignments, office hours, and exams**
- **Both instructors and all TAs answer all questions in any session**
- **Exams are also aligned**
- **Our goal is**
 - Fair evaluation
 - More flexible for your schedule

Office Hours



Office

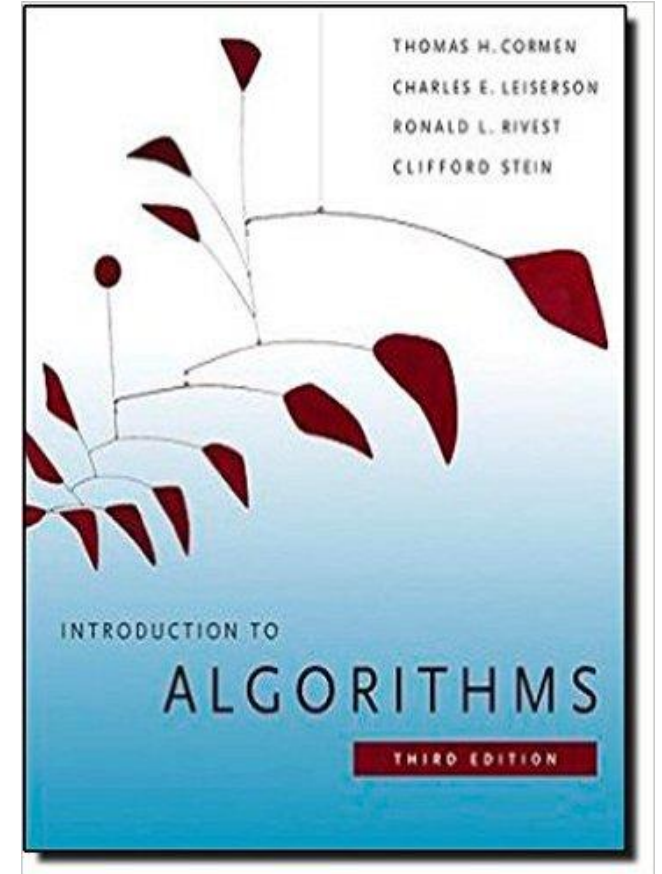


Online tools used

- **Campuswire**: course announcements and Q&A
- **GradeScope**: homework solutions
- **Codeforces**: online judge for programming assignments
- **Slack**: Unofficial discussions for students
- **Online Poll**: <https://PollEv.com/yangu797> or text YANGU797 to 37607 to join the session
- **Please join as soon as possible!**
- **Post your questions and discuss with others!**
- **Homework, lecture slides and videos are in a Dropbox folder, and the link is available on Campuswire**

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**

- Big-O notation, Master Theorem

- **Divide-and-conquer algorithms**

- Divide into smaller size and solve recursively, then combine the results

- **Greedy algorithms**

- Always choose the current best solution

- **Dynamic programming (DP)**

- Record subproblem solutions to get global optimal solution

- **Graph algorithms**

- Connectivity, topological sort
- Minimum spanning tree (MST)
- Single-source shortest path (SSSP)

- **Parallel algorithms**








- Simple concepts and algorithms in parallel computing

- **Road ahead**

- How are the algorithms we learned in 141 used in the CS world
- What are more advanced algorithm courses offered at UCR

CS141 is challenging...

- According to UCR class database on reddit, CS 141 is the **hardest required course** among CS 10, 12, 14, 61, 100, 111, 141, 150, 152, 153, 161, 179

 UCR class difficulty database   			
File Edit View Insert Format Data Tools Add-ons			
  100% 			
<i>fx</i>	=average(9,9,9,8,8,10,8,8,9,8,7,7,10,8,9,7,7,10,10,5)		
	A	B	
1	Class	Average Difficulty	Additional Comments
503	CS141	8.30	The curve was huge at the en

What is a typical job interview process? (Google's software developer as an example, [link](#))

- **Round 1: online assessment (90 minutes)**

- Two data structures and algorithms questions that you have to complete in less than 90 minutes in total
- 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)**

- You will solve data structure and algorithm questions
- You have to write the code and explain the correctness / time complexity to the interviewer
- Additional behavioral questions that are usually not decisive

What is a typical job interview process? (Google's software developer as an example)

- **Round 3: Onsite interviews**

- You'll typically spend a full day at a Google office and do usually four to six interviews
You'll typically get three coding interviews with data structure and algorithm questions, and one system design interviews
- **You are expected to do extremely well in coding interviews.** If you're relatively junior (L4 or below) then the bar will be lower in your system design interviews

- **Most companies are similar**

- **Of course algorithms are also important for doing research in all CS areas**

Why do companies / research positions 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)
- **Doing well in algorithm design indicates that you 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 by your work in the future

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:

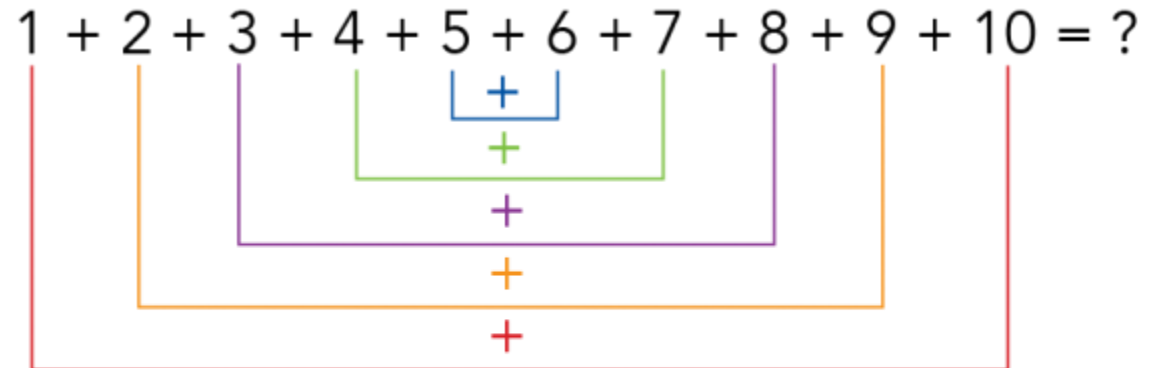
```
sum = 0;  
for (int i = 1; i <= n; i++)  
    sum += i;
```

$O(n)$ time complexity

Gauss:

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

$O(1)$ time complexity



$$(1+10) + (2+9) + (3+8) + (4+7) + (5+6) = ?$$

Algorithm/coding is challenging, but not hard to grasp, once you get some training!

- **Adding 1 to 100 was considered remarkable in Guass's time, but is a simple trick today**
 - You learned this in class!
- **Similar for algorithms**
 - Most of the interview questions are “routine”, which have been covered in classes
 - Real-world problems are much more complicated, but all can be derived from classroom algorithms
- **If you do your work, we are confident that you will get a satisfactory grade, and will one day (if not immediately) find what you learned in this course useful in your career (and fun, hopefully)**
 - You learn through A LOT OF practice: understanding existing algorithms / coding / debugging / showing proofs

More practice: programming assignments

- **Five programming assignments (biweekly)**
- **Each assignment has four problems**
 - Two mandatory, two bonus
- **If you do not program, you will forget everything in a week; if you do, the algorithms are yours**
- **These problems are still much easier than the interview problems, but they are a good start, and Yihan and I have 4 more courses after 141 to help you**
- **Tutorial available on course website (Campuswire post #2), and we will talk about it more on Week 1's discussion**

Which means...

- **This year's CS 141 is even more challenging**
 - You need to prepare 15+ hours on this course if you want to do well
- **But your work will be rewarding**
- **Algorithm and programming are the ultimate goal for a CS program**
 - Once you grasp the knowledge that is hard to learn, you are irreplaceable
 - At UCR, we are here to help you succeed

Which means...

- **This year's CS 141 is even more challenging**
 - You need to prepare 15-20 hours on this course if you want to do well
- **But your work will be rewarding**
- **Algorithm and programming are the ultimate goal for a CS program**
 - Once you master the knowledge that is hard to learn, you are irreplaceable
 - At UCR, we are here to help you succeed
- **It is possible that you do not like programming or to take this challenge**
 - You are free to take CS 141 in other quarters (possibly no programming assignments)
 - However, you take the course for knowledge, not only for the credit

Algorithm courses are not just about new algorithms

- Real world problems have fancy disguise for their algorithmic essence
- 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

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

- Real world problems have fancy disguise for their algorithmic essence
- 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

- **On a social network, how to find all my friends' friends?**
 - First needs to model the social network as a graph
 - Then a BFS for two rounds will work
- **What's the similarity between two DNA sequences?**
 - First model them as strings
 - Then solve it using dynamic programming

Knowledge

**Data
structures**

**Methodo-
logies**

**Certain widely-
used algorithms**

**Analysis (time
complexity, work-span)**

Problem-Solving

**Problem
formalization**

**Mapping to initial solutions,
and further optimizations**

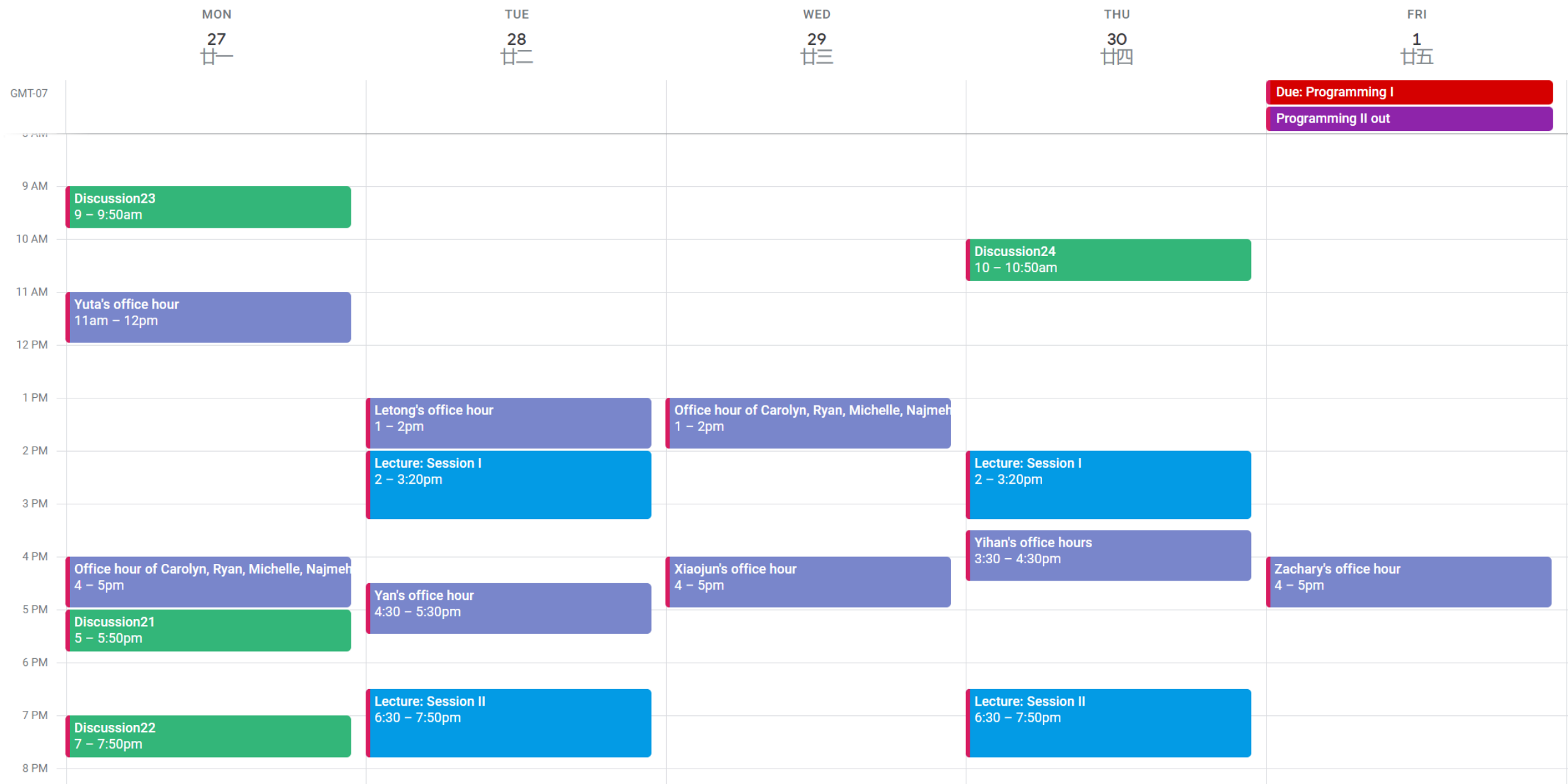
Implementation

**Reasoning and
debugging**

More practice: programming assignments

- **Five programming assignments (biweekly)**
- **Each assignment has four problems**
 - Two mandatory, two bonus
- **If you do not program, you will forget everything in a week; if you do, the algorithms are yours**
- **These problems are still much easier than the interview problems, but they are a good start, and Yihan and I have 4 more courses after 141 to help you**
- **Tutorial available on course website (Campuswire post #2), and we will talk about it more on Week 1's discussion**

Office Hours



Course work

- **Five written assignments (25% = 5 * 5%)**
 - Prepared on Latex (templates provided)
- **Five programming assignments (10% = 5 * 2%)**
- **Background test (1%)**
- **Quizzes (9%)**
- **One midterms (20%): 6:30 PM on Oct 28 (for both sessions)**
- **Final exam (35%)**

- **A lot of bonus credits that will be discussed later**

Grace Days

- **For emergencies**

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

- **5 grace days to use for the **entire** quarter, 2 days maximum each time**

- 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

Urgent situations

- **There are things happened all of a sudden (e.g., a car crash) such that the default 5 grace days are not enough for you**
 - We really don't want it to happen to any of you, and if it happens, just let us know
- **But you use it when you really need it, and you'll need to provide “proof” for that**

PLAGIARISM WARNING



- **Cheating or plagiarism will NOT be tolerated!!!**
- **Collaboration policy (Homework)**
 - You can get help from the instructors, TAs, textbook (or relevant books), internet and your classmates (any source should be cited, including collaborator's name, links to online sources, book title, etc.)
 - You CANNOT:
 - Share/show your written solutions with anyone else (except for course staffs)
 - Read other's solutions
 - Copy anything from other source
 - Get help from others without acknowledging them in your submission, or use any relevant resources (e.g., online article) without citing them.
- **More information available on the course website**
- **See UCR academic integrity for additional information:**
 - <https://conduct.ucr.edu/policies/academic-integrity-policies-and-procedures>

Programming Assignments

- **5 * 2 regular problems**
 - 1 point each in your overall score, partial points allowed
- **5 * 2 bonus problems**
 - 1 bonus point each in your overall score, partial points allowed
 - 1 bonus candy if you pass all the test cases of each problem (more details later)
 - You can work on these problems after deadline, and get 50% bonus points if you finish before final exam, but no candies in this case
- **Most languages supported, but using C++ is recommended**
- **Each problem is tested using 10-20 test cases**
- **Instruction has been sent to your email**

Bonus Candies and Chocolate!

- **You can get bonus “candies” from**
 - Answering questions in class
 - Actively asking/answering questions online
 - Bonus questions in homework, exams, programming problems
 - Participating in UCR Programming Challenge (more details later)
- **You could use it for:**
 - Get real candies in the classroom
 - Earn class participation bonus points (up to 10pts)

UCRPC

- UCR Programming Challenge: ucrpc.net
- <https://youtu.be/Hd1bt0iWIYc>
- 0.5 candy for participating
- 0.5 candy every solved problem (pass all test cases)
- 3 candies for any award earned

Background Test

- **Anonymous vote:** no worries. It's for you to evaluate yourselves and decide if you're ready for the course.
- **[Poll]** Vote here: PollEv.com/yihansun724

Poll link

- <https://PollEv.com/yangu797>

Or

- Text YANGU797 to 37607 to join the session



Background

- **Concepts on data structures/data types**

- Lists, stacks, queues, hash table, binary search tree, priority queue, heaps, ...

- **Basic programming knowledge**

- Pointers, arrays, ...

- **Sorting algorithms**

- Insertion sort, bubble sort, quicksort, merge sort, ...

- **Graphs**

- Basic concepts and definitions
- BFS and DFS

- **Basic math skills**

- Asymptotic notations
- Solving recurrence

Prerequisites:

CS 14 (10C) or equivalent courses

CS 11 and CS 111

Proficiency with C++

Q1. Which of the following sorting algorithm has **worst-case** running time $O(n \log n)$ for n elements?

- A. Quicksort
- B. Mergesort
- C. Selection sort
- D. Bubble sort

Solution: B. Bubble sort and selection sort have worst-case running time $O(n^2)$. Quicksort has expected running time $O(n \log n)$, but in the worst case, it could select the smallest (or largest) element as the pivot all the time, and the running time could be $O(n^2)$.

Q2: $3n + 5 \log n = O(_)$ (O is the big-O notation)

Choose all that apply

- (1). $\log n$
- (2). n
- (3). $n \log n$
- (4). n^2

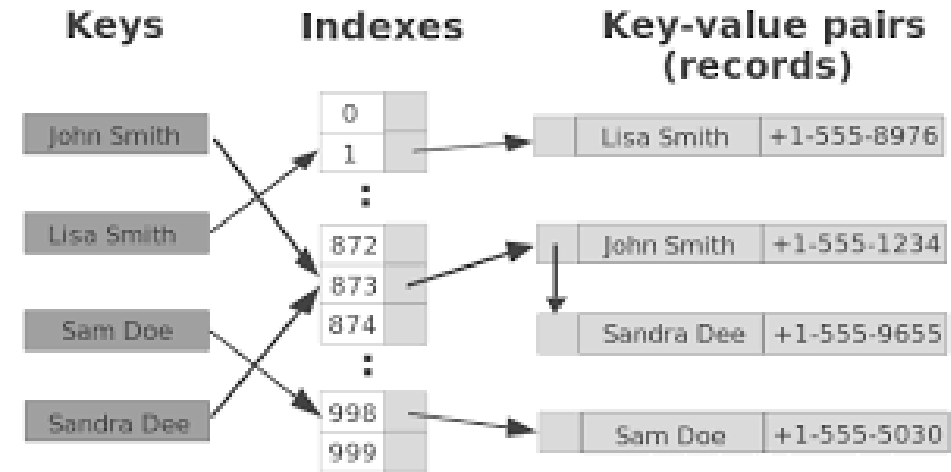
- A. (1)(2)(3)(4)
- B. (2)(3)(4)
- C. (1)(2)
- D. (2)

Solution: B. Big-O means “asymptotically no larger than ($<$ or $=$)”.

$3n + 5 \log n$ asymptotically equals to n , and is smaller than n^2 or $n \log n$

Q3. What is the expected insertion and lookup time for a hashtable of size n (not considering resizing)?

- A. $O(1)$ and $O(1)$
- B. $O(\log n)$ and $O(\log n)$
- C. $O(\log n)$ and $O(1)$
- D. $O(1)$ and $O(\log n)$

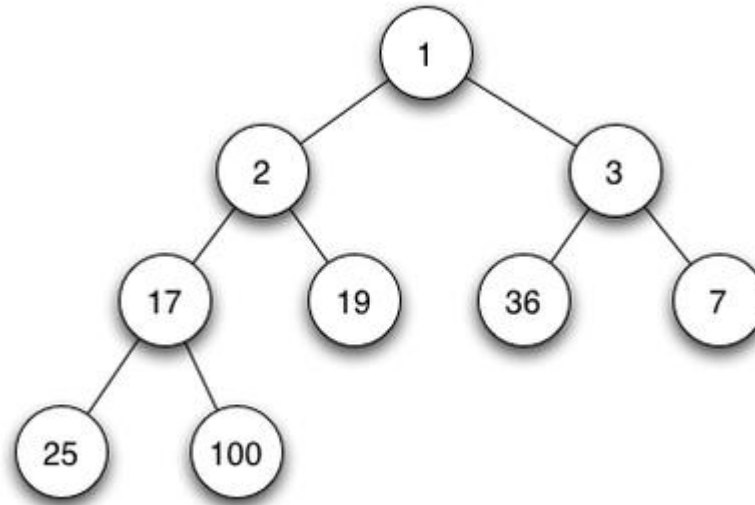


Source: edureka.co

Solution: A. Both will first need to hash the element into a hash value, and directly find the address in the hashtable. The probability of a collision is a constant. Hence if there's a collision, using linear probing or open addressing requires an extra constant time.

Q4. Given a binary heap, to get a sorted list of all elements in the heap needs _____ time.

- A. $O(\log n)$
- B. $O(n)$
- C. $O(n \log n)$
- D. $O(n^2)$

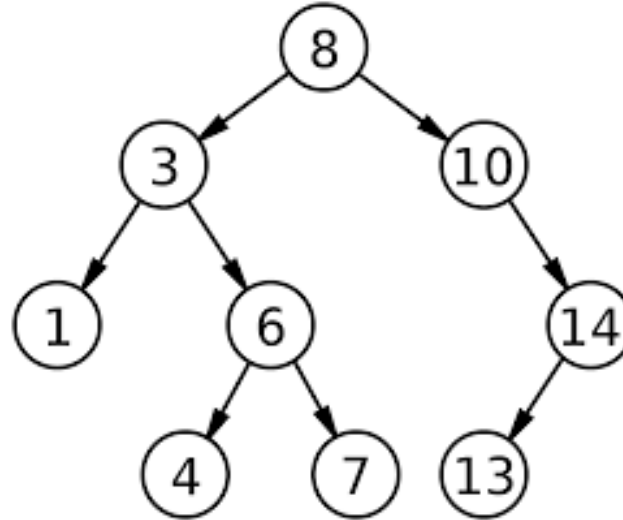


Source: Wikiwand

Solution: C. Binary heap could provide the interface of finding the smallest/largest element in $O(1)$ time, and deleting it in $O(\log n)$ time. It doesn't maintain the total order of all elements in the heap.

Q5. Given a binary search tree (not necessarily balanced), to get a sorted list of all elements in the tree needs _____ time.

- A. $O(\log n)$
- B. $O(n)$
- C. $O(n \log n)$
- D. $O(n^2)$

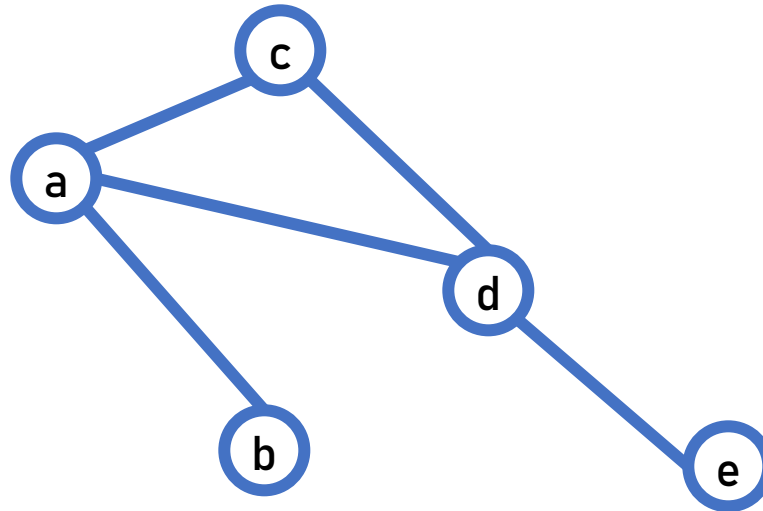


Source: Wikipedia

Solution: B. The binary search tree (BST) has its in-order traversal as the total ordering of all elements in the tree. We just need to traverse the tree and output the in-order. It's easy to use a recursive algorithm. It doesn't require the tree to be balanced.

Q6. Which of the following is a valid DFS sequence of the following graph from a?

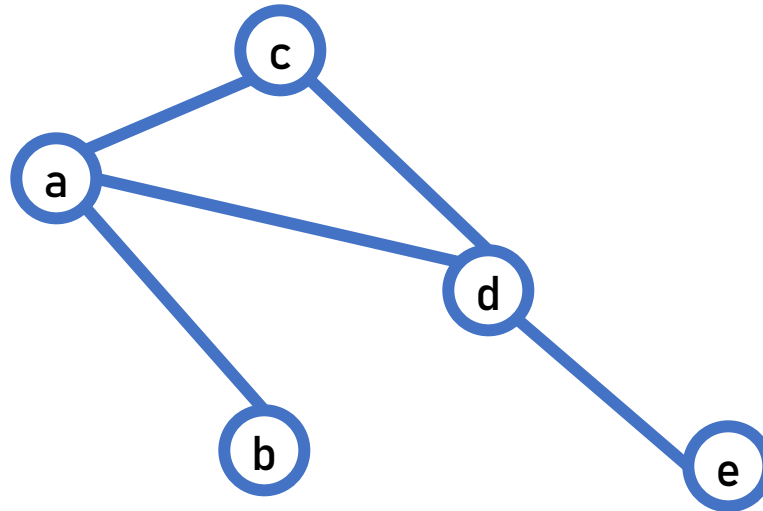
- A. acbde**
- B. abdce**
- C. adcbe**
- D. abced**



Solution: B. For DFS, we follow a path, and keep going until a node that has been visited appears. Then we go back by the same way. Anytime there's a possible alternative diverging path, we choose the new path and continue. The DFS order is not unique. Any sequence that obeying the rule is valid

Q7. Which of the following is a valid BFS sequence of the following graph from a?

- A. abedc**
- B. adebc**
- C. abcde**
- D. acdeb**



Solution: C. For BFS, we first visit all elements that is one-hop from the source, then two-hop, three-hop, ... The ordering of all elements that have the same hop distance from the source doesn't matter.

Q8: If G is a complete graph of size n , how many edges are there in G ?

- A. $\Theta(n)$
- B. $\Theta(n \log n)$
- C. $\Theta(n^2)$
- D. $\Theta(n^3)$

Solution: C. There are $\binom{n}{2}$ pairs of vertices in G , which are $n(n - 1)/2$ edges

Q9. Which of the following is not a good analogy to a stack?

A. A tile of bowls

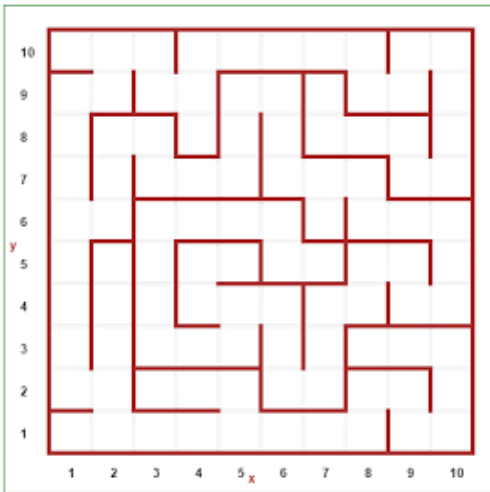
Source: dreamstime.com



B. Solving maze by trying all possible path

Source:

<https://aroberge.blogspot.com/2015/06/generating-mazes.html>



C. A recursive algorithm

D. Waiting for checkin in the airport

```
// Factorial function
int fact(int n) {
    if (n==1) return 1;
    return n*fact(n-1);
}
```



Source: <http://globalperspectives.info>

Solution: D. D is a typical example for FIFO queues.

Background Test

- If you made mistake at more than five of them
- If you don't understand the solutions for more than three of them
- If you don't understand the terms in the problem in any of them

Then you are probably not ready for the course

What can I do if I'm not ready?

- If you haven't taken previous algorithm course (E.g., CS 14), you need to finish it first
- If you have taken previous algorithm courses, but forgot most of the contents
 - Review the content. Most content in CS14 will be used intensively in this course
 - You should be aware that you might need to spend more time on this course – it's much harder than CS 14
 - Sections 6, 10-13 and 22.1-22.3 in CLRS

Other suggestions

- **If you feel like you don't have solid background for the course**
 - Get help from instructors or TAs! We are happy to help!
 - Read the textbook and other books, get help from the internet (e.g., Wikipedia), review contents of previous relevant courses ...
- **Reading slides and the textbook before each lecture could be very helpful**
- **Participate in the discussions online!**
- **You can also get help from the **ULA program** (more info will be available here: <https://ucr-ula.github.io/>)**

Time requirements

- **This is a four-unit CS course. As such, you should expect to spend the following approximate amount of time:**
 - 3 hours/week in lecture
 - 1 hour/week discussion
 - 10 to 15 hours/week doing individual study (readings, homework, preparation for lectures, etc).
- **Please **don't underestimate the time** you will need to spend on this course. These are real time amounts spent by average successful past students.**
- **Again, this is a hard course, please be prepared!**

Recap

- **CS 141 is very hard but useful**
 - The teaching group has 11 people and we are happy to help
 - You are also welcome to discuss with your classmate (not the case before!)
 - We have a new programming track. Have fun!
- **Join Campuswire, gradescope, codeforces, and Slack (optional)**
- **Background Test on GradeScope**
 - 1 pt if you finish it
- **Programming Homework 1 is out**
 - Due Oct 1 (next Friday)
- **Written Homework 1 is out**
 - Due Oct 6