CS 341
Lecture Notes
Winter 2025

Collin Roberts

January 30, 2025

Contents

1 Lecture 01 - Introduction, review of asymptotics 6

1.1 Course Intro . . . 6
1.2 Slide 09 . . . 6
1.3 Slide 10 . . . 6
1.4 Slide 13 Exercise . . . 7
1.5 Slide 14 Exercise . . . 7
1.6 Slide 16 . . . 7
1.7 Slide 17 . . . 7
1.8 Slide 18 . . . 8
1.9 Slide 19 . . . 8
1.10 Slide 21 . . . 8
1.11 Slide 22 . . . 8
1.12 Slide 24 . . . 8
1.13 Notes and Tasks from the Lecture . . . 8

2 Lecture 02 - Solving recurrences 9

2.1 Slide 03 . . . 9
2.2 Slide 07 . . . 10
2.3 Slide 08 . . . 11
2.4 Slide 10 . . . 11
2.5 Slide 11 . . . 12

2.6 Slide 12 12
2.7 Notes and Tasks from the Lecture 13

3 Lecture 03 - Divide and conquer I 13
3.1 Slide 04 13
3.2 Slide 06 14
3.3 Slide 08 14
3.4 Slide 09 15
3.5 Slide 10 15
3.6 Slide 11 16
3.7 Slide 12 16
3.8 Slide 13 16
3.9 Slide 14 17
3.10 Slide 16 18
3.11 Slide 17 18
3.12 Notes and Tasks from the Lecture 18

4 Lecture 04 - Divide and conquer II 19
4.1 Slide 03 19
4.2 Slide 05 20
4.3 Slide 07 20
4.4 Slide 08 20
4.5 Slide 09 20
4.6 Slide 12 21
4.7 Notes and Tasks from the Lecture 21

5 Lecture 05 - Divide and conquer III 22
5.1 Rough Plan, To Be Fleshed Out 22
5.2 Notes and Tasks from the Lecture 22

6 Lecture 06 - Graphs algorithms I - breadth first search 23
6.1 Global 23
6.2 Slide 04 23
6.3 Slide 09 23
6.4 Slide 10 23
6.5 Slide 12 24
6.6 Slide 13 24
6.7 Slide 14 24

6.8 Slide 16 24
6.9 Slide 17 24
6.10 Slide 18 26
6.11 Slide 20 26
6.12 Slide 21 26
6.13 Notes - Éric in F24 26
6.14 Notes and Tasks from the Lecture 27

7 Lecture 07 - Graph algorithms II - depth-first search 28
7.1 Global 28
7.2 Slide 03 28
7.3 Slide 04 28
7.4 Slide 05 29
7.5 Slide 07 29
7.6 Slide 09 30
7.7 Slide 10 30
7.8 Slide 12 31
7.9 Slide 15 31
7.10 Slide 17 31
7.11 Notes - Éric in F24 32
7.12 Tasks 32
7.13 Notes and Tasks from the Lecture 32

8 Lecture 08 - Graph algorithms III - Directed graphs 33
8.1 Global 33
8.2 Slide 04 33
8.3 Slide 06 33
8.4 Slide 07 33
8.5 Slide 08 34
8.6 Slide 09 34
8.7 Slide 10 35

9 Lecture 09 - Graph algorithms IV - Dijkstra's algorithm 35
9.1 Lecture Content 35

10 Lecture 10 - Graph algorithms V - Minimum spanning trees 35
10.1 Bellman-Ford Algorithm 35
10.2 Floyd-Warshall Algorithm 36

10.3 Slide 17 36
10.4 Slide 18 36

11 Lecture 11 - Greedy algorithms I 37

12 Lecture 12 - Greedy algorithms II 37

13 Lecture 13 - Greedy algorithms III 37

14 Lecture 14 - Dynamic Programming I 37
14.1 Slide 02 37
14.2 Slide 04 37
14.3 Slide 10 38
14.4 Slide 11 38
14.5 Slide 12 38
14.6 Slide 14 38
14.7 Slide 15 38
14.8 Slide 16 39

15 Lecture 15 - Dynamic programming II 39

16 Lecture 16 - Dynamic programming III 39

17 Lecture 17 - Dynamic programming IV 39

18 Lecture 18 - Reductions 39

19 Lecture 19 - Reductions, P, NP, co-NP 39

20 Lecture 20 - NP-completeness 40

21 Lecture 21 - NP-completeness 40

22 Lecture 22 - NP-Completeness 41

23 Lecture 23 - NP-Completeness 41

24 Lecture 24 - Misc 41

25 Lecture 25 - Max flow **41**

25.1 Max Flow 41

26 Lecture 26 - Max flow = Min cut **43**

27 Lecture 27 - Applications of Flows and Cuts **43**

1 Lecture 01 - Introduction, review of asymptotics

1.1 Course Intro

1. ISC: Sylvie Davies.
2. **Textbooks**
   (a) CLRS = **Introduction to Algorithms** by Cormen, Leierson, Rivest, Stein
   (b) KT = **Algorithm Design** by Kleinberg, Tardos
   (c) DPV = **Algorithms** by Dasgupta, Papadimitriou, Vazirani

1.2 Slide 09

1. Bullet 3 is the **Limit Rule**, say from CS 240.

1.3 Slide 10

Examples True or False?

1. $2^{n-1} \in \Theta(2^n)$?
   True.
   (a) $2^{n-1} \in O(2^n)$: $c = 1, n_0 = 1$ works.
   (b) $2^{n-1} \in \Omega(2^n)$: $c = \frac{1}{2}, n_0 = 1$ works.
   Alternatively, just apply a Lemma from the CS 341 Background Information.
2. $(n-1)! \in \Theta(n!)$?
   False.
   (a) $(n-1)! \in O(n!)$ holds: $c = 1, n_0 = 1$ works.
   (b) $(n-1)! \in \Omega(n!)$ does not hold: Towards a contradiction, suppose that constants c and n_0 satisfy the definition. Choose an arbitrary n such that $n > n_0$ and $n > \frac{1}{c}$. Then we have

$$\begin{aligned} & c \cdot n! \\ = \ & c \cdot n \cdot (n-1)! \\ > \ & c \cdot \frac{1}{c} \cdot (n-1)! \\ = \ & (n-1)!, \end{aligned}$$

   which is a contradiction.

1.4 Slide 13 Exercise

1. Cost of the Sum Routine:
   (a) The for loop executes n times.
   (b) Each loop iteration requires $\underbrace{O(1)}_{\text{access } A[i]} + \underbrace{O(1)}_{+}$ time.
   (c) So we get $O(n)$ in total.

1.5 Slide 14 Exercise

1. Cost of the Product Routine:
   (a) If multiplication is a basic operation, then this is the same as the sum routine.
   (b) If multiplication is not basic, but must instead be implemented using addition, then it will be $O(n^2)$.

1.6 Slide 16

1. The problem stated here is solved (partially - we only return the sum, not the bounds that created it) in the following ways on the subsequent slides. Note that the run time improves as we go. We will explain each of these techniques, later in the course.
   (a) Brute force: 17-19
   (b) Divide-and-conquer: 20-22
   (c) Dynamic Programming: 23-25
2. We adopt the stated Convention to keep our notation as clean as possible in what follows, and not need to handle empty cases separately.

1.7 Slide 17

1. Per Armin's note, Slide 17 is not actually a solution. this is a useless pseudocode which does nothing. They have potentially seen this in CS240 as is. In the first module of CS240, this was used to show them how they can find the runtime of nested loops. There exists a reference if you look at my lecture plan.

1.8 Slide 18

1. Should all the matrix entries be negative, this algorithm will return 0. This is correct: a sum of 0 is realized by the empty sub-array.
2. The $\Theta(n^3)$ runtime is clear from the structure of the code.

1.9 Slide 19

1. The $\Theta(n^2)$ runtime is clear from the structure of the improved code.

1.10 Slide 21

1. This entire slide is to handle Case 3 from the previous slide; Cases 1 and 2 are trivial. This explains why the right boundary entry are included in `MaximizeLowerHalf` (and, symmetrically, why the left boundary entry would be included in `MaximizeUpperHalf`).

1.11 Slide 22

1. I tried, and failed, to understand Beidl's "bare hands" proof that the Divide-And-Conquer version of Maximum Subarray's worst case run time (same as MergeSort's worst case run time) lies in $\Theta(n \log n)$.
2. Every other source, including CS 341 itself, relies on a recursion tree.
3. From now on, so shall I.

1.12 Slide 24

1. The boxed pseudo-code computes $\overline{M}(n)$.

1.13 Notes and Tasks from the Lecture

1. <u>Notes</u>
   (a) Answer to the Question, is the W25 offering the same as the F24 offering: The topics will be mostly the same. The one exception is that the topic **max-flow/min-cut** was included during F24 but will be omitted during W25.
   (b) <u>Slide 12</u> Explain better why the $\wedge$-rule is less strict than the $\vee$-rule. It is to do with the choice of C:

        i. The first version ($\vee$) makes it more difficult to fix a C, hence it is more strict.
        ii. The second version ($\wedge$) makes it easier to fix a C, hence it is less strict.

2. Tasks
    (a) Get Piazza set up and populdated for the W25 term, if it's not done already (touch base with Sylvie).
    (b) Add a link to the unsecured website, to the LEARN site.
    (c) Fix my screen timeout settings!
    (d) Bring treats to class, from now on!
    (e) Consistently include or exclude the Ericson textbook everywhere (It's mentioned in Armin's slides, but not elsewhere, I think).
    (f) Post to the course website:
        i. Lecture Notes
        ii. CS 341 Background Information
    (g) Turn the Exercises into Clicker Questions, where possible.
    (h) Start L02 with the problem stated on Slide 16, and its many solutions.
    (i) Announce: no tutorials on January 10; first tutorials will be on January 17.
    (j) When we start into dynamic programming later on, recall this last example: it is a great example where, by adding some storage, and remembering work already done, we can effectively cut down our run-time.

2 Lecture 02 - Solving recurrences

2.1 Slide 03

Exercise: Prove that $T^w(n) \leq T(n)$ and $T(n)$ is increasing (an easy induction).
Solution:

1. Proof that $T^w(n) \leq T(n)$, for all $n \geq 1$:
    (a) The proof is by induction on $n \geq 1$.
    (b) Base $n = 1$:
        i. $T^w(1) = d = T(1)$.
    (c) Induction $n > 1$:

i. The induction hypothesis is that $T^w(m) \leq T(m)$, for all $m < n$.

ii. Then

$$\begin{aligned} & T^w(n) \\ \leq\ & T^w\left(\left\lceil\frac{n}{2}\right\rceil\right) + T^w\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + cn \\ \underbrace{\leq}_{I.H.}\ & T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + cn \\ =\ & T(n). \end{aligned}$$

2. Proof that $T(n)$ is increasing, for all $n \geq 1$:
   (a) We show that, for all $n \geq 1$, $T(n+1) > T(n)$.
   (b) The proof is by induction on $n \geq 1$.
   (c) Base $n = 1$:
      i.

$$\begin{aligned} T(1) &= d \\ T(2) &= T(1) + T(1) + cn \\ &= d + d + cn \\ &= 2d + cn \\ &> T(1), \end{aligned}$$

since all quantities are positive.

   (d) Induction $n > 1$:
      i. The induction hypothesis is that $T(m) > T(\ell)$, for all $m > \ell$.
      ii. Then

$$\begin{aligned} & T(n) \\ =\ & T\left(\left\lceil\frac{n}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + cn \\ \underbrace{>}_{I.H.}\ & T\left(\left\lceil\frac{n-1}{2}\right\rceil\right) + T\left(\left\lfloor\frac{n-1}{2}\right\rfloor\right) + c(n-1) \\ =\ & T(n-1). \end{aligned}$$

2.2 Slide 07

1. We do the proof for n a power of b; the result holds for $n \in \mathbb{R}_{\geq 0}$.

2. As on the following slides, $T(1) = d$ (for some $d > 0$) should be part of the definition here too.
3. We should add here that $c > 0$.
4. Checking that the Master Theorem implies that, for MergeSort $T(n) \in \Theta(n \log n)$:
   (a) Let

   $$\begin{aligned} a &= 2 \\ b &= 2 \\ y &= 1 \\ x &= \log_b a, \text{ so that} \\ b^x &= a. \end{aligned}$$

   This gives us that

   $$x = \log_2 2 = 1,$$

   which, applying the Master Theorem, says that

   $$T(n) \in \Theta(n \log n),$$

   as desired.
5. The statement of the Theorem should be clarified, to say that, when $x = y$, we get $T(n) \in \Theta(n^y \log_b n)$ (i.e. state the base explicitly - it depends on b - it is not always 2).

2.3 Slide 08

1. We should add here that $y \in \mathbb{Z}$ and $j \geq 0$.
2. The final size number, namely $\frac{n}{b^j}$, equals 1, because $n = b^j$.
3. Also the number of levels, namely $\log_b n$, equals j, because $\log_b n = \log_b(b^j) = j$.

2.4 Slide 10

1. Suggested revisions for Armin: "is a geometric sequence" $\mapsto$ "involves a geometric series"

2.5 Slide 11

Setup

1. x, y are integers.
2. $a \geq 1$ and $b \geq 2$ are integers, with $a = b^x$, equivalently $x = \log_b a$.
3. The geometric series has first term $a = 1$ and common ratio $r = \frac{a}{b^y} = \frac{b^x}{b^y} = b^{x-y}$.
4. $n = b^j$, equivalently $j = \log_b n$.
5. $a^j = (b^x)^j = (b^j)^x = n^x$.
6. Simplify r^j as much as possible:

$$r^j = \left(\frac{a}{b^y}\right)^j = \frac{a^j}{(b^j)^y} = \frac{n^x}{n^y} = n^{x-y}.$$

Cases of the proof, explained more fully

1. $r < 1$ equivalently $x < y$:
   (a) Per the CS 240 geometric series summary, $\sum_i r^i \in \Theta(1)$.
   (b) This shows that $T(n) \in \Theta(n^y)$ (since $x < y$, the second term dominates the first).
2. $r = 1$ equivalently $x = y$:
   (a) Per the CS 240 geometric series summary, $\sum_i r^i \in \Theta(j) \underbrace{=}_{j=\log_b n} \Theta(\log_b n)$.
   (b) This shows that $T(n) \in \Theta(n^y \log_b n)$ (since $x = y$, the second term dominates the first).
3. $r > 1$ equivalently $x > y$:
   (a) Per the CS 240 geometric series summary, $\sum_i r^i \in \Theta(r^{j-1})$.
   (b) By a Lemma from CS 240 recalled in the Background Information document, this says that $\sum_i r^i \in \Theta(r^j) \underbrace{=}_{above} \Theta(n^{x-y})$.
   (c) This shows that both terms lie in $\Theta(n^x)$, so that by the sum rule, we have $T(n) \in \Theta(n^x)$.

2.6 Slide 12

1. Example: $T(n) = 2T\left(\frac{n}{2}\right) + n, T(1) = 0, n$ a power of 2.

(a) In the notation of the Master Theorem:

$$\begin{aligned}
a &= 2 \\
b &= 2 \\
y &= 1 \\
x &= \log_b a \\
&= \log_2 2 \\
&= 1 \\
x &= y\text{, equivalently} \\
r &= 1\text{, so that} \\
T(n) &\in \Theta(n^y \log_b n) \\
&= \Theta(n \log n).
\end{aligned}$$

2. CR to type up the notes on the guess-and-check approach to solving this example.

2.7 Notes and Tasks from the Lecture

1. Notes
   (a) Our course convention is (as it was in CS 240) that the base of log is 2, unless otherwise specified.
2. Tasks
   (a) Correct the suggested readings on the course website, in the second half of the term.

3 Lecture 03 - Divide and conquer I

3.1 Slide 04

1. **Examples:** Amazon, YouTube, etc where you are a member of a group who are all interested in some stuff.
2. We are not trying to solve the collaborative filtering problem. What we are trying to solve is one of the many tools which might be useful in collaborative filtering.
3. The Padlet question here is just to give them some time to think about the problem and hopefully convinces them what we are doing has some applications.

4. **Answer to Exercise:** Something like "compare the similarity of two rankings" is a good answer.
5. Counting inversion is related to the answer. It is counting the places in two rankings which are different.

3.2 Slide 06

1. Notation:
   - c_ℓ: # of inversions in $A\left[1,\dots,\frac{n}{2}\right]$
   - c_r: # of inversions in $A\left[\frac{n}{2}+1,\dots,n\right]$
   - c_t: # of **transverse** inversions $i \le \frac{n}{2}, j > \frac{n}{2}$.
2. Example: $A = [1,5,2,6,3,8,7,4], n = 8$. Then

$$\begin{array}{rcl} c_\ell &=& 1 - Swap : (2,5) \\ c_r &=& 3 - Swap : (8,7),(8,4),(7,4) \\ c_t &=& 4 - Swap : (6,3),(6,4),(5,3),(5,4) \end{array}$$

   Note, this accounts for all of the 8 inversions we listed earlier, on Slide 05.

3.3 Slide 08

1. **Claim:** $T(n) = 2T\left(\frac{n}{2}\right) + cn\log n$ gives $T(n) \in \Theta(n\log^2 n)$.

   Proof. Sketchy proof that $T(n) \in O(n\log^2 n)$

$$\begin{array}{rcl} T(n) &=& 2T\left(\dfrac{n}{2}\right) + cn\log n \\ &=& 2\left[2T\left(\dfrac{n}{4}\right) + c\left(\dfrac{n}{2}\right)\log\left(\dfrac{n}{2}\right)\right] + cn\log n \\ &=& 4T\left(\dfrac{n}{4}\right) + cn\log\left(\dfrac{n}{2}\right) + cn\log n \\ &\cdots& \\ &=& cn\left[\underbrace{\log 2 + \log 4 + \cdots + \log n}_{\log n \text{ terms}}\right] \\ &\le& cn\left[\underbrace{\log n + \log n + \cdots + \log n}_{\log n \text{ terms}}\right] \\ &=& cn\log^2 n. \end{array}$$

Proof that $T(n) \in \Omega(n \log^2 n)$

This proof follows the technique of substitution outlined in CLRS. Suppose that there exists a constant $d > 0$ and an n_0 such that, for all $n \geq n_0$,

$$d\left(\frac{n}{2}\right)\log^2\left(\frac{n}{2}\right) \leq T\left(\frac{n}{2}\right).$$

Then

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + cn\log n \\
&\geq 2\left[d\left(\frac{n}{2}\right)\log^2\left(\frac{n}{2}\right)\right] + cn\log n \\
&= dn\left(\log n - \log 2\right)^2 + cn\log n \\
&= dn\left(\log n - 1\right)^2 + cn\log n \\
&= dn\left(\log^2 n - 2\log n + 1\right) + cn\log n \\
&= dn\log^2 n + dn(1 - 2\log n) + cn\log n \\
&\geq dn\log^2 n,
\end{aligned}$$

provided $dn(1 - 2\log n) + cn\log n \geq 0$, which will hold provided

$$d \leq \frac{c\log n}{2\log n - 1}.$$

No boundary conditions are given. We could work through the boundary conditions as in CLRS, if needed. $\square$

3.4 Slide 09

1. Recall the notation: c_t denotes the number of **transverse** inversions, with $i \leq \frac{n}{2}, j > \frac{n}{2}$.

3.5 Slide 10

1. The array A in this example is the same as in the previous example. Hence the counts of inversions are also the same.
2. How to Compute c_t:
   (a) Keep a running total.
   (b) Each time we insert $S[i]$ into A, count how many new transverse inversions have been carried out since the previous $S[i]$-insertion.

(c) line 5: j has gotten to big; all right-hand entries are already inserted. Hence the i^{th} entry must be transversely inverted with all of the right hand entries, $\frac{n}{2}$ of them. This gives $c = c + \frac{n}{2}$.
(d) line 6: j is still in bounds; the i^{th} entry must be transversely inverted with the right hand entries inserted to date, $j - \left(\frac{n}{2} + 1\right)$ of them. This gives $c = c + j - \left(\frac{n}{2} + 1\right)$.

3. We showed in Lecture 02 (Slide 07) that Mergesort has $T(n) \in O(n \log n)$. The merge then contributed $d_n \in O(n)$ then; this part is the same here.

3.6 Slide 11

1. No divide and conquer yet. It's coming on the next slide.
2. The first, brute force approach is in $\Theta(n^2)$.

3.7 Slide 12

1. Assume that n is even.
2. F_0 captures the low-order terms of F (and G_0 does the same for G).
3. F_1 captures the high-order terms of F (and G_1 does the same for G).
4. Exercise: Want: $F_0G_1+F_1G_0$, using only one polynomial multiplication, starting from $F_0, F_1, G_0, G_1, F_0G_0, F_1G_1$.

$$\begin{aligned} & (F_0 + F_1)(G_0 + G_1) - F_0G_0 - F_1G_1 \\ = \; & F_0G_0 + F_0G_1 + F_1G_0 + F_1G_1 - F_0G_0 - F_1G_1 \\ = \; & F_0G_1 + F_1G_0, \end{aligned}$$

3.8 Slide 13

1. Check the identity:

$$\begin{aligned} & (F_0 + F_1x^{\frac{n}{2}})(G_0 + G_1x^{\frac{n}{2}}) \\ = \; & F_0G_0 + (F_0G_1 + F_1G_0)x^{\frac{n}{2}} + F_1G_1x^n, \end{aligned}$$

so that we will be done if we can confirm that the middle coefficient equals $(F_0 + F_1)(G_0 + G_1) - F_0G_0 - F_1G_1$. But this is exactly the exercise from the previous slide, no?

2. Analysis: 3 recursive calls, each in size $\frac{n}{2}$:
(a) F_0G_0

   (b) $(F_0 + F_1)(G_0 + G_1)$
   (c) $F_1 G_1$

3. $T(n) = 3T\left(\frac{n}{2}\right) + cn$, analyzed using the Master Theorem:

$$\begin{aligned}
a &= 3 \\
b &= 2 \\
y &= 1 \\
x &= \log_b a \\
&= \log_2 3 \\
&= \frac{\ln 3}{\ln 2} \\
&\approx 1.58 \\
x &> y\text{, so that} \\
r &> 1\text{, and therefore} \\
T(n) &\in \Theta(n^x) \\
&= \Theta(n^{\log_2 3}).
\end{aligned}$$

3.9 Slide 14

1. Gets close to exponent 1, as $k \to \infty$. Check:

$$\begin{aligned}
& \lim_{k\to\infty} \log_k(2k-1) \\
= \quad & \lim_{k\to\infty} \frac{\ln(2k-1)}{\ln k} \\
\underbrace{=}_{L'Hopital} \quad & \lim_{k\to\infty} \frac{\frac{2}{2k-1}}{\frac{1}{k}} \\
= \quad & \lim_{k\to\infty} \frac{2k}{2k-1} \\
= \quad & 1. \checkmark
\end{aligned}$$

2. FFT stands for **Fast Fourier Transforms**.

3.10 Slide 16

1. $T(n)$, analyzed using the Master Theorem:

$$\begin{aligned} a &= 8 \\ b &= 2 \\ y &= 2 \\ x &= \log_b a \\ &= \log_2 8 \\ &= 3 \\ x &> y\text{, so that} \\ r &> 1\text{, and therefore} \\ T(n) &\in \Theta(n^x) \\ &= \Theta(n^3). \end{aligned}$$

3.11 Slide 17

1. $T(n)$, analyzed using the Master Theorem:

$$\begin{aligned} a &= 7 \\ b &= 2 \\ y &= 2 \\ x &= \log_b a \\ &= \log_2 7 \\ &= \frac{\ln 7}{\ln 2} \\ &\approx 2.807 \\ x &> y\text{, so that} \\ r &> 1\text{, and therefore} \\ T(n) &\in \Theta(n^x) \\ &= \Theta(n^{\log_2 7}). \end{aligned}$$

3.12 Notes and Tasks from the Lecture

1. Notes

   (a) Our course standard will be to number our arrays starting from 1, not from 0. We will explicitly state if any particular example deviates from this standard.
   (b) Slide 2: If possible, remove the extraneous page down at the end of the page. Ask Armin.
   (c) Slide 18: Do the results quoted here sit on top of the approach taught in CS 370? Ask Armin/Mark.
   (d) Slide 19: Should this blank page at the end be removed? Ask Armin/Mark.

2. Tasks
   (a) Update the website:
      i. Post office hours, and start holding them this week.
   (b) Slides 7-8: Make it clearer where we are talking about entries, not indices. Where appropriate, change i into $A[i]$. Suggest to Armin to revise the slides accordingly.
   (c) Document, for Exams:
      i. Reference Sheets
      ii. Study Guide (what you will need to memorize, and what you won't)
      iii. Practice Problems, about topics covered by the exam but not yet by any assignment.

4 Lecture 04 - Divide and conquer II

4.1 Slide 03

1. Brute-force: $\Theta(n^2)$.
2. Goal: $\Theta(n \log n)$, using a Divide-and-Conquer approach.
3. See §33.4 in CLRS:
   (a) Divide: Find a vertical line which bisects the point set into L and R, of equal sizes (see the following pictures).
   (b) Conquer: Make two recursive calls, one to handle each of the subsets created above. This returns δ_L and δ_R, both of which are needed as described below.
   (c) Combine: Take the minimum over the three possibilities arising from the setup:
      i. min in L

ii. min in R

iii. min is transverse

4.2 Slide 05

1. $dist(P, R)$ and $dist(Q, L)$ are horizontal distances. In this example, this is where the white band comes from. $\delta = 4$, so the white band covers all points at $dist \leq 4$ from the other side.
2. I suggest the more clear notation $y_P \leq y_Q < y_P + \delta$, instead of $y_P \leq y < y_P + \delta$. We have already restricted to the one point of interest on the left, labelled P at the previous step: it is the only point in the white band created in the previous step.
3. One small confusing point: we were looking for **transverse** pairs just a moment ago, but the constructed rectangle contains points on the left.

4.3 Slide 07

1. A square on the left contains at most one point from L. Reason: If some square contained two points, then the distance separating them would be $\leq \frac{\delta}{2} < \delta$, contradicting the definition of δ.
2. The same argument shows that the square on the right contains at most one point from R.

4.4 Slide 08

1. The reason for $O(n \log n)$ runtime for initialization: sort the points twice, with respect to x and y (c.f. **kd-trees**, in Module 8 of CS 240).
2. Finding the x-median is easy, because we have already sorted the points by x-co-ordinate, when we initialized.
3. Run time: Recursive calls: all to justify the $\Theta(n)$ term in the recursive formula.

4.5 Slide 09

1. We should standardize our notation here. In Lecture 03, our arrays were indexed $1 \ldots n$. Here our arrays are indexed $0 \ldots n - 1$.
2. I also suggest that we create a new line for the heading "Known Results". Talk to Armin.

3. Reason why a randomized algorithm has expected run time in $\Theta(n)$: Refer to CS 240, Module 03, Section on Randomized Algorithms.
4. <u>Assumption:</u> All the $A[i]$s are distinct.

4.6 Slide 12

1. Explanation for $\frac{3n}{10}$:
   (a) $\frac{1}{2}$ of the m_is are $> p$.
   (b) There are $\frac{n}{5}$ m_is.
   (c) So the number of m_is that are $> p$ is $\left(\frac{1}{2}\right)\left(\frac{n}{5}\right) = \frac{n}{10}$.
   (d) Each m_i is the median of a set of size 5; hence there are 3 entries in that set of size 5 which are $\geq m_i$.
   (e) Each of these 3 entries is $\geq m_i > p$, by transitivity.
   (f) Hence the total number of entries which are $> p$ is $3\left(\frac{n}{10}\right) = \frac{3n}{10}$.
2. Why "same thing for $n-i-1$" is correct: swap less / greater throughout: the analysis still works the same way.
3. If time permits, you can (make sure you tell the students this is optional, since it's not part of the W25 slide deck) Slide 13 from Éric's slide deck.
4. This (almost) completes our section on divide-and-conquer.
5. We will actually finish it at the end of Lecture 05, using the remaining time for additional examples and techniques.

4.7 Notes and Tasks from the Lecture

1. <u>Notes</u>
   (a) <u>Slide 04</u>
      i. Explain that the point on the vertical boundary is another choice for P, to be handled at a different time.
   (b) <u>Slide 05</u>
      i. Label the RH point as Q? Ask Armin.
      ii. Why is it enough to
         A. draw the rectangle with P at its bottom, i.e.
         B. only consider points with $y_P \leq y \leq y_P + \delta$?
   (c) <u>Slide 07</u>
      i. Explain why the maximum distance between two points in one of the small squares is $< \delta$: The diagonal distance for a

square with side length $\frac{\delta}{2}$ equals $\left(\frac{\sqrt{2}}{2}\right)\delta < \delta$.

   (d) Slide 08

      i. Explain where the recursion stops: for any set containing ≤ 2 points, no recursive calls are needed - just handle transverse pairs.

   (e) Slide 11

      i. Why 5? So far, it appears arbitrary. The analysis happens to work out.

   (f) Slide 12

      i. **Note:** In this example, **we do not actually divide**: We just create one smaller instance to process at each level of recursion!

2. Tasks

   (a) Run a poll (on Piazza?) to discover whether there is any appetite for virtual office hours.

5 Lecture 05 - Divide and conquer III

5.1 Rough Plan, To Be Fleshed Out

1. Method of Substitution, from CLRS.

   (a) Rigourous proof that $T(n) = 2T\left(\frac{n}{2}\right) + dn\log n$ gives $T(n) \in \Theta(n\log^2 n)$.

2. Method of Change of Variables, from CLRS.
3. Correctness Proof(s), skipped earlier, from CLRS.

5.2 Notes and Tasks from the Lecture

1. Notes

   (a) Question: Is there any improved version of the **Master Theorem**, which can handle recursions like $T(n) = 2T\left(\frac{n}{2}\right) + dn\log n$?

2. Tasks

   (a) Post the clicker questions to the course website.

6 Lecture 06 - Graphs algorithms I - breadth first search

6.1 Global

1. Refer to §20.2 in CLRS.

6.2 Slide 04

1. Both representations of the provided graph are worked out in CLRS.

6.3 Slide 09

1. s is a chosen **source** vertex.
2. Note that we are using the adjacency list representation of the graph here (as stated in the pseudocode).

6.4 Slide 10

1. <u>Explanation of $O(n+m)$ run time for BFS, from CLRS</u>
   (a) Each vertex is enqueued at most once, and hence dequeued at most once.
   (b) The operations of enqueuing and dequeuing take $O(1)$ time, and so the total time devoted to queue operations is $O(n)$.
   (c) Because the procedure scans the adjacency list of each vertex only when the vertex is dequeued, it scans each adjacency list at most once.
   (d) Since the sum of the lengths of all n adjacency lists is $\Theta(m)$, the total time spent in scanning adjacency lists is $O(n+m)$.
   (e) The overhead for initialization is $O(n)$, and thus the total running time of the BFS procedure is $O(n+m)$.
   (f) Thus, breadth-first search runs in time linear in the size of the adjacency-list representation of G.
2. <u>One useful further comment from Armin:</u> $O(nm)$ would mean that we are looking at all edges for each dequeued node, which is not happening here.

6.5 Slide 12

1. The proof is by induction **on** i.
2. "by assumption" $\mapsto$ "by the induction hypothesis"
3. Reason why "$\{v_j, v_i\}$ is in E": From the algorithm, we discovered v_i as a neighbour of v_j.

6.6 Slide 13

1. The proof is by induction **on** i.
2. We should keep our induction approach similar across these proofs. In the previous proof, we used strong induction for $i > 0$. In this proof we use weak induction to get from i to $i + 1$.

6.7 Slide 14

1. Exercise:
   (a) The contrapositive is that $m < n - 1$ implies that G is not connected.
   (b) **Claim:** With $m \geq 1$ edges, at most $m + 1$ vertices can be connected.
   (c) This can be proved by a straightforward induction on $m \geq 1$.
   (d) The claim clearly implies the above contrapositive.

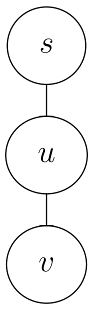
6.8 Slide 16

1. The algorithm set s to be its own parent. This is why we need to exclude the edge from s back to itself in the setup here.
2. Explanation for why T remains a tree when we set $\mathit{parent}[w] \leftarrow v$
   (a) The new graph is connected, because the old graph was.
   (b) We add a vertex, which has only one edge incident with it. There is no possibility of creating a cycle by doing this.
3. For a rigourous proof of Sub-Claim 1, see Lemma 20.3 in CLRS, 4th Edition (Equivalently, Lemma 22.3 in CLRS, 2nd Edition).

6.9 Slide 17

1. Consider these two examples, corresponding with the two cases of the proof that follows:

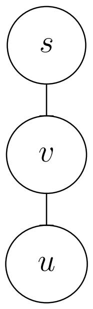
(a)



$$\begin{aligned} level[u] &= 1 \\ level[v] &= 2 \end{aligned}$$

u is dequeued before v.

(b)



$$\begin{aligned} level[u] &= 2 \\ level[v] &= 1 \end{aligned}$$

v is dequeued before u.

2. Improved Proof:
   (a) If we dequeue v before u, then $level[v] \leq level[u]$. which implies $level[v] \leq level[u] + 1$.
   (b) Otherwise we dequeue u before v. Then the parent of v is either u, or was dequeued before u.
      i. If the parent of v is u, then $level[v] = level[u] + 1$. This implies $level[v] \leq level[u] + 1$.

   Otherwise the parent of v was dequeued before u. This implies that $level[v.parent] \leq level[u]$. Also, $level[v] = level[v.parent] + 1$. Putting it all together gives $level[v] = level[v.parent] + 1 \leq level[u] + 1$, as required.

6.10 Slide 18

1. The induction is on i.

6.11 Slide 20

1. We can swap W_1 and W_2, as needed.

6.12 Slide 21

1. Use an integer to keep track of the "colours" that identify each component.
2. Start BFS at a vertex v.
3. When it finishes, all vertices that are reachable from v are colored (i.e., labeled with a number).
4. Loop through all vertices which are still unlabeled and call BFS on those unlabeled vertices to find other components.

6.13 Notes - Éric in F24

1. To start, no edge can connect to itself, so every edge is defined by a pair of **distinct** nodes.
2. Convince yourself that, given a graph, the m mentioned in the definitions is constant.
3. **Good Student Question:** Given a tree, does it matter which node we choose to be the root?
   A: No! Parent-child relationships will change, but no properties that we will need will change, if we make a different choice of root!
4. Convince yourself that Éric's statement that induction and contradiction are really the same thing (to prove POMI is correct, we argue by contradiction), is actually correct!
5. Correctness 1 needs strong induction; Correctness 2 needs only simple induction.
6. To test whether there is a walk from v to w, run BFS from v, then test whether $visited(w) = true$.
7. To test whether a graph is connected, run BFS from anywhere, then test that $m = n - 1$.

8. A given vertex comes out of the queue at most once (it can only go into the queue once; it might never come out).
9. d_v denotes the **degree** of vertex v (i.e. the number of edges emanating from it).
10. **Keeping track of parents and levels**: Now, to test whether a node was visited, we check whether its parent is not NIL.
11. **Graph Convention:** The distance between two nodes which are not connected, is **infinite**.
12. Shortest paths from the BFS tree:
    (a) Let $v_0 \to v_1 \to \cdots \to v_{i-1} \to v_i \to \cdots \to v \to v_k$ be a shortest path $s \to v$.
    (b) $level(v) \leq dist(s, v) = k$.
    (c) For all $i, level(i) \leq i$.
    (d) The level of the parent of v_i is either v_{i-1} or a node that came before v_{i-1}.
    (e) $level(parent(v_i)) \leq level(v_{i-1})$

6.14 Notes and Tasks from the Lecture

1. Notes
    (a) **Q:** In the adjacency list example, does it matter that the linked lists are sorted?
    A: No. I just ordered them in the example to be systematic, and make sure that I didn't miss anything.
    (b) **Q:** Does Sub-Claim 1 imply that the constructed graph has no cycles?
    A: Collin to answer, ASAP.
    (c) All the content of Sub-Claims 1 and 2 happens within the context of the BFS Tree.
    (d) Where to Start L07: L05, Slide 18 (the 2-part claim about the BFS tree, for a graph G).
2. Tasks
    (a) Better explain the statement and proof of Sub-Claim 2: $level[v] \leq level[u] + 1$.

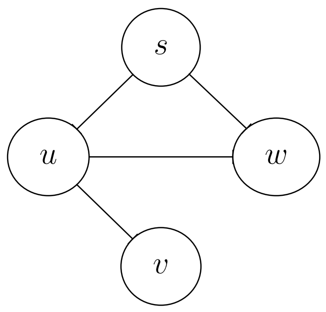
7 Lecture 07 - Graph algorithms II - depth-first search

7.1 Global

1. Refer to §20.3 in CLRS.

7.2 Slide 03

Example: Perform depth-first-search on the graph



Solution:

1. Start from s.
2. Visit s's first child, namely u.
3. Visit u's first child, namely v.
4. We cannot go any deeper: v has no children. We must now back up to u.
5. Visit u's next child, namely w.
6. We cannot go any deeper: w has no children. We must now back up to u.
7. u has no more children. We must now back up to v.
8. v has no more children. We are now done.

7.3 Slide 04

1. This first version of the algorithm simply records which nodes have been visited.
2. As is pointed out in the final bullet, we could easily enhance this to add a parent array, as in BFS.

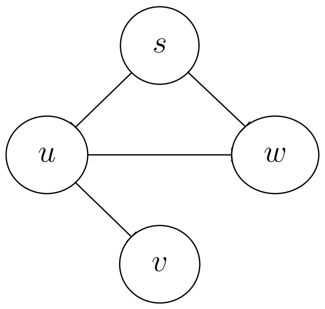
7.4 Slide 05

1. The proof is by induction on $i \geq 0$.
2. Improved Induction Step ($i > 0$):
   (a) Induction Hypothesis: The result is true for some $i < k$, i.e. we visit v_i during $explore(v)$.
   (b) We will show that the result is then true for $i + 1$.
   (c) By our assumption, $explore(v)$ is not yet finished.
   (d) Because there is a path $v = v_0, \ldots, v_i, v_{i+1}, \ldots, v_k = w$, therefore v_{i+1} is a neighbour of v_i.
   (e) 2 cases for when we reach v_{i+1} at step 3 of *explore*:
      i. We have already visited v_{i+1}: the desired result holds.
      ii. We have not already visited v_{i+1}: we visit it now; again the desired result holds.

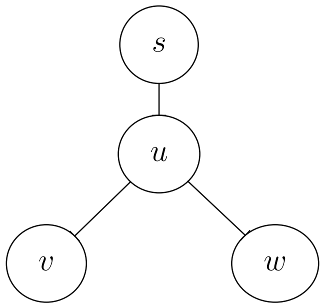
7.5 Slide 07

1. Explanation for Why Run Time is in $\Theta(n + m)$ See pp 566-567 of CLRS!
   (a) The loop on line 2 of DFS takes $\Theta(n)$ time, exclusive of the time to execute the calls to *explore*.
   (b) As we did for breadth-first search, we use aggregate analysis.
   (c) The procedure *explore* is called exactly once for each vertex $v \in V$, since the vertex u on which *explore* is invoked must not be visited yet, and the first thing *explore* does is set vertex u to visited.
   (d) During an execution of *explore*, the loop on line 2 executes $|Adj(v)|$ times.
   (e) Since $\sum_{v \in V} |Adj(v)| \in \Theta(m)$ and *explore* is called once per vertex, the Depth-first search total cost of executing *explore* is $\Theta(n+m)$.
   (f) The running time of DFS is therefore in $\Theta(n + m)$.
2. Example to show DFS need not find the shortest path: Consider the ex-

ample that started the lecture:



(a) Building a tree from this graph starting from s and using DFS yields:



(b) In the original graph, $dist(s, w) = 1$.

(c) But, in the constructed tree, these nodes are at distance of 2 from each other.

7.6 Slide 09

1. Improved Proof:
    (a) Let $\{v, w\}$ be any edge in G.
    (b) W.L.O.G. Suppose we visit v first (If not, just swap).
    (c) Since we visit v before w, therefore at the time we visit v, $\{v, w\}$ is an unvisited path of G.
    (d) Therefore, by the White Path Lemma, w will be visited during $explore(v)$.
    (e) Hence there is a path $v \rightsquigarrow w$.
    (f) Hence v is an ancestor of w.

7.7 Slide 10

1. Explanation for Observation: CLRS Theorem 20.10.

   (a) Let $\{u, v\}$ be an arbitrary edge of G, and suppose without loss of generality that u is visited before v. Then, during $explore(u)$, the search must discover and finish v before it finishes u, since v is on u's adjacency list.
      i. If the first time that the search explores edge $\{u, v\}$, it is in the direction from u to v, then v is undiscovered until that time, for otherwise the search would have explored this edge already in the direction from v to u. Thus, $\{u, v\}$ becomes a tree edge.
      ii. If the search explores $\{u, v\}$ first in the direction from v to u, then $\{u, v\}$ is a back edge, since there must already be a path of tree edges from u to v (by assumption, u was visited before v).

7.8 Slide 12

1. Explanation:
   (a) First bullet of Observation:
      i. first (red) case: the intervals are distinct
      ii. second (blue) case: the interval for v is contained in the interval for u
   (b) If $finish[u] < start[v]$, then there is nothing to prove.
   (c) Otherwise, it must be that $start[v] < finish[u]$. In this case, observing that we pop v before we pop u implies that $finish[v] < finish[u]$.

7.9 Slide 15

1. The first bullet is a proof of the contrapositive of "if s is a cut vertex, then s has > 1 child".
2. The second bullet is a proof of "if s has > 1 child, then s is a cut vertex".

7.10 Slide 17

1. Improved Proof
   (a) Take a child w of v.
   (b) Let T_w be the subtree rooted at w.

   (c) Let also T_v be the subtree rooted at v.
   (d) We have these cases:
      i. <u>For all children w of v, $m(w) < level[v]$</u>
         A. Then there is an edge from T_w to a vertex above v.
         B. Therefore after removing v, T_w remains connected to the root.
         C. Hence v is not a cut vertex.
         D. We have proved "If v does not have a child w with $m(w) \geq level[v]$, then v is not a cut vertex." This is contrapositive of "If v is a cut vertex, then v has a child w with $m(w) \geq level[v]$".
      ii. <u>v has a child with $m(w) \geq level[v]$</u>
         A. I claim that all edges originating from T_w end in T_v (so that v is a cut vertex). **Proof:** Consider any edge originating from $x \in T_w$. Then since $m(w) \geq level[v]$, it follows that this edge from x ends at a level at least $level[v]$. By the Key Property, this edge connects x to one of its ancestors or descendants. This proves the claim.

7.11 Notes - Éric in F24

1. Connected components are the equivalence classes under the equivalence relation: $a \sim b$ if and only if there exists a path from a to b.
2. DFS is BFS, with the queue replaced by a stack.
3. DFS is much more natural to define, using recursion.
4. CLRS colour scheme:
   (a) white - not started visiting yet
   (b) grey - visiting in process (on the stack)
   (c) black - visiting is completed

7.12 Tasks

1. Look up the odd-cycle lemma (I think), from MATH 239?

7.13 Notes and Tasks from the Lecture

1. <u>Notes</u>

   (a) **Q:** Is the worst-case run-time for solving a maze the same for BFS as for DFS?
   A: Intuitively, I think so. I will confirm ASAP.
   (b) **Slide 09**
      i. Up to re-labelling of the first vertex visited, this is correctly stated in the context of G (and not in the constructed DFS tree).
      ii. The proof (I think) ignores the case where $\{v, w\}$ was already visited when constructing the tree. Or does it? Check it again; explain it better.
2. <u>Tasks</u>
   (a) Before the mid-term and final exams, suggest some CLRS exercises, useful for preparation.
   (b) Produce a better graph example that exercises both inductive cases of the proof of the White Path Lemma, at different times.
   (c) Start L08 at **Cut Vertices**.

8 Lecture 08 - Graph algorithms III - Directed graphs

8.1 Global

1. Refer to §20.4 and §20.5 in CLRS.

8.2 Slide 04

1. There can exist edges **in** G connecting the trees T_i.
2. We will call these **cross edges**, starting on the next slide.

8.3 Slide 06

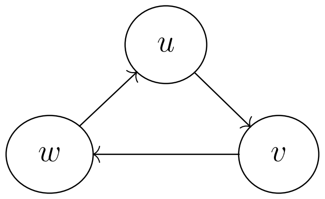
1. See Armin's hand-written notes for brief explanations of the 4 cases. Type these up when time permits.

8.4 Slide 07

1. <u>Reason Why Acyclicity is in $O(n + m)$</u>: Because DFS is.

8.5 Slide 08

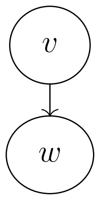
1. A **topological order** is useful to find an order of doing tasks. E.g. a DAG might be used to model dependency relations. E.g. we can model course pre-requisites using a DAG. Then to find an order of taking courses, we find a topological ordering on this graph.
2. The example on the slide is from a recipe.
3. No such order **is possible** if G contains a cycle. item E.g. no topological ordering is possible on



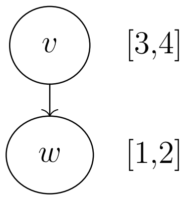
4. A topological order would require $u < v < w < u$.

8.6 Slide 09

1. We want to use DFS to find a topological ordering.
2. We can simply think of start and finish times.
3. Consider a simple DAG:



4. Starting DFS on the bottom vertex gives:



5. Starting DFS on the top vertex gives:

$$\begin{tikzcd}
v \arrow[d] & {[1,4]} \\
w & {[2,3]}
\end{tikzcd}$$

8.7 Slide 10

1. This Proposition affords us a linear time algorithm for finding a topological order, using DFS.

9 Lecture 09 - Graph algorithms IV - Dijkstra's algorithm

9.1 Lecture Content

1. Catch up, ASAP!

10 Lecture 10 - Graph algorithms V - Minimum spanning trees

10.1 Bellman-Ford Algorithm

1. Slide 7: The induction here is on i.
2. Slide 8: I think he said that, in the MIT notes, the corresponding result is named "Safety Lemma", for some reason.
   (a) $\delta(s, v) \leq \delta(s, u) + w(u, v)$ for any edge $(u, v) \in E$.
   (b) If $\delta(s, v) \leq \cdots$ too slow!
3. Slide 10: Summary.
   (a) If no negative cycle is reachable from s, for all $u, v \cdots$ too slow!
   (b) How to derive the contradiction (assuming triangle inequality holds everywhere, I think):
   Sum inequalities of the form $d(v) \leq d(v_i) + w(v_i, u)$.

If there is a negative weight somewhere, then you will get something like $0 \leq -5$ (as in the example), contradiction.
Work out an example for yourself, ASAP!

10.2 Floyd-Warshall Algorithm

1. Slide 13: **SCC** means **strongly connected component**.
2. Slide 16:
   (a) Exercise 1, not proved. He says it is annoying. Check carefully that you are summing the same thing.
   (b) Exercise 2: Supposing we know the P array; then we can easily construct the desired shorted path. He says this one is not difficult, but should be checked.

10.3 Slide 17

1. Exercise: recover the optimum subset.
   (a) Add another two-dimensional indicator array $I[0 \ldots W, 0 \ldots n]$ to the setup.
   (b) As the value is being updated in the O array, Update the corresponding cell of the I array to capture whether that item is included or not.
   (c) Examine the row $I[W, 1 \ldots n]$ of the constructed matrix.
   (d) Also check this with Mark.
2. NP-completeness will be the last topic in our course.
3. Task: Check CLRS for any further explanation about pseudo-polynomial algorithms!

10.4 Slide 18

1. Option 2 is not quite correct yet: it will yield a choice satisfying $\leq$, not necessarily $=$.
2. We would, at a minimum, need to add a step at the end, to check that the maximal choice returned does actually satisfy equality.
1. Stuff.

11 Lecture 11 - Greedy algorithms I

Stuff.

12 Lecture 12 - Greedy algorithms II

Stuff.

13 Lecture 13 - Greedy algorithms III

Stuff.

14 Lecture 14 - Dynamic Programming I

14.1 Slide 02

14.2 Slide 04

1. Explanation for $T(n) = F(n+1) - 1$:
   (a) Proof by (strong) induction on $n \geq 0$.
   (b) Base ($n = 0$):
      i. $T(0) = 0$.
      ii. $F(0+1) - 1 = F(1) - 1 = 1 - 1 = 0\checkmark$
   (c) Base ($n = 1$):
      i. $T(1) = 0$.
      ii. $F(1+1) - 1 = F(2) - 1 = 1 - 1 = 0\checkmark$
   (d) Induction ($n > 1$):
      i. I.H. $T(n-1) = F(n) - 1$ and $T(n-2) = F(n-2) - 1$.
      ii.

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &\underbrace{=}_{I.H.} [F(n) - 1] + [F(n-1) - 1] + 1 \\ &= F(n) + F(n-1) - 1 \\ &\underbrace{=}_{Fibonacci\ definition} F(n+1) - 1. \end{aligned}$$

2. Explanation for $T(n) \in \Theta(\varphi^n)$, where $\varphi = \frac{1+\sqrt{5}}{2}$, the **Golden Ratio**: Refer to Background Information document.

14.3 Slide 10

1. all indices $< n \mapsto$ all indices $t < n$.

14.4 Slide 11

1. increasing end time $\mapsto$ non-decreasing end time.

14.5 Slide 12

1. Definition of $M[j]$: from the two cases mentioned earlier:
    (a) where we exclude interval j, and
    (b) where we include interval j: w_j is from including interval j, and $M[p_j]$ is from all intervals that don't overlap with interval j.
2. Exercise: recover the optimum set, not only $M[n]$, for extra $\Theta(n)$.
    (a) I think we just need to add an indicator array of size n, and indicate in that array for each interval j, whether we have included interval j or not, as we go through the main procedure.
    (b) Then at the end, make one pass through the array to list off which intervals we included.
    (c) Check all of this with Mark, when time permits.

14.6 Slide 14

1. $$S \subset \{1, \ldots, n\} \mapsto S \subseteq \{1, \ldots, n\}.$$
2. While the above is mathematically more correct, the problem will be trivial if we can include everything!

14.7 Slide 15

1. we choose item n or not $\mapsto$ we include item n, or we don't
2. "choose" $\mapsto$ "include" through the rest of the bullets also.
3. Indent the list of two items, of which we take the max.

14.8 Slide 16

1. The array O is two-dimensional!
2. <u>Explanation for why the run time is in $\Theta(nW)$:</u>
   (a) The outer loop runs n times.
   (b) The inner loop runs W times.
   (c) The work inside the inner loop is all in $\Theta(1)$.

15 Lecture 15 - Dynamic programming II

Stuff.

16 Lecture 16 - Dynamic programming III

Stuff.

17 Lecture 17 - Dynamic programming IV

Stuff.

18 Lecture 18 - Reductions

1. Stuff.

19 Lecture 19 - Reductions, P, NP, co-NP

Still from Lecture 19, I think

1. The definition of a **clique** in a graph does not make sense to me yet. Thnink about it some more.
2. All reductions below are polynomial time.
3. <u>IS *leq* Clique $\leq$ IS</u> Too slow! Second reduction is same as first, I think.
4. <u>IS *leq* VC $\leq$ IS</u> first reduction: $Q \mapsto G$; $K \mapsto n \setminus K$.

Now from Lecture 20, I think

1. <u>Slide 4</u> Correct "conjonctive" to "conjunctive"!

20 Lecture 20 - NP-completeness

Still from Lecture 20, I think

1. Global: To verify a decision problem lies in NP: it must have a polynomial size certificate and a polynomial time verification algorithm.
2. Slide 16: Stuff.

Now from Lecture 21, I think

1. Slide 9:
   (a) certification: are at least 2 y_is 1?
   (b) Darn! Too slow!
   (c) Hey, he mentioned that students see Turing machines in CS 245!
2. Given an instance $x \in PROB \in NP$, build circuit from $B(x, \cdot)$. Input to the circuit = certificate, y.
3. He waved his hands over constructing the circuit. Still, polynomial size.
4. Slide 12:
   (a) To prove $3SAT \leq Indepenent - Set$.
   (b) We know $I.S. \leq Clique$, $I.S. \leq Vertex{-}Cover$, so $I.S.$, $Clique$, $Vertex{-}Cover$ are all NP-complete.
   (c) Exercise: explain (English, pseudo-code not required) why the provided construction is polynomial time.

21 Lecture 21 - NP-completeness

Still from Lecture 21.

1. Slide 18:
   (a) input size = $\underbrace{\ell}_{\#\ of\ clauses} \cdot \underbrace{\log n}_{\#\ of\ bits\ needed\ to\ write\ indices\ in \{1,\ldots n\}}$
   (b) $x_{1000} \vee x_{1001} \vee \overline{x_{1000}}$
   (c) (becomes)
   $x_1 \vee x_2 \vee \overline{x_1}$

Now from Lecture 22

1. Slide 4:
   (a) We all agree to quietly forget the Euclidean Travelling Salesman Problem.
2. Slide 6:
   (a) $k = 0$: if and only if there exist no vertices. Silly, but correct.

22 Lecture 22 - NP-Completeness

Stuff.

23 Lecture 23 - NP-Completeness

Still from Lecture 22.

1. Slide 17:
   (a) Per variable, $2s$ tips $\rightarrow 2ns$ total.
   (b) ns covered in pink
   (c) s covered (at least) by clauses
   (d) So we get $ns - s$ tips $(= 4)$ uncovered ???

Now from Lecture 23

1. Slide 4:
   (a) Stuff.
2. Slide 6:
   (a) Stuff.

24 Lecture 24 - Misc

Still from Lecture 23.

1. Slide 4:
   (a) $\log t$, because we express the bound on the run-time, in binary form.
2. Slide 7:
   (a) The Halting Problem is NP-hard, but not in NP.

25 Lecture 25 - Max flow

25.1 Max Flow

1. Slide 5:
   (a) The edge in the first sum is named e.
2. Slide 6:
   (a) Not clearly a flow problem yet, but it is "close enough".

    (b) See the graph at the bottom of the slide, where the labels indicate capacities.

3. Slide 7:
    (a) The algorithm might not be polynomial. It might only be pseudo-polynomial.
4. Slide 8:
    (a) Modify the provided flow, to increase its value from 3 to 4.
5. Slide 10:
    (a) Explanation for why we want a **minimal** value of all capacities on γ in G_f:
        i. It is the most conservative choice, hence the least likely to violate any flow constraints after we have modified the graph as in the algorithm.
    (b) Why the new flow is improved: As on the slide itself!
6. Slide 11: Why we still have a flow afterwards: Let f be the new flow.
    (a) For all integers $0 \leq f'(e) \leq c(e)$
    (b) Suppose e is blue: $f'(e) = f(e) + x$.
    (c) Hence $f'(e) \geq 0$ because $x \geq 0$.
    (d) Also, $\underbrace{x}_{min\ capacity} \leq \underbrace{c(e) - f(e)}_{capacity\ of\ e\ in\ G_f}$ so $\underbrace{f(e) + x}_{f'(e)} \leq c(e)$.
    (e) Now, 1 of 4 possible cases: blue-in, red-out, I think
    red edge got decreased by x.
    blue edge got increased by x.
    Things work out in this case.
    (f) The other 3 cases are similar
    (g) Now suppose e is red? Maybe I missed this case.
    (h) Check these details, ASAP!
7. Slide 13
    (a) After 200000 steps, we will terminate and return the max flow.
    (b) I think that he said this is true polynomial time.
    (c) We can do better at choosing our augmented graph; he did not explain how.
8. Slide 14
    (a) Check that $r^2 = 1 - r$.
    (b) This implies (multiplying through by r^i) $r^{i+2} = r^i - r^{i+1}$.
9. Slide 18
    (a) No need to know how the example was created.

(b) **Moral:** If we stick to integers, the algorithm will terminate, finding the maximum flow.
(c) Next Lecture: proof of correctness.

26 Lecture 26 - Max flow = Min cut

Stuff.

27 Lecture 27 - Applications of Flows and Cuts

1. General I think that he said he proved in Lecture 17 that max-flow equals min-cut. Check it!
2. Slide 4
   (a) I think we need a bit more care in the "loop" case: What if we loop back to the source???
   (b) I think the induction step is is (quietly) a proof by contradiction. Check it!
   (c) Recall that the **value** of a flow is the total amount leaving the source node.
   (d) Check all of this, and generate questions for Éric, ASAP.
   (e) Stuff.
3. Slide 11
   (a) Stuff.