CS3230 Tutorial 4

Divide & Conquer, Lower Bounds, Average-Case Analysis

(AY 25/26 Semester 2)

February 9, 2026

(Prepared by Benson and Hua Jun)

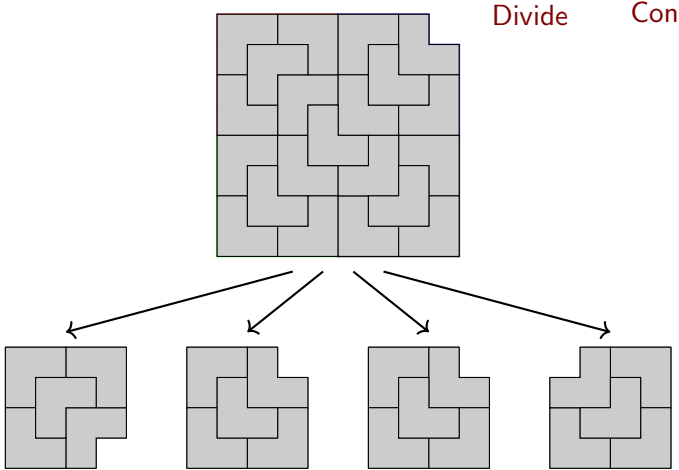
Admin Info

- Let's do F2F tutorial next week!
  - Attendance will *not* be taken. Session will be recorded.
  - Hopefully attendance ≥ 5 :p
- Midterm Tips:
  - "It is a good idea for students to review the lectures, tutorials, and assignments carefully, as the exam questions are often extensions or applications of techniques from there."
  - You should be able to derive things ground-up (especially examples shown in lectures).
  - Manage your time carefully, slow solutions $>$ incorrect solutions/pure smoke.

Contents

Re-recap: Divide & Conquer

Example: Tiling L-pieces



Q1-4. Multiplying Polynomials

Let $A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$.
Let $B(x) = b_0 + b_1x + b_2x^2 + \cdots + b_nx^n$.
Let $C(x) = A(x) \times B(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{2n}x^{2n}$.
In the above, a_i, b_i, c_i are integers.

Assume addition and multiplication of two integers take $O(1)$ time.
It is easy to calculate c_i as $\sum a_j b_{i-j}$ where the sum over j such that both i and $i-j$ are between 0 and n (both inclusive).
Show how to compute the coefficients c_i of $C(x)$ in time $O(n^{\log_2 3})$.

Q1-4. Multiplying Polynomials

$$
\begin{array}{rr}
 & 3x^2 + 5x + 2 \\
\times & 2x^2 + 2x + 1 \\
\hline
 & 6x^4 + 16x^3 + 17x^2 + 9x + 2
\end{array}
$$

$$c_i = \sum a_j b_{i-j}$$

$c_0 = a_0 b_0$
$c_1 = a_0 b_1 + a_1 b_0$
$c_2 = a_0 b_2 + a_1 b_1 + a_2 b_0$
$c_3 = a_1 b_2 + a_2 b_1$
$c_4 = a_2 b_2$

So, $C(10) = 6 * 10^4 + 16 * 10^3 + 17 * 10^2 + 9 * 10 + 2 = 77792$

Q1-4. Multiplying Polynomials

(Let's assume $n = 2^k$. And we're using A instead of $A(x)$ for better readability.)
Let $A = x^{n/2}A_1 + A_2$.
Let $B = x^{n/2}B_1 + B_2$.
$C = A \cdot B = (A_1B_1)x^n + (A_1B_2 + A_2B_1)x^{n/2} + (A_2B_2)$

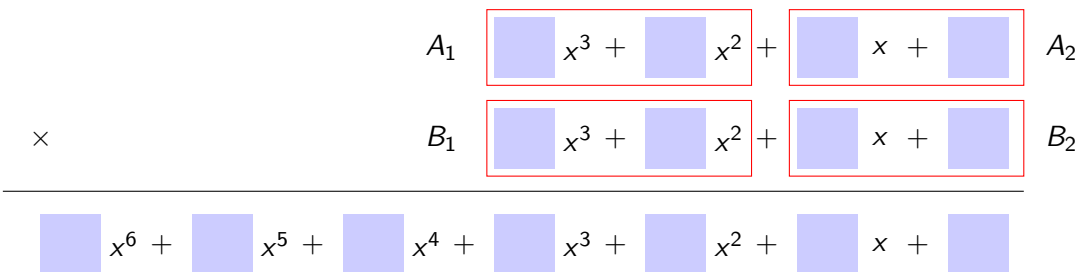
- We have to compute the coefficients of A_1B_1, A_1B_2, A_2B_1 and A_2B_2.
- Then, we can compute C from these coefficients.
- Since A_1, A_2, B_1, B_2 are polynomials with degree $n/2$, the recurrence is:

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n)$$

  (note that computing C from A_1B_1, A_1B_2, A_2B_1 and A_2B_2 takes $\Theta(n)$ time)
- By Master Theorem case 1, $T(n) = \Theta(n^2)$.

Q1-4. Multiplying Polynomials

The first Divide & Conquer idea:



Q1-4. Multiplying Polynomials

Let $A = x^{n/2}A_1 + A_2$.
Let $B = x^{n/2}B_1 + B_2$.
$C = A \cdot B = (A_1B_1)x^n + (A_1B_2 + A_2B_1)x^{n/2} + (A_2B_2)$

- Observe that $(A_1 + A_2) \cdot (B_1 + B_2) = A_1B_1 + A_2B_2 + (A_1B_2 + A_2B_1)$.
- Therefore, we can compute the coefficients of A_1B_1, A_2B_2 and $(A_1 + A_2) \cdot (B_1 + B_2)$. Retrieve the value of $A_1B_2 + A_2B_1$ from there.
- This reduces the recurrence to

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

- By Master Theorem case 1, $T(n) = \Theta(n^{\log_2 3})$.

Q1-4. Multiplying Polynomials

Bonus: There exists an $O(n \log n)$ algorithm for multiplying polynomials - look up Fast Fourier Transform.

Q1-4. Multiplying Polynomials

Optional Exercise: Implement this Divide & Conquer algorithm at https://onecompiler.com/embed/challenges/424f67ckw?theme=dark.

Recap: Decision Trees

Problem: Sorting $n = 3$ distinct numbers by comparing two numbers each time.

Leaves: Output / decision for the input

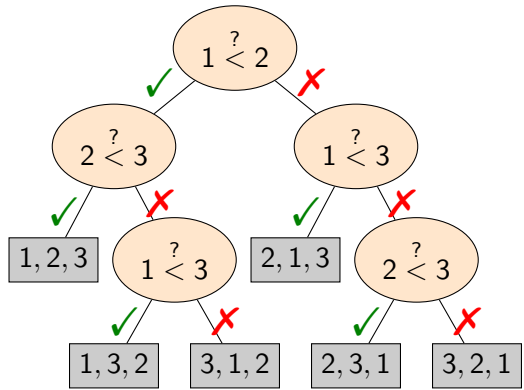
- What are the decisions?
- How many different decisions?

(Internal) Nodes / Vertices: A comparison

Branches: Outcome of comparison

- What are the outcomes?
- What will be the implication?

Worst Case (number of comparisons) = Height of Tree



Recap: Decision Trees

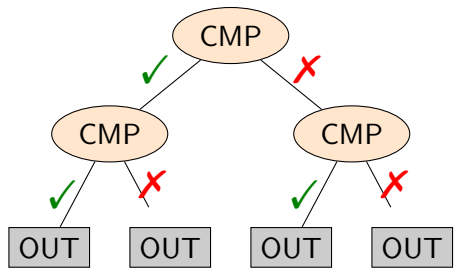
Lower Bound Analysis: To prove that 3 comparisons, is the lower bound for **any** solution that can solve the problem.

Assume we have a solution that can solve the problem in at most 2 comparisons.

Thus, height (H) of decision tree $= 2$.

Since, each comparison can have two results, each internal node can branch twice (B).

The number of leaves in the tree must be $\leq B^H = 2^2 = 4$.



However, we know that there are $3! = 6$ permutations, which is > 4, thus with just 2 comparisons, we are unable to account for all outcomes.

More Lower Bound Proofs: Adversary Argument

Extra Slide

Example: The lower bound for any comparison-based algorithm to determine the index of the maximum element of an array is $\Omega(n)$.
Proof. Suppose the algorithm makes less than $\frac{n}{2}$ comparisons. Then, there exists an element $A[i]$ not involved in any comparisons. Construct an adversary such that:

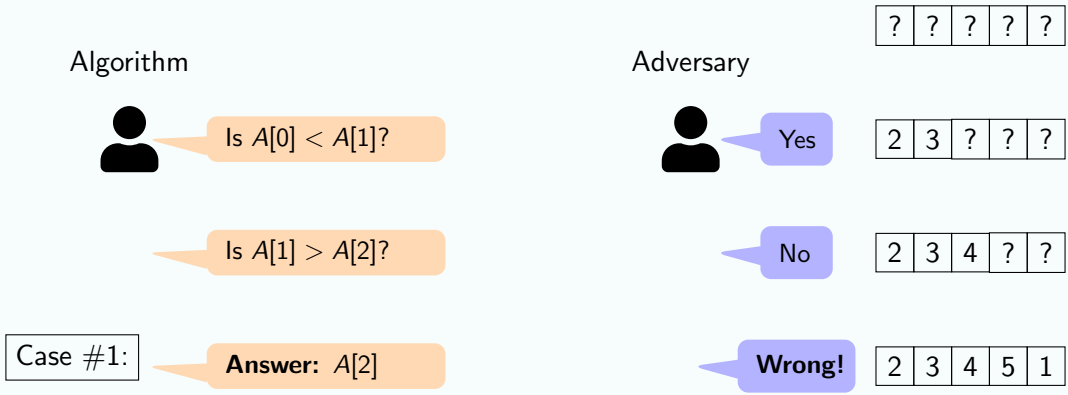
- If i is outputted by the algorithm, then fix $A[i]$ to be the smaller than all other elements.
- Otherwise, fix $A[i]$ to be larger than all other elements.

Therefore, any algorithm requires at least $\frac{n}{2} = \Omega(n)$ comparisons.

Exercise: Tweak this argument to prove that at least $n - 1$ comparisons are needed!

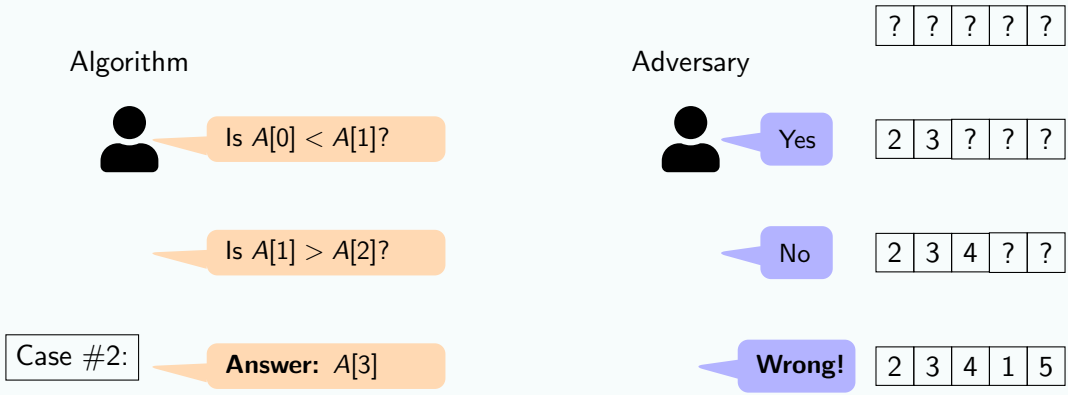
More Lower Bound Proofs: Adversary Argument

Extra Slide



More Lower Bound Proofs: Adversary Argument

Extra Slide



Q6. Weighing Balls

Suppose we have 243 balls, all of which have the same weight, except for one which is heavier.
You need to find which of the balls is heavier.
Your friend has a balance scale, but will charge you for each weighing.
Thus, you want to minimize the number of weighings needed.

What is the minimum number of weighing needed to find the ball which is heavier?

Q6. Weighing Balls

Leaves: Output / decision for the input

- What are the decisions? Which ball is the heavier one
- How many different decisions? 243

Branch: Outcome of comparison

- What are the outcomes? Either $<$, $=$ or $>$
- What will be the implication? The node can be split into three subtrees

Worst Case (number of comparisons) = Height of Tree

- What is the minimum possible height of the tree? $\log_3 243 = 5$
- Therefore, at least $\boxed{5}$ weighings are needed.

Note: This only shows the lower bound analysis. It doesn't mean that an algorithm that use 5 weighings exists.

Q6. Weighing Balls

Lower bound analysis (optimality):

- Assume we have a solution that can solve the problem in at most 4 weighings.
- Thus, height (H) of the decision tree $= 4$.
- Since each weighing can only have three results, each internal node can branch three times (B).
- The number of leaves in the tree must be $\leq B^H = 3^4 = 81$.
- However, there needs to be at least 243 leaves (the number of possible outcomes), contradiction.

Q6. Weighing Balls

Strategy (**Upper Bound**): Can be proven by providing a solution that solves the problem in that number of queries.

Q6. Weighing Balls

Strategy (**Upper Bound**):

- Divide the balls into 3 equal groups A, B and C.
- Weigh one group (A) against another (B).
  - $A < B$: The heavier ball is in B.
  - $A > B$: The heavier ball is in A.
  - $A = B$: The heavier ball is in C.
- Thus, in each weighing we can decrease the size of the problem to $1/3^{\text{rd}}$.
  - $243 \to 81 \to 27 \to 9 \to 3 \to 1$, 5 weighings are needed.

Q5. H-Index

Given an array $A[1 \dots n]$ sorted in **non-increasing** order, find the largest index i such that $A[i] \geq i$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c_i	170	108	60	43	21	15	14	14	14	12	11	9	8	6	3	2	2

Q5. H-Index

Participation Marks!

Given an array $A[1 \dots n]$ sorted in **non-increasing** order, find the largest index i such that $A[i] \geq i$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c_i	170	108	60	43	21	15	14	14	14	12	11	9	8	6	3	2	2

Question 5: What is the lower bound for any comparison-based algorithm?

Q5. H-Index

How many different decisions are there? All n are possible.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c_i	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c_i	99	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c_i	99	99	99	0	0	0	0	0	0	0	0	0	0	0	0	0	0

⋮

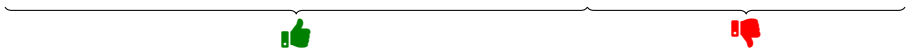
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c_i	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99

The lower bound of any comparison-based algorithm would be $\Omega(\log n)$.

Q5. H-Index

Binary Search $\Rightarrow$ Exploiting **Monotonic Properties**:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
c_i	170	108	60	43	21	15	14	14	14	12	11	9	8	6	3	2	2



- If the index j is valid ($A[j] \geq j$), then index $j-1$ is valid as well ($A[j-1] \geq A[j] \geq j \geq j-1$).
- If the index j is invalid ($A[j] < j$), then index $j+1$ is invalid as well ($A[j+1] \leq A[j] < j < j+1$).

Q5. H-Index

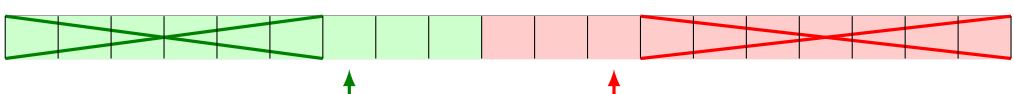
Algorithm Binary Search to find H-Index

```
1:  procedure HIndex(A)
2:      if A[1] < 1 then
3:          return None
4:      lo ← 1
5:      hi ← n
6:      while lo < hi do
7:          mid ← ⌈(lo+hi)/2⌉
8:          if A[mid] ≥ mid then          ▷ mid is valid
9:              lo ← mid
10:         else                          ▷ mid is invalid
11:             hi ← mid − 1
12:     return lo
```

- If valid, we can discard the left range (also valid, but with smaller index).
- If invalid, we can discard the right range (also invalid).

This algorithm runs in $O(\log n)$.

Exercise: Try to find an $O(\log i)$ solution where i is the final answer. Hint: Quickly find a smaller interval to start our binary search?



Enjoy your recess week!

Bonus. Tree Puzzle (Part 1)

Extra Slide

There is an (unknown) labeled tree with N vertices numbered $1, 2, \ldots, N$. Your are tasked to find the structure of this tree. Luckily, you are given an oracle which gives you a Yes/No answer to the following question (you may specify any X and Y):

- Given two *disjoint* sets X and Y. Does there exist $x \in X$ and $y \in Y$ such that the vertices x and y are connected by an edge?

What is the minimum number of questions (asymptotically) needed to complete this task? Prove both the lower and upper bounds.

(Hint: Search up online for any facts you need.)

Bonus. Tree Puzzle (Part 1)

Extra Slide

Solution: The minimum number of questions needed is $O(N \log N)$.

- By Cayley's formula, there are N^{N-2} distinct labeled trees with N vertices.
- Each query only comes with two possible outcomes (Yes/No), so we need a lower bound of $\log(N^{N-2}) = (N-2)\log N = \Omega(N \log N)$ questions.
- Let's now construct an algorithm that uses at most $2N\lceil \log N \rceil$ queries:
  - Consider each vertex v. Let $S = \{v+1, v+2, \ldots, N\}$.
  - While querying $(\{v\}, S)$ gives the answer "Yes",
    - Keep removing half of the elements from S and query again. If the answer is "No", then change S to the elements removed. As such, we can find a vertex that is connected to v within $\lceil \log N \rceil$ queries, or determine that there are no more vertices connected to v.
    - After we find a vertex u that is connected to v, remove u from S.
  - There are at most $2N-1$ searches with $\lceil \log N \rceil$ queries each ($N-1$ searches successfully finding an edge, and N searches not being able to find an edge).