



# Lecture 1

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

# Course staff

Instructor: Ioannis Panageas

Email: ipanagea at ics dot uci dot edu

Office hours: Wednesday 2:00-4:00pm (zoom)

TAs:

Rohan Chauhan (rmchauha at uci dot edu)

Office hours: TBA

Nikolas Patris (npatris at uci dot edu)

Office hours: (zoom)

Jingming Yan (jingmy1 at uci dot edu)

Office hours: TBA

# Course material

We will use canvas for announcements. Slide materials will be posted on  
<https://panageas.github.io/algo2026.html>

We will use gradescope for posting homeworks and grading.

We will be using Edstem for questions of general interest about the course material, the homework, and the tests

<https://edstem.org/us/courses/90397/discussion>

## Required Textbook

- Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia.

## Recommended Textbook

- Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.

# Grading

- **Homeworks:** 4%
    - There will be given 4 Homeworks to solve.
  - **Midterms:**  $23+23+23 = 69\%$ 
    - There will be given 3 midterms, on Tuesdays of week 4,6 and 9.  
Each midterm will contain topics from previous weeks.
  - **Final :** 30%
    - Material from all weeks (except last week).
- Total: 103 (3 points extra credit).
- If you score 21 or more in each midterm, you get an A (no need to take the final).
  - If grade of final is better than the total, you keep the final.

# Letter Grades

- Not a straight scale nor straight curve
- 90% and up guaranteed some sort of A or A-
- 80% and up guaranteed at least B-
- 70% and up guaranteed at least C-

# Submitting Assignments

- Written assignments in Gradescope.
  - Must be legible
    - If you have messy handwriting, type your homework!
  - Must be on-time.
  - Deadline: Fridays 23:59pm (see syllabus)
  - HWs are worth 4 points. Are meant for practice.

# Exam Dates and Rules

- The exams are held on the **days listed (syllabus)**
  - See policy in syllabus, **no** makeup exams
- Exams will not be excused for reasons within your control.
- If there is a valid reason (needs approval from instructor) for missing an exam, the grade will be **split equally** among the other components.

# Academic Integrity Policy

- If you **need help**, see:
  - Ioannis
  - TAs
- **Plagiarism** risks an **F** in the class and more.
- The following are examples of **not okay**:
  - Chegg                    GeeksForGeeks
  - CourseHero            Quora
  - StackOverflow        Github (generally)
  - Chatgpt or related platform

# Collaboration with classmates

- You can discuss some things freely with others:
  - What a problem is asking
  - How to do a non-homework or non-exam problem
  - How something from lecture worked
- You should **never**:
  - **Show** your homework assignment to someone else
  - Write your solutions from notes taken **outside** lecture / **discussion**
  - Seek homework solutions from **outside** sources -- especially online!
  - Tell a student specifically how to solve a homework problem
- Penalty for academic **dishonesty**: F in the course.

# To-Do This Week

- Read **the syllabus**
  - Treat it as though it's a reading assignment.
  - Main document plus associated policy documents
- Review Prerequisites
  - Help is available all week, including at all discussion sections

# What is algorithm

- Algorithm is a procedure for solving a task



e.g. how do you sort a cart of books in increasing order of the volume number? (i.e. volume 1, volume 2, volume 3....)

- Bad algorithm: compare all books, put smallest volume in the beginning and repeat.
- Clever algorithm: divide the cart into two, sort the first half, sort the second half, merge them.

# What is algorithm

- Algorithm is a procedure for solving a task



e.g. How to find the best travelling time between from a station to **any other** station?

- Bad algorithm: manually find the travelling between each station.
- Clever algorithm: just record the travelling time between consecutive stations, then use the **Dijkstra shortest path** algorithm.

# Case study I: Finding a Celebrity

Since coming to UC Irvine, has anyone met a celebrity?



# What is a celebrity?

- Within a group of people  $G$ , we say a person  $p$  is a **celebrity** (famous) if:
  - Everyone knows who  $p$  is  
(celebrities must be known by everyone)
  - Person  $p$  does not know who anyone else is
- **Goal:** Find a celebrity from  $G$  if there exists one.

# What is a celebrity?

- Within a group of people  $G$ , we say a person  $p$  is a **celebrity** (famous) if:
  - Everyone knows who  $p$  is  
(celebrities must be known by everyone)
  - Person  $p$  does not know who anyone else is
- **Goal:** Find a celebrity from  $G$  if there exists one.
- You are allowed to only **query** if person  $i$  knows person  $j$  for various choices of  $(i, j)$ .

# Brute force approach

- Given a person  $p$  we want to check if it is a celebrity
  - How efficiently can I check if person  $p$  is a celebrity? # of queries

# Brute force approach

- Given a person  $p$  we want to check if it is a celebrity
  - How efficiently can I check if person  $p$  is a celebrity? # of queries
  - Query all other persons if they know  $p$  and also if  $p$  does not know them.

# Brute force approach

- Given a person  $p$  we want to check if it is a celebrity
  - How efficiently can I check if person  $p$  is a celebrity? # of queries
  - Query all other persons if they know  $p$  and also if  $p$  does not know them.

This gives  $2n - 2$  queries where  $n$  is the group size.

# Brute force approach

- Given a person  $p$  we want to check if it is a celebrity
  - How efficiently can I check if person  $p$  is a celebrity? # of queries
  - Query all other persons if they know  $p$  and also if  $p$  does not know them.

This gives  $2n - 2$  queries where  $n$  is the group size.

- We have to do the above for all possible persons  $p$ .

# Brute force approach

- Given a person  $p$  we want to check if it is a celebrity
  - How efficiently can I check if person  $p$  is a celebrity? # of queries
  - Query all other persons if they know  $p$  and also if  $p$  does not know them.

This gives  $2n - 2$  queries where  $n$  is the group size.

- We have to do the above for all possible persons  $p$ .

Total queries are  $(2n - 2) \cdot n$  which gives  $\Theta(n^2)$  .

# Brute force approach

- Given a person  $p$  we want to check if it is a celebrity
  - How efficiently can I check if person  $p$  is a celebrity? # of queries
  - Query all other persons if they know  $p$  and also if  $p$  does not know them.

This gives  $2n - 2$  queries where  $n$  is the group size.

- We have to do the above for all possible persons  $p$ .

Total queries are  $(2n - 2) \cdot n$  which gives  $\Theta(n^2)$  .

Can we do better?

# Faster approach

- Put all the members in a list (arbitrary order)
  - Pick the first two members of the list, let  $p, q$ .
  - Query if  $p$  knows  $q$ .

# Faster approach

- Put all the members in a list (arbitrary order)
  - Pick the first two members of the list, let  $p, q$ .
  - Query if  $p$  knows  $q$ .

2 Cases:

1.  $p$  knows  $q$ . Then  $p$  is **not a celebrity** (remove  $p$  from the list).
2.  $p$  does not know  $q$ . Then  $q$  is **not a celebrity** (remove  $q$  from the list).

# Faster approach

- Put all the members in a list (arbitrary order)
  - Pick the first two members of the list, let  $p, q$ .
  - Query if  $p$  knows  $q$ .

2 Cases:

1.  $p$  knows  $q$ . Then  $p$  is **not a celebrity** (remove  $p$  from the list).
  2.  $p$  does not know  $q$ . Then  $q$  is **not a celebrity** (remove  $q$  from the list).
- Repeat the above process. At **every iterate**, we remove **one person**.

# Faster approach

- Put all the members in a list (arbitrary order)
  - Pick the first two members of the list, let  $p, q$ .
  - Query if  $p$  knows  $q$ .
  - Repeat the above process. At **every iterate**, we remove **one person**.

After  $n - 1$  "iterates" we have **one** member in the list.

Check if this **remaining** person is a celebrity.

# Faster approach

- Put all the members in a list (arbitrary order)
  - Pick the first two members of the list, let  $p, q$ .
  - Query if  $p$  knows  $q$ .
  - Repeat the above process. At **every iterate**, we remove **one person**.

After  $n - 1$  "iterates" we have **one** member in the list.

Check if this **remaining** person is a celebrity. Why do you need to check?

# Faster approach

- Put all the members in a list (arbitrary order)
  - Pick the first two members of the list, let  $p, q$ .
  - Query if  $p$  knows  $q$ .
  - Repeat the above process. At **every iterate**, we remove **one person**.

After  $n - 1$  "iterates" we have **one** member in the list.

Check if this **remaining** person is a celebrity. Why do you need to check?

Total queries are  $2n - 2 + n - 1 = 3n - 3$  which gives  $\Theta(n)$  .

# Case study II: Finding the heaviest and lightest item

- We are given a set of  $n$  items of different weights:

$$x_1, x_2, \dots, x_n$$

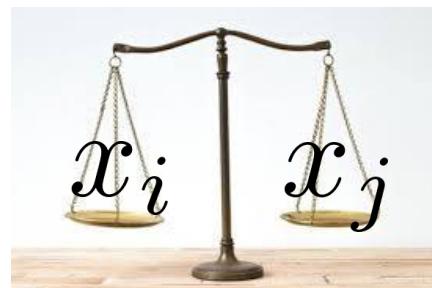
- Goal: Find the **heaviest** and the **lightest** item.

# Case study II: Finding the heaviest and lightest item

- We are given a set of  $n$  items of different weights:

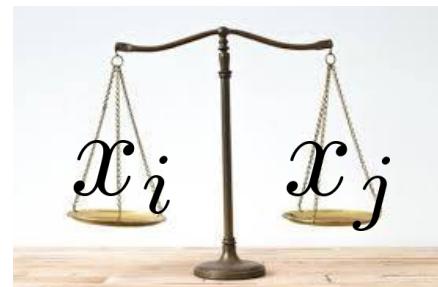
$$x_1, x_2, \dots, x_n$$

- Goal: Find the **heaviest** and the **lightest** item.
- You are allowed to only compare  $x_i$  with  $x_j$  for various choices of  $i, j$ .



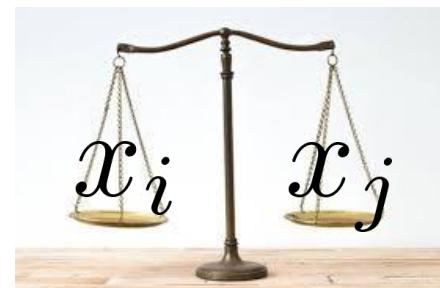
# Brute force approach

- Find the heaviest item among  $x_1, x_2, \dots, x_n$ . How many comparisons?



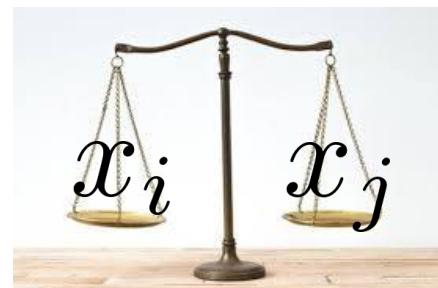
# Brute force approach

- Find the heaviest item among  $x_1, x_2, \dots, x_n$ .  $n - 1$



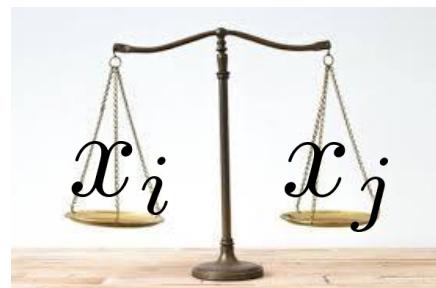
# Brute force approach

- Find the heaviest item among  $x_1, x_2, \dots, x_n$ . n - 1
- Find the lightest item among  $x_1, x_2, \dots, x_n$ . How many comparisons?



# Brute force approach

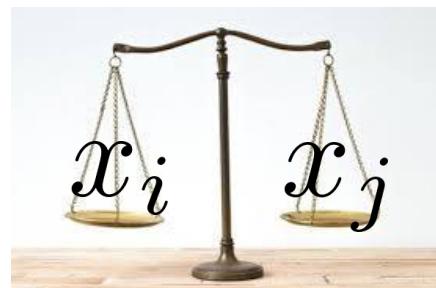
- Find the heaviest item among  $x_1, x_2, \dots, x_n$ . n - 1
- Find the lightest item among  $x_1, x_2, \dots, x_n$ . n - 1



# Brute force approach

- Find the heaviest item among  $x_1, x_2, \dots, x_n$ . n - 1
- Find the lightest item among  $x_1, x_2, \dots, x_n$ . n - 1

Total number of comparisons  $2(n - 1)$ .

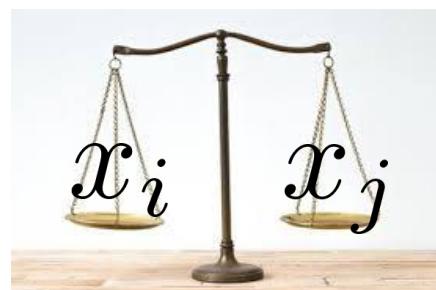


# Brute force approach

- Find the heaviest item among  $x_1, x_2, \dots, x_n$ . n - 1
- Find the lightest item among  $x_1, x_2, \dots, x_n$ . n - 1

Total number of comparisons  $2(n - 1)$ . You may get  $2n - 3$ .

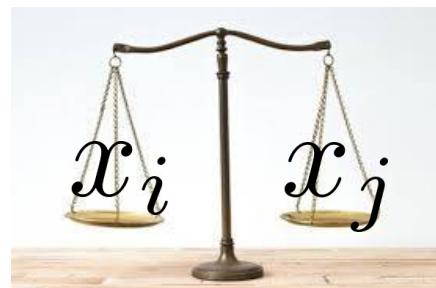
Can we do better?



# Faster approach

- Compare  $x_1$  with  $x_2$ ,  $x_3$  with  $x_4$  etc (like round 1 of knock-out tournament).

Total number of comparisons  $\frac{n}{2}$ .

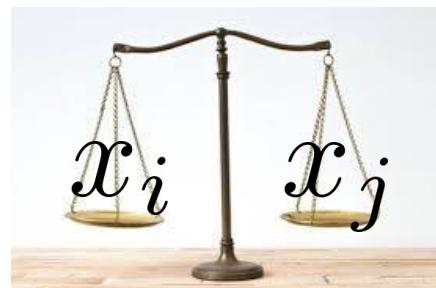


# Faster approach

- Compare  $x_1$  with  $x_2$ ,  $x_3$  with  $x_4$  etc (like round 1 of knock-out tournament).

Total number of comparisons  $\frac{n}{2}$ .

- Find heaviest among winners of round 1.



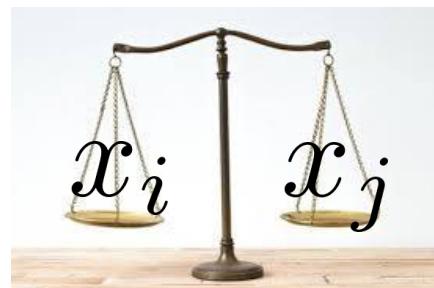
# Faster approach

- Compare  $x_1$  with  $x_2$ ,  $x_3$  with  $x_4$  etc (like round 1 of knock-out tournament).

Total number of comparisons  $\frac{n}{2}$ .

- Find heaviest among winners of round 1.

$$\frac{n}{2} - 1$$



# Faster approach

- Compare  $x_1$  with  $x_2$ ,  $x_3$  with  $x_4$  etc (like round 1 of knock-out tournament).

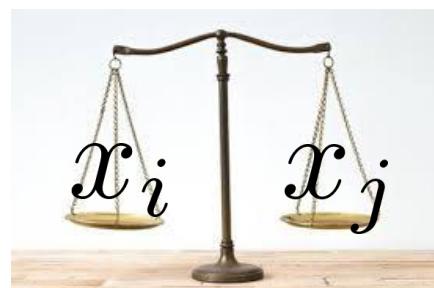
Total number of comparisons  $\frac{n}{2}$ .

- Find heaviest among winners of round 1.

$$\frac{n}{2} - 1$$

- Find lightest among losers of round 1.

$$\frac{n}{2} - 1$$



# Faster approach

- Compare  $x_1$  with  $x_2$ ,  $x_3$  with  $x_4$  etc (like round 1 of knock-out tournament).

Total number of comparisons  $\frac{n}{2}$ .

- Find heaviest among winners of round 1.

$$\frac{n}{2} - 1$$

- Find lightest

$$\# \text{ comparisons } \frac{3n}{2} - 2$$

