# Introduction to Competitive Programming

Workshop – Term 1, Week 1

**Susan He, Austin Song**

# Outline

# Attendance!

Please scan the attendance form so you can get free pizza

# About Programming Workshops

- We will run a **2 hour** workshop around once every two weeks.
- Learn about competitive programming, develop problem solving skills.
- Structured learning – some workshops might require prerequisite knowledge:
  - e.g. A Graph Theory II workshop might assume you know the basics taught by a previous Graph Theory I workshop.
- Difficulty will slightly build throughout the year:
  - If too easy, come back in a few weeks and keep an eye out for new topics!
  - We'll move at a steady rate, so don't be worried about sudden jumps in difficulty!
- Informal; feel free to ask questions at any time (and there's **free food**)!
- Don't worry if you miss an event, **workshops will be recorded and slides will be uploaded**.

# VJudge For Workshop

- For most workshops we'll have a problemset on VJudge for you to practice the workshop problems and content.
- There's no time limit or realistically close end time so feel free to spend however long on these problems.

Link: https://vjudge.net/contest/790218

Password: `cpmsoc`

# Outline

# What is Competitive Programming

**Definition (Competitive Programming)**

**Competitive programming** is an activity where participants write programs to solve algorithmic problems. These programs are usually under time constraints.

These are quite different from your average hackathons or project specifications – but they're quite similar to interview questions!

# Why Learn Competitive Programming?

- Hella fun!!!
- Chilling with your team and make new friends
- You'll meet super smart people who can nepo you! (ok it's like a joke right)
- Great for developing problem solving skills and acing technical (coding) interviews
- Exposure to common algorithms and data structures which are relevant to any programming field

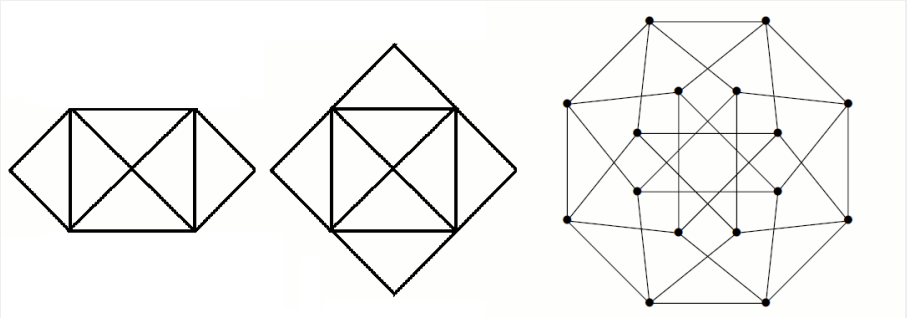
- get a job :)

# Outline

# Problem Warmups

## Drawing Shapes

For each of the shapes below, is it possible to draw them without lifting your pencil?



This is a **graph** problem! Specifically, we are trying to find an **Eulerian trail**.

# Problem Warmups

## Counterfeit Coin

Suppose you have a 100 silver coins. Among these is a single counterfeit coin, which weighs slightly less than all other coins. You have a simple scale, and you can put any number of coins on either side of the scale.

What is the minimum number of weighings needed to find the counterfeit?

What if there are $N$ silver coins?

This is a **divide and conquer problem**! We repeatedly divide the problem, which results in a quicker solution than weighing one by one.

# Problem Warmups

## Simple Nim

There is a pile of 14 jellybeans. You and your friend decide to play a game to see who gets all of the jellybeans. Each turn, you must remove 1, 2 or 3 jellybeans from the pile. The person who removes the last jellybean wins.

If you go first, how can you ensure your victory?

What if there were $N$ jellybeans?

This is a classic **game theory** problem. In fact, there is a theorem which states any **impartial** game is equivalent to some Nim game!

# Outline

# Anatomy of a Problem

Let's break down a simple programming problem:

## Cardiac Arrest

**Statement**:
You are a human with a heart. Unfortunately, you crashed when driving, so now you are in hospital. Your heart has $N$ more beats left. Every $K$ beats, the electrocardiograph beeps. How many beeps will you hear before you die?

**Input/Output**:
There will be one line with space-separated integers $N$ $K$. Output a single integer – the number of beeps you'll hear.

**Subtasks & Constraints**:

- Subtask 1 (10%): $1 \leq K \leq N \leq 10^6$.
- Full (90%): $1 \leq K \leq N \leq 10^9$.

# Sample Cases

Problem statements usually come with simple sample cases.

## Cardiac Arrest - Sample Cases

Sample Input 1:

```
11 3
```

Sample Output 1:

```
3
```

Explanation:

You will hear a beep on heartbeats 3, 6, 9. You die before you hear 12. Note that you can hear a beep if it occurs on beat 11 (beat $N$), but unfortunately there's nothing on $11$.

# Submissions and Test Cases

The most common format of competitive programming problems are **batch problems**.

## Batch Problems

You submit a program, and the contest server will test it against a batch of inputs. Usually you must pass all test cases of a subtask to score points, and test cases are hidden.

On a specific test case, a program can throw a number of different fail **verdicts**...

- Wrong Answer (WA): Program output isn't correct.
- Time Limit Exceeded (TLE): Program was too slow.
- Runtime Error (RE/RTE): Program encountered an error during runtime.

...and throw exactly one type of success verdict:

- Accepted (AC): The program correctly solves all the test cases in time :).

# Outline

# Solving a Problem

There are two main parts to solving any sort of competitive programming-style problem (follow this in interviews too!)

1. **Headsolving: coming up with the algorithm to implement.**
   - This is independent of programming language.
   - Should be able to predict speed and correctness.
   - Make observations to help you!
   - Start with a simple algorithm and slowly improve it!
2. **Implementing: coding the algorithm.**
   - During some contests, you can refer to pre-coded implementations of common algorithms.

# Time Complexity (Setup)

Let's revisit the sample problem **Cardiac Arrest**. Consider two solutions:

## Solution A

Consider every heartbeat index from $1$ to $N$. The $i$'th heartbeat causes a beep and only if $i$ is divisible by $K$. We can loop through all numbers from $1$ to $N$, and count how many fulfil this condition.

## Solution B

Notice that we hear a beep for every lot of $K$ heartbeats. Therefore, we can simply divide $N$ by $K$, rounding down if there is a remainder.

Clearly one seems 'faster' than the other (no matter how fast your computer is), especially when $N$ grows large. *How do we quantify this?*

# Time Complexity (Big-O Notation)

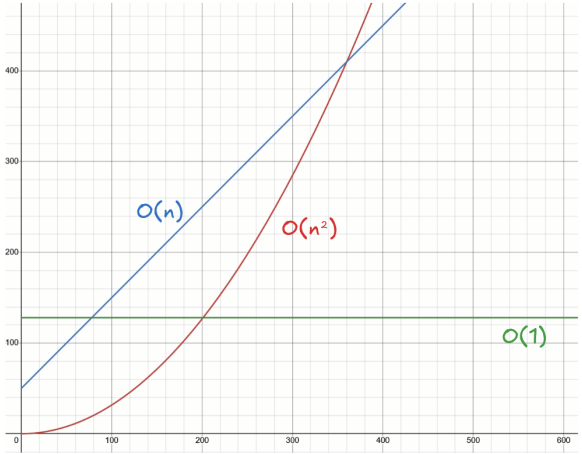
The efficiency of an algorithm is an important factor in competitive programming. Big-O notation quantifies the **time complexity** of an algorithm, i.e. how the algorithm scales with input size.

- As $N$ increases, the time taken by solution A also increases. As $N$ doubles, we can expect the time to also double. This is called a **linear time complexity**, denoted $O(N)$.
- For solution B, no matter how big $N$ gets, the solution will take roughly the same time. This is called a **constant time complexity**, denoted $O(1)$.

Note that we disregard **constant factors** and any **non-dominating term** since they become irrelevant pretty quickly. As long as $N$ grows large enough, an $O(N)$ solution will always become 'quicker' than an $O(N^2)$ solution.

**Exercise**: So what's faster, an $O(N^3 + 2N)$ algorithm or an $O(10^6 N^2)$ one?

# Time Complexity




The average contest server can perform around $10^7$ operations in one second. That means for an $O(N^2)$ solution, the program will take $< 1$ s for roughly $N \leq 3000$.

# Consequences of Time Complexity

What does all this maths actually mean?

Well, it means you can **guess** the rough algorithm *based on the constraints*!

| Constraint | Complexity |
|---|---|
| $N \leq 10$ | $O(N!)$ |
| $N \leq 20$ | $O(2^N)$ |
| $N \leq 500$ | $O(N^3)$ |
| $N \leq 5 \times 10^3$ | $O(N^2)$ |
| $N \leq 2 \times 10^5$ | $O(N \log N)$ |
| $N \leq 2 \times 10^6$ | $O(N)$ |
| $N \leq 10^9$ | $O(\log N)$ or $O(1)$ |

Table: Common constraints and their intended complexities.

# Outline

# Triplet Pairs

## Problem

**Statement**: Your favourite toy as a kindergartener was a group of $N$ number blocks, with the $i$'th one displaying the number $a_i$. Back then, you always wondered how many distinct pairs of blocks $(i, j)$ (note $i < j$) had a sum that was a multiple of 3. Now that you're older, you can computationally solve this problem.

**Input/Output**: You're given $N$ on the first line, followed by $N$ space separated integers on the second line, the $i$'th of which corresponding to $a_i$. Output the answer (a single integer).

**Subtasks & Constraints**: Time limit: $2$ s; Memory limit: $1$ GB.

- Subtask 1 (50%): $N \leq 2 \times 10^3$.
- Subtask 2 (50%): $N \leq 2 \times 10^6$.
- For all subtasks, $1 \leq a_i \leq 10^5$.

# Triplet Pairs

## Sample Data

Sample Input 1:

```
5

2 6 3 8 1
```

Sample Output 1:

```
3
```

Explanation:

The pairs $(2, 1), (8, 1)$ and $(3, 6)$ have sums divisible by 3.

# Triplet Pairs (Headsolving)

Look at subtasks individually to help break down the problem.

How to solve subtask 1?

- The bounds call for an $O(N^2)$ solution.
- There are also $\binom{N}{2} \sim O(N^2)$ pairs – hmmm...
- Try every distinct pair $(i, j)$ where $i < j$. If `(a[i]+a[j]) % 3 == 0`, then we increment a counter storing the answer.

What observations can we make for a full solution?

- Only the residues mod 3 matter. We can compute the number of elements (denote as $c_i$ for $0 \leq i < 3$) which are $0 \bmod 3$, $1 \bmod 3$, and $2 \bmod 3$.
- The final answer is $c_1 \times c_2 + \frac{c_0 \times (c_0 - 1)}{2}$.
- Thus, perform an $O(N)$ pass to compute $c_i$ for $i \in \mathbb{Z}_3$. Then, $O(1)$ computation of the answer yields an $O(N)$ solution scoring 100%.

# Guess The Array

## Problem

**Statement**: This is an **interactive** problem. You *interact* live with the grader in an attempt to find the list of numbers it's hidden!

They tell you the list $A$ has $N$ numbers. You can ask $Q$ questions:

*"What's* $A_i + A_j$*?"*

where $A_i$ means the $i$'th number in the list. Help recover the array!

**Subtasks & Constraints**:

- Subtask 1 (40%): $N = 3$, $Q = 10^5$.
- Subtask 2 (60%): $N = 2 \times 10^5$, $Q = 5 \times 10^5$.

# Chocolate

## Problem

**Statement**: The novel nonpareil Multi-Flavour-Choco-Bar™ has $N$ pieces, each with a different flavour. You consider the $i$'th piece as having an integer $a_i$ deliciousness (might be negative!). Find the contiguous segment of chocolate with greatest sum of deliciousness.

**Constraints**: $N \leq 2 \times 10^6$, $a_i \leq 10^5$ for all $i$.

# Outline

# Practice!

Practice makes perfect; wanna be confident that you can devise an efficient solution and think clearly under heavy time pressure (and people pressure)!

- **ORAC2**: Good introductory problems and materials, suitable for beginners.
  https://orac2.info/
- **Leetcode**: Everyone's heard of this; good place to get the basics down and practice applications of new techniques.
  https://leetcode.com/
- **Codeforces**: Huge problem list spanning wide range of difficulties and topics. Regular contests if you're down to stay up until 5 am!!1!
  https://codeforces.com
- **Atcoder**: Reasonable quality beginner contests at reasonable contest times (11 pm - 1 am), and past contest questions.
  https://atcoder.jp/

# Other Events

We have **subcommittee recruitment** happening now!

Reasons to join:

- The workshop insulted your entire ancestral line and prognosticated progeny so you want to help make it better.
- You can make funny reels.
- You can write an impossible problem and put it in the O-Week Contest and gaslight everyone into thinking it's easy.
- You can be a nerd and join maths.
- Whenever the contest site goes down you can get desperately notification-spammed to fix the server.
- You can cold-email large companies for money.
- You can make friends. (optional)

# Other Events

We have **subcommittee recruitment** happening now!

Real reasons to join:

- Good way to get to know really cracked people!
- You can learn a lot in a tight-knit supportive/like-minded group.
- Make lots of friends (esp. if you're a first year).

**Beginners are highly welcome to apply to non-technical ports (marketing, careers, socials)!**

# Other Events

There's an **IMC Contest** coming soon! (monetary prizes and career development)

Interested in trading? Come to our **Introduction to Quant workshop** planned for Friday Week 2.

There's also a **introductory competitive mathematics workshop** coming on the 6th of March 4pm-6pm!

Enjoyed this workshop? Our next **workshop on common algorithms** will be on Week 3 Wednesday, 5pm-7pm.

If you want to get into competitive programming in UNSW, try our **SPAR** rounds, or the **ICPC Regionals**! CPMSoc also organises **small contests** throughout the year (like our O-Week Horse Contest).

# Feedback!

Please fill out feedback form so we know how to run future workshops :D

(a) Feedback form

(b) Attendance form


UNSW CPMSoc