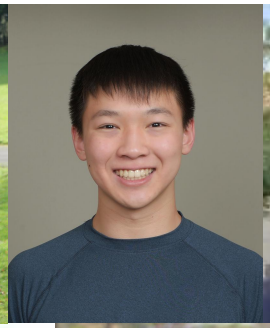
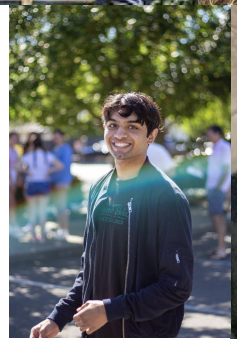
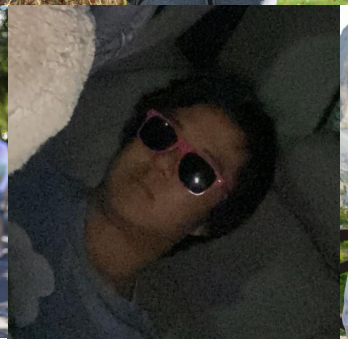
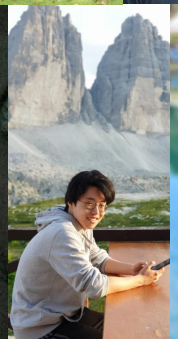
Introduction to Competitive Programming and Algorithms

Mondays 6:30-8:00PM - Physics Building 2

Officer Introductions

The lectures, content, and OH for this DeCal will be conducted by the officers of Competitive Coding at Berkeley.

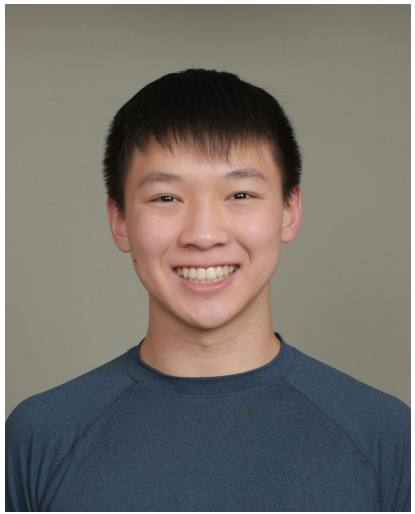
Xavier Plourde

Sophomore, *CS* - Lectures and Content

James Rungsawang

Junior, *CS* - Lectures and Content

Ryan Zhu



Junior, CS + DS + Linguistics - Lectures, Content, and OH

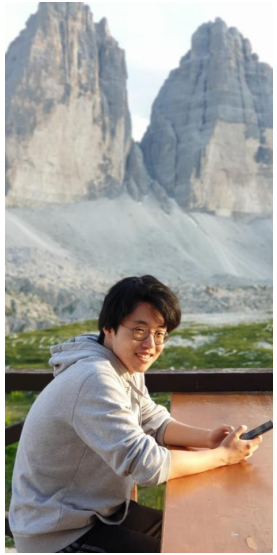
Nishant Bhakar

Sophomore, *CS + Physics* - Content, Office Hours

Chris Liu

Junior, *CS* - Problem Writing

Youngmin Park



Freshman, *CS + Math* - Office Hours

Vivek Verma

Sophomore, *CS + Math* - Lectures

Competitive Programming

Competitive Programming is a mind sport involving solving as many algorithmic programming problems as possible within a short amount of time (usually 2-5 hours).

In a typical programming contest, contestants are given a problem set with 5-15 problems, and are asked to write solutions to solve as many problems as possible.

Competitive Programming is a great way to practice using algorithms and data structures, and the competitions are often fun and fast-paced!

Competitive Programming

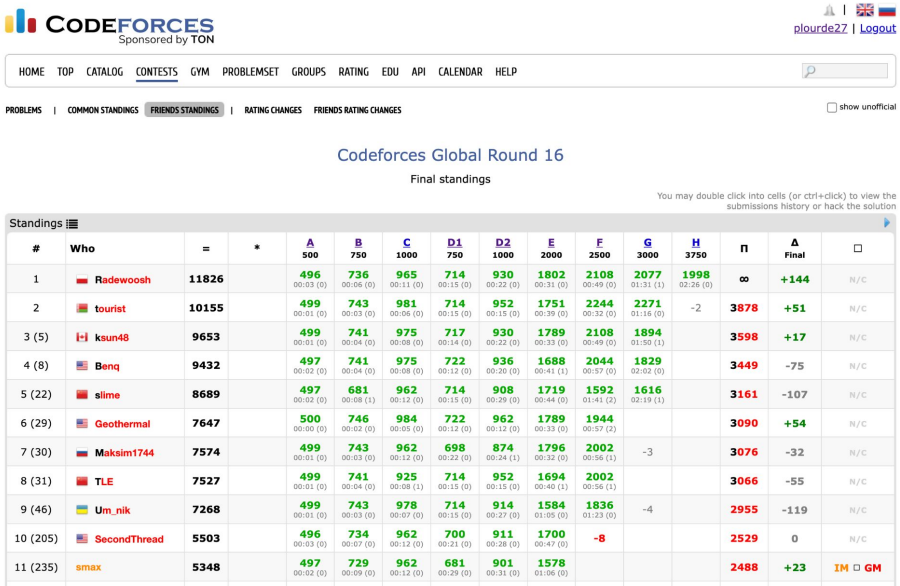
Many concepts from competitive programming are partially taught in classes like CS61B, CS70, and CS170 (e.g. binary search, graph theory, minimum spanning trees, etc.)

However, in this DeCal we'll emphasize techniques and strategies not taught in these classes. Although we'll go over all of the necessary algorithms, we'll emphasize

Why do competitive programming?

1. Interview Prep
2. Implementation and coding skills
    a. Good prep for a large amount of classes at Berkeley especially CS 170, CS 61B, and CS 70
3. Problem solving
4. It’s fun!

Competitive Programming



Some examples of competitive programming contest leaderboards

Codeforces

Codeforces is the leading website for frequent competitive programming contests.

They host contests every 2-3 days, which attract around 10k participants on average from all over the world.

Contests range in difficulty from Div. 1 (very difficult) to Div. 4 (easier). Compared to ICPC, it focuses more on problem-solving ability, and less on theory and algorithms.

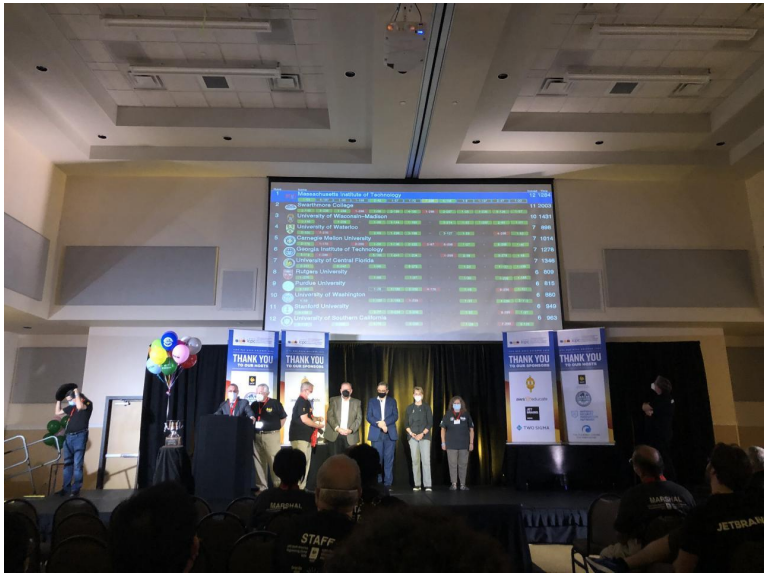
https://codeforces.com

International Collegiate Programming Contest

ICPC is the main programming contest for college students, consisting of regionals, nationals and world finals.

- Regionals involve competing against teams from across the PacNW region, including NorCal, Oregon, Washington, Idaho, Hawaii. Up to 6 teams from a given school can go to regionals. This year, Regionals is taking place in-person at Chico State!
- The top 5 teams from regionals (only one per school) advance to nationals, which is in-person in Orlando, FL.
- The top 10-20 teams from nationals advance to the World Finals. World Finals 2023 will likely be held in Egypt!

International Collegiate Programming Contest



Some photos from ICPC Nationals 2022!

Berkeley ICPC History

Berkeley won the World Finals 1996 in Waterloo, Canada!

- Berkeley is the third-most-recent school from the U.S. to win the world finals - all of the most recent winners are from Russia or China (except for MIT, who won World Finals 2021)

Berkeley also won a silver medal at World Finals in 2015 in St. Petersburg, Russia.

Berkeley ICPC History

However, Berkeley has not had a competitive programming club for several years, until last fall

We still sent teams to ICPC every year, but it’s been much less official and formal.

Competitive Coding at Berkeley

Last fall, we founded Competitive Coding at Berkeley. In addition to this DeCal, the club runs many more competitive programming events:

- Weekly advanced topic tutorials (this is an unofficial meeting, rather than a DeCal - we may expand this into a second DeCal for future semesters)
- Weekly ICPC team practice contests on Sundays
- Organized virtual participations of Codeforces contests
- Monthly Lockout tournaments with other schools (our last tournament featured from participants from Swarthmore, Georgia Tech, and Harvard)

DeCal Policies

This DeCal will meet **Mondays from 6:30-8:00 PM** in **Physics Building 2**.

Here is our website: https://competitivecodingberkeley.github.io/decal/

This course is designed to be a survey of useful algorithms and their applications in competitive programming and coding interviews. Before taking this course, you should have a basic understanding of programming (61A or equivalent) and preferably some experience with either Python or C++.

Reference book: https://cses.fi/book/book.pdf

DeCal Policies

week	topic	book ch	resources
1 (jan. 23)	input and output, time complexity	1.2, 2	slides
2 (jan. 30)	prefix sums	9.1, 2.4	slides
3 (feb. 6)	greedy algorithms	6	slides
4 (feb. 13)	sorting, two pointers	3.1, 3.2	slides
5 (feb. 20)	*BREAK (PRESIDENT'S DAY)*		
6 (feb. 27)	binary search	3.3	slides
7 (mar. 6)	graph representations	11, 12	slides
8 (mar. 13)	complete search with recursion	5	slides
9 (mar. 20)	dynamic programming	7	slides
10 (mar. 27)	*BREAK (SPRING RECESS)*		
11 (apr. 3)	more dynamic programming	7	slides
12 (apr. 10)	segment trees	9.3	slides
13 (apr. 17)	rolling hashes	26.3	slides
14 (apr. 24)	wrap-up and final contest		slides

DeCal Policies

The class is composed of two components, the lecture between **6:30-8 PM on Mondays** introducing concepts and a corresponding weekly assignment due at Sunday midnight. **Two homework assignments will be dropped!**

We expect the homework to take around one hour of work. Some homeworks may have optional challenge problems. Homework will be conducted through Codeforces.

Remote office hours will be available on discord; schedules will be announced later this week.

DeCal Policies

Category	Percent	Criteria
Homework	70%	Demonstrated effort, participation
Attendance	30%	Form with password at each lecture

You receive a P in this class if you receive 70% or higher aggregate score. Late work will get 50% credit.

The class shouldn’t be stressful! If you have extenuating circumstances or find yourself falling behind, please reach out to us.

DeCal Policies

Study groups are allowed and encouraged. Sharing ideas is fine, but sharing code is **not allowed**.

Immediately after weekly assignments are due, submissions will be public and we will release solutions.

To incentivize lecture attendance, lecture recordings will **not** be posted publicly, but if you have extenuating circumstances for missing a lecture (e.g. midterm, sick) you can DM one of the officers and we'll send you the recording.

Join our Discord Server!

That is, if you're not already in it – course announcements and office hours will take place on this server going forward.

Here's a link for it too:

https://discord.gg/shgm4rKj

Create a Codeforces Account

Then, join our group at
https://codeforces.com/group/1HQN7rY5SE/contests

Content

For the rest of today’s lecture, we’ll go over how to read problem statements, time complexity, and how to parse input.

Anatomy of a Problem Statement

"How do you read this thing?"

1. **Anatomy of a Problem Statement**
2. Input Parsing
3. Time Complexity, Revisited!
4. Solving a Problem with Big O

Problem Statements

- **Problem statement** (describes the problem itself)
- **Input details** (constraints, format, etc.)
- **Output details** (format)
- **Example input(s)**
- **Example output(s)**
- **Notes/clarifications** (explains the example test cases)

It's important to read and really understand a problem statement. You can easily make mistakes and waste time if you misread or make invalid assumptions

A. Good Pairs

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array $a_1, a_2, \ldots, a_n$ of positive integers. A *good pair* is a pair of indices (i, j) with $1 \le i, j \le n$ such that, for all $1 \le k \le n$, the following equality holds:

$$|a_i - a_k| + |a_k - a_j| = |a_i - a_j|,$$

where $|x|$ denotes the absolute value of x.

Find a good pair. Note that i can be equal to j.

Input
The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 10^5$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) where a_i is the i-th element of the array.

The sum of n for all test cases is at most $2 \cdot 10^5$.

Output
For each test case, print a single line with two space-separated indices i and j which form a good pair of the array. The case $i = j$ is allowed. It can be shown that such a pair always exists. If there are multiple good pairs, print any of them.

Example

input (Copy)

```
3
3
5 2 7
5
1 4 2 2 3
1
2
```

output (Copy)

```
2 3
1 2
1 1
```

Note
In the first case, for $i = 2$ and $j = 3$ the equality holds true for all k:

- $k = 1$: $|a_2 - a_1| + |a_1 - a_3| = |2 - 5| + |5 - 7| = 5 = |2 - 7| = |a_2 - a_3|$,
- $k = 2$: $|a_2 - a_2| + |a_2 - a_3| = |2 - 2| + |2 - 7| = 5 = |2 - 7| = |a_2 - a_3|$,
- $k = 3$: $|a_2 - a_3| + |a_3 - a_3| = |2 - 7| + |7 - 7| = 5 = |2 - 7| = |a_2 - a_3|$.

Example Problem Statement

Add Two Numbers

Richard is a brand new competitive programmer and has already been given the infamous "Add two numbers" assignment. Unfortunately he cannot do it alone and needs your help.
There will be ***t*** queries in which you will have to compute the sum of two numbers.

Input

The first line of input contains **t** ($1 \leq t \leq 10$), the number of queries.

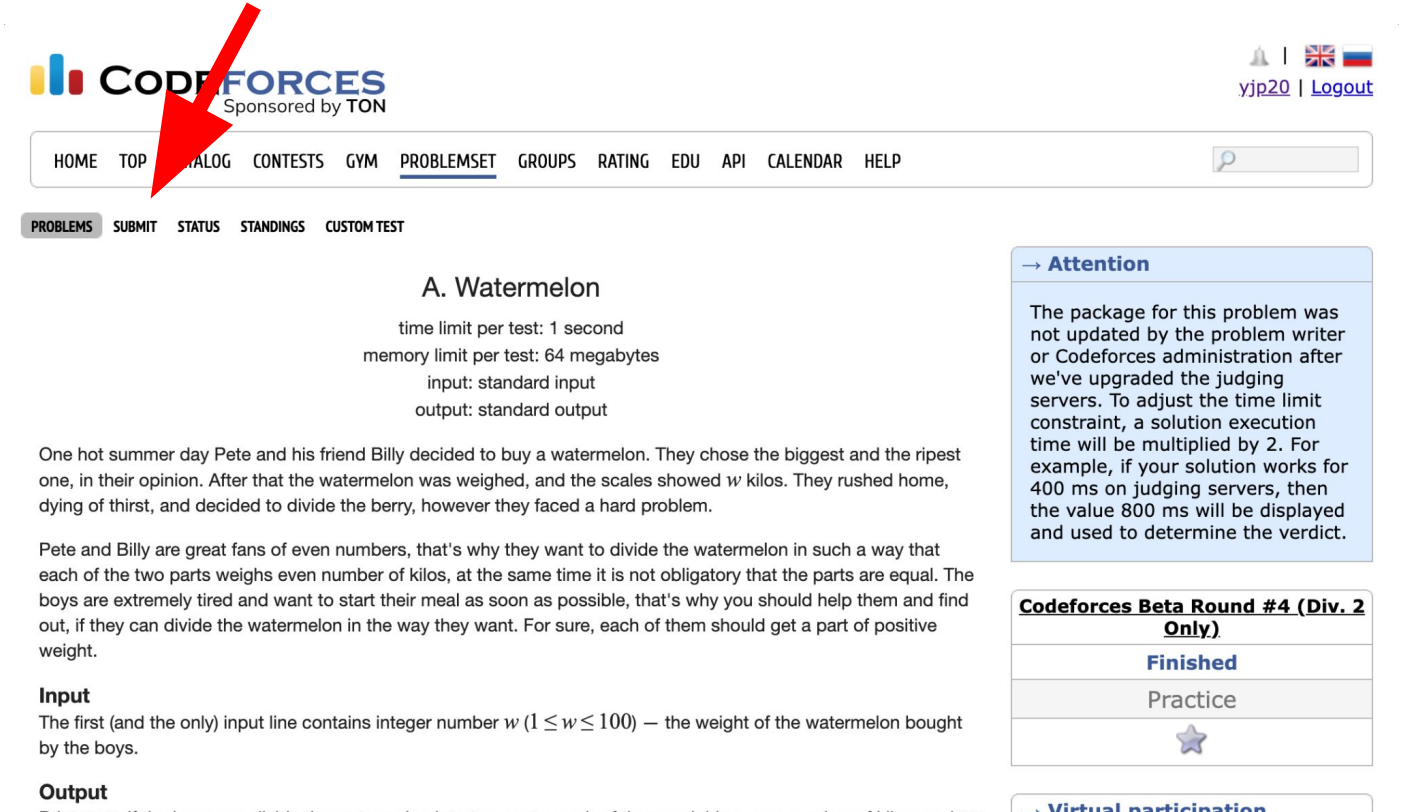
On the next *t* lines you will be given ***a*** ($0 \leq a \leq 99$) and ***b*** ($0 \leq b \leq 99$), the two numbers that you must add.

Output

Print the sum of **a** and **b**.

Input	Output
5 1 9 10 4 55 99 0 0 3 20	10 14 154 0 23

Submission


CODEFORCES
Sponsored by TON
yjp20 | Logout
HOME TOP CONTESTS GYM PROBLEMSET GROUPS RATING EDU API CALENDAR HELP
PROBLEMS SUBMIT STATUS STANDINGS CUSTOM TEST
A. Watermelon
time limit per test: 1 second
memory limit per test: 64 megabytes
input: standard input
output: standard output
One hot summer day Pete and his friend Billy decided to buy a watermelon. They chose the biggest and the ripest one, in their opinion. After that the watermelon was weighed, and the scales showed w kilos. They rushed home, dying of thirst, and decided to divide the berry, however they faced a hard problem.
Pete and Billy are great fans of even numbers, that's why they want to divide the watermelon in such a way that each of the two parts weighs even number of kilos, at the same time it is not obligatory that the parts are equal. The boys are extremely tired and want to start their meal as soon as possible, that's why you should help them and find out, if they can divide the watermelon in the way they want. For sure, each of them should get a part of positive weight.
Input
The first (and the only) input line contains integer number w ($1 \le w \le 100$) — the weight of the watermelon bought by the boys.
Output
→ Attention
The package for this problem was not updated by the problem writer or Codeforces administration after we've upgraded the judging servers. To adjust the time limit constraint, a solution execution time will be multiplied by 2. For example, if your solution works for 400 ms on judging servers, then the value 800 ms will be displayed and used to determine the verdict.
Codeforces Beta Round #4 (Div. 2 Only)
Finished
Practice

Submission


MAIN ACMSGURU | PROBLEMS SUBMIT STATUS STANDINGS CUSTOM TEST
Submit solution
Codeforces Beta Round #4 (Div. 2 Only)
Problem: 4A - Watermelon
Language: GNU G++17 7.3.0
Source code:
Switch off editor
Or choose file: Choose File No file chosen
Submit
Paste code
Or select file from your computer
→ Pay attention
Before contest
Educational Codeforces Round 142 (Rated for Div. 2)
12:30:28
Register now »
→ Settings
Show tags for unsolved problems
Hide solved problems
→ Last unsolved
Name Last submission
102501K Birdwatching 190264333
104114I Inadequate Operation 190160962
104030F Foreign Football 189806994
1665E MinimizOR 153085350
468C Hack it! 121291697
1539E Game with Cards 120149932
1400B RPG Protagonist 90942007
1028A Find Square 69503976
439A Devu, the Singer and Churu, the Joker 69503950
1257D Yet Another Monster Killing Problem 64834006
1248D2 The World Is Just a Programming Task (Hard Version) 63040657
791A Bear and Big Brother 59331726

Verdict

When you submit a solution, you will get one of the following verdicts:

- **Accepted/All Correct (AC)**: your program passed all test cases
- **Wrong Answer (WA)**: your program gave the wrong answer on at least one test case
- **Time Limit Exceeded (TLE)**: your program took too long to run on at least one test case
- **Runtime Error (RTE)**: your program compiled, but ran into an error on at least one test case
- **Memory Limit Exceeded (MLE)**: your program used too much memory on at least one test case
- **Compilation Error (CE)**: your program failed to compile

Verdict

189912509	Jan/20/2023 21:47 UTC-8	yjp20	B - Berry Battle	GNU C++17	Accepted	436 ms	22400 KB
189912401	Jan/20/2023 21:47 UTC-8	yjp20	B - Berry Battle	GNU C++17	Wrong answer on test 2	0 ms	4700 KB
189911254	Jan/20/2023 21:38 UTC-8	yjp20	B - Berry Battle	GNU C++17	Wrong answer on test 4	15 ms	4700 KB
189909966	Jan/20/2023 21:27 UTC-8	yjp20	E - Enigmatic Enumeration	GNU C++17	Accepted	468 ms	600 KB
189894298	Jan/20/2023 17:35 UTC-8	yjp20	E - Enigmatic Enumeration	GNU C++17	Wrong answer on test 6	15 ms	300 KB
189892884	Jan/20/2023 16:43 UTC-8	yjp20	A - Ace Arbiter	GNU C++17	Accepted	15 ms	0 KB

Input Parsing

1. Anatomy of a Problem Statement
2. **Input Parsing**
3. Time Complexity, Revisited!
4. Solving a Problem with Big O

Input Parsing

For the vast majority of competitive programming problems, your program will read in a certain input, and you'll be asked to do something to the input, and write a certain output.

Most competitive programming problems (and all that you'll see in this class) will involve reading input through the standard input channel (as opposed to file I/O)

For example, you might be asked to read in an array, and output the sum of elements in the array.

Input Parsing Example (Python)

Let’s say we were asked to read in a single integer, and print out the integer squared:

```
N = int(input())

print(N * N)
```

Input Parsing Example (C++)

Same example as before, but in C++:

```
#include <iostream>
using namespace std;
int main() {
    int N;
    cin >> N;
    cout << N * N << ‘\n’;
};
```

A. Watermelon

time limit per test: 1 second
memory limit per test: 64 megabytes
input: standard input
output: standard output

One hot summer day Pete and his friend Billy decided to buy a watermelon. They chose the biggest and the ripest one, in their opinion. After that the watermelon was weighed, and the scales showed w kilos. They rushed home, dying of thirst, and decided to divide the berry, however they faced a hard problem.

Pete and Billy are great fans of even numbers, that's why they want to divide the watermelon in such a way that each of the two parts weighs even number of kilos, at the same time it is not obligatory that the parts are equal. The boys are extremely tired and want to start their meal as soon as possible, that's why you should help them and find out, if they can divide the watermelon in the way they want. For sure, each of them should get a part of positive weight.

Input

The first (and the only) input line contains integer number w ($1 \le w \le 100$) — the weight of the watermelon bought by the boys.

Output

Print `YES`, if the boys can divide the watermelon into two parts, each of them weighing even number of kilos; and `NO` in the opposite case.

Examples

input	Copy
8	

output	Copy
YES	

Note

For example, the boys can divide the watermelon into two parts of 2 and 6 kilos respectively (another variant — two parts of 4 and 4 kilos).

Time Complexity, Revisited!

This shouldn't trigger your CS61B/170 PTSD. Hopefully.

1. Anatomy of a Problem Statement
2. Input Parsing
3. **Time Complexity, Revisited!**
4. Solving a Problem with Big O

Basics

Most competitive programming problems have a time limit of 1-2 seconds. This means that if your program takes more than this amount of time to run, on any of the test cases, it will return a Time Limit Exceeded verdict.

Most computers can compute around **10^7 operations per second**. This is an important number to keep in mind when solving competitive programming problems, though it can vary depending on factors such as programming language.

Big O Notation

When analyzing an algorithm, competitive programmers use **Big O Notation**, which is a way to quickly approximate the runtime of an algorithm without going into too much detail.

For competitive programming problems*, you can think about the Big-O complexity of an algorithm as being the number of simple operations the program must make in total, but only depending on any input variables. This will be less accurate than directly counting the number of operations, but is still fairly accurate and is much easier to count.

A program with a certain Big-O complexity is noted as O([complexity]). For example, a solution that depends on an input variable N might have $O(N^2)$ complexity.

*if you've taken CS 170, you've probably seen a more rigorous and mathematical definition of this; for competitive programming purposes this definition is fine

Big O Notation

If the time complexity of a solution contains terms that are strictly smaller than other terms in the expression, then the Big O complexity would only include the larger terms.

For example, a solution that completes $N^2 + N$ operations would have a Big O complexity of $O(N^2)$.

Big O Notation Examples

```
for(1, N) {
  print("Hello World")
}
```

Big O Notation Examples

The following code* has a complexity of O(N):

```
for(1, N) {
  print("Hello World")
}
```

*all code in the following slides are pseudo code

Big O Notation Examples

```
for(1, N) {
  for(1, N) {
    print("Hello World")
  }
}
```

Big O Notation Examples

The following code has a complexity of $O(N^2)$:

Since there are two nested loops, each iteration of the inner loop will take N operations, and this is run N times, which totals to N^2 operations.

```
for(1, N) {
  for(1, N) {
    print("Hello World")
  }
}
```

Big O Notation Examples

```
for(1, N) {
  print("Hello World")
}
print("Hello World")
```

Big O Notation Examples

The following code has a complexity of O(N):

Even though this code takes N + 1 operations to run, it will still be O(N) since 1 is constant and is not one of the input variables.

```
for(1, N) {
  print("Hello World")
}
print("Hello World")
```

Big O Notation Examples

```
for(1, N) {
  for(1, N) {
    print("Hello World")
  }
}
for(1, N) {
  print("Hello World")
}
```

Big O Notation Examples

The following code has a complexity of $O(N^2)$:

Even though this code takes $N^2 + N$ operations to run, it will still be $O(N^2)$ since N^2 is strictly larger than N and will dominate the time complexity calculation.

```
for(1, N) {
  for(1, N) {
    print("Hello World")
  }
}
for(1, N) {
  print("Hello World")
}
```

Using Big O Notation In Practice

Once you analyze the Big O time complexity of a solution, you can evaluate whether or not the solution will run fast enough on the input.

For example, if an input consists of a variable N that goes up to 10^5, an $O(N^2)$ solution would take roughly 10^{10} operations, which is too slow, while an $O(N)$ solution would likely be fast enough, as long as the constant factor wasn't too high.

Using Big O Notation In Practice

Competitive Tip

The following table describes approximately what complexity you need to solve a problem of size n if your algorithm has a certain complexity when the time limit is about 1 second.

Complexity	n
$O(\log n)$	$2^{(10^7)}$
$O(\sqrt{n})$	10^{14}
$O(n)$	10^7
$O(n \log n)$	10^6
$O(n\sqrt{n})$	10^5
$O(n^2)$	$5 \cdot 10^3$
$O(n^2 \log n)$	$2 \cdot 10^3$
$O(n^3)$	300
$O(2^n)$	24
$O(n2^n)$	20
$O(n^2 2^n)$	17
$O(n!)$	11

Table 5.1: Approximations of needed time complexities

Note that this is in no way a general rule – while complexity does not bother about constant factors, wall clock time does!

input size	required time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

Solving a Problem with Big O

Lights, camera, action!

1. Anatomy of a Problem Statement
2. Input Parsing
3. Time Complexity, Revisited!
4. **Solving a Problem with Big O**

Solving an Example Problem

Now, let's put our problem statement and time complexity knowledge to use! In solving Codeforces 1656A (Good Pairs), we'll demonstrate how to identify a suboptimal solution, then optimize it for a better time complexity.

Try out this problem yourself for a few minutes! See if you can find an efficient algorithm.

A. Good Pairs

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array $a_1, a_2, \ldots, a_n$ of positive integers. A *good pair* is a pair of indices (i, j) with $1 \le i, j \le n$ such that, for all $1 \le k \le n$, the following equality holds:

$$|a_i - a_k| + |a_k - a_j| = |a_i - a_j|,$$

where $|x|$ denotes the absolute value of x.

Find a good pair. Note that i can be equal to j.

Input
The input consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 10^5$) — the length of the array.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) where a_i is the i-th element of the array.

The sum of n for all test cases is at most $2 \cdot 10^5$.

Output
For each test case, print a single line with two space-separated indices i and j which form a good pair of the array. The case $i = j$ is allowed. It can be shown that such a pair always exists. If there are multiple good pairs, print any of them.

Example

input Copy

```
3
3
5 2 7
5
1 4 2 2 3
1
2
```

output Copy

```
2 3
1 2
1 1
```

Note
In the first case, for $i = 2$ and $j = 3$ the equality holds true for all k:

- $k = 1$: $|a_2 - a_1| + |a_1 - a_3| = |2 - 5| + |5 - 7| = 5 = |2 - 7| = |a_2 - a_3|$,
- $k = 2$: $|a_2 - a_2| + |a_2 - a_3| = |2 - 2| + |2 - 7| = 5 = |2 - 7| = |a_2 - a_3|$,
- $k = 3$: $|a_2 - a_3| + |a_3 - a_3| = |2 - 7| + |7 - 7| = 5 = |2 - 7| = |a_2 - a_3|$.

A Naive Solution

A straightforward solution is to consider all pairs of (i, j) and check if the equality holds for all values of k.

This can be done with a straightforward triple loop, but is very inefficient: $O(n^3)$.

Since n can go up to 10^5, we may need to check around $(10^5)^3 = 10^{15}$ equations in the worst case (which is way way bigger than 10^7!).

Even if we can solve an equation in a single operation, if we do 10^7 operations per second, it will take **3 years** for our code to finish a single test case!

```
t = input()
for (1, t) {
  n = input()
  a = input_array(n)
  pair = (-1, -1)
  for i in (1, N) {
    for j in (1, N) {
      if (equation true for all k in (1, N)) {
        pair = (i, j)
      }
    }
  }
  print(pair)
}
```

Exploration, Observation, and Optimization

The mathematical notation can be a bit hard to unpack. It might be easier to understand what the problem is really asking if we start by breaking down the equation.

$$|a_i - a_k| + |a_k - a_k| = |a_i - a_j|$$

The equation consists of 3 sub-expressions that look like $|x - y|$.

Q: What does $|x - y|$ represent?

Exploration, Observation, and Optimization

Q: What does $|x - y|$ represent?

A: It represents the *distance* between two numbers!

With this in mind, we can rephrase the question:

Find two numbers a_i and a_j in the array such that for all numbers in the array, the sum of the distance between it and a_i and the distance between it and a_j is equal to the distance between a_i and a_j.

Let's do some more exploring by plotting a sample input on a number line, and see how this notion of distance helps us:

https://www.desmos.com/calculator/zsn5ofuycz

Exploration, Observation, and Optimization

Observe through exploration that all a_k *must* be between a_i and a_j or else the LHS will add up to be more than the RHS.

Therefore, we can always pick i and j to be indices of the minimum and maximum values, and have a correct solution! Finding min and max is O(n).

```
t = input()
for (1, t) {
  n = input()
  a = input_array(n)
  i = j = 0
  for x in (1, n) {
    if (a[x] < a[i]) {
      i = x
    }
    if (a[x] > a[j]) {
      j = x
    }
  }
  print((i, j))
}
```

Another Example

Simon is a cashier and he is very good at giving change to customers who pay. He always makes sure that he gives them the least amount of coins needed. The coins in the cash register are your traditional coins: the **quarter, dime, nickel, and penny {25, 10, 5, 1}**. You will be given **T** customers to make change for. Each customer needs **N** cents in change. Can you figure out **how many coins** Simon gives to each customer? Constraints: **(1 ≤ N ≤ 10^12)** and **(1 ≤ t ≤ 20)**

Example Input

```
3   <-- T
17
50
9
```

Example Output

```
4
2
5
```

A Third Example!

You are given an array a with n integers. You can perform the following operation at most k times:

- Choose two indices i and j, in which $i \bmod k = j \bmod k$ ($1 \le i < j \le n$).
- Swap a_i and a_j

After performing all operations, you have to select k consecutive elements, and the sum of the k elements becomes your score. Find the maximum score you can get. ($1 \le k \le n \le 100$)

Sample I/O

input

```
5
3 2
5 6 0
1 1
7
5 3
7 0 4 0 4
4 2
2 7 3 4
3 3
1000000000 1000000000 999999997
```

output

```
11
7
15
10
2999999997
```

First thoughts!

- The problem asks you to pick where to start your k consecutive elements from
  - It's natural to think this is important!
- But, let's consider starting with an index a, and pick indices a, a+1, a+2, ... a+(k-1).
- If we take the remainder when dividing each number by k, all possible remainders are touched! (a + i != a + j iff i != j over Z/kZ, see binary operation)
- Consider: k = 4, a = 3
  - The sequence is 3, 4, 5, 6.
  - Mod 4: 3, 0, 1, 2

A realization...

- So regardless of where we start, for each of the k possible remainders, we choose only one index.
- And so, it's optimal to simply choose the highest element for each possible remainder!

```
m = [0] * k

for i in range(n):
    m[i % k] = max(m[i % k], a[i])

print(sum(m))
```

Attendance

Thanks for coming! If you're interested in participating in other parts of Competitive Coding @ Berkeley, come to our first meeting on **Thursday at 7:00PM!**

To get attendance credit, fill out the following form: