ICPC 2024 Internal Solution Sketch

C: Chopstick (39/39)

Just select the top two longest pairs of chopstick.

Bruteforce: Calculate area from every two different pairs.

Sort: Sort and select two maximum values.

Linear: Find two maximum values by looping array.

C: Chopstick

Common mistakes: Integer overflow!!!

(max answer 10^{18}, maximum value of *int* is ~ $2*10^{9}$).

Solution: Use *long long* or **python**.

E- Hidden Project (33/39)

For any given day, one is either do the project or do the normal work.

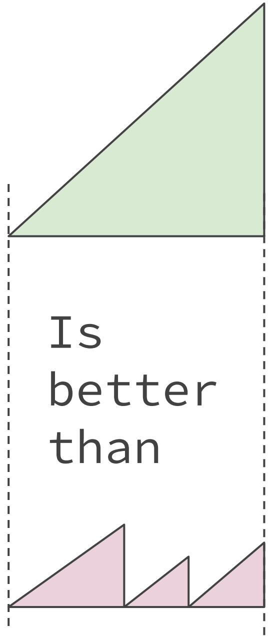
Observe that If we choose to do project for a total of K days, it is best to do the project on K consecutive days for K(K+1)/2 Bath instead of splitting into several project with shorter duration.

The best strategy is to either

1) conduct the project once, starting on the **1st** day and finishing on the **Nth** day, and receive **N(N+1)/2** baht, or

2) not conduct any project and receive **N*a** baht. Doing the project often will have the income start from 1 baht again, so doing the project as long as possible is always better.

Hence, the answer is **max(N*a, N(N+1)/2)**.



G: Ki Chang Jab Takkataen (8/39)

Greedy strategy: Catch nearest grasshoppers possible by shortest nets possible.

Detail: We sort all nets by length. Then, we try to catch grasshoppers from 1 to N respectively. If there is a net that can catch a grasshopper i, catch it with a shortest net, then disable the net. This way, we can find the earliest grasshoppers for any possible amount.

G: Ki Chang Jab Takkataen

Proof: The proof is left as an exercise for the contestant.

(Hint) Suppose that there is an optimal catching that does not follow our strategy i.e. not catching the same grasshoppers or not using the same net, prove that we can change nets or grasshoppers to match our strategy.

K: A Potion Shopping on This Wonderful World! (6/39)

This problem is just a knapsack problem, but the operation on weight is bitwise OR instead of addition!

For a set of stat types, we represent it as an integer. The ith bit will be 1 if and only if it is in the set. We call it **value** of a set.

Formally, the value **v** of set **s** is sum of 2^x for every stat type x that is in **s**.

Let **DP[*mask*]** be the least cost for obtaining stat types set ***mask***.

K: A Potion Shopping on This Wonderful World!

First, we set all DP of all possible sets to infinity, except DP[0] = 0

Then, for each item **i**, suppose the buff stat types set is value **x**, for each stat type set **s** in all possible stat type set, let it valued ***mask***. We update

```
DP[mask | x] = min(DP[mask | x],
                  DP[mask] + (cost of item i))
```

where "|" is bitwise OR.

K: A Potion Shopping on This Wonderful World!

After we update DP using all items, for each set **s**, we must also consider the DP value of its super set.

This can be done by iterating set with largest value to smallest value. For each set valued **Mask**, for each its element i, we update

$$\mathbf{DP[Mask - 2^i] = min(DP[Mask], DP[Mask - 2^i]).}$$

Time complexity: $O(4^K)$

Bonus: Find a solution with $O(3^K)$

H: Final Quiz (6/39)

I) **n = 3t**

There are **t** groups of three questions that you have to choose an answer for each group that cannot be the same as the previous one.

Answer = **k(k-1)^(t-1)**

H: Final Quiz

II) **n = 3t+1**

There are **t** groups of three questions and **1** group of one question. There are **t+1** ways to arrange the group. For **t+1** groups, you have to choose an answer to not be the same as the last one.

Answer = **(t+1)k(k−1)^t**

H: Final Quiz

III) **n = 3t+2**

There are two possible way to split the group.

A) **t** groups of three questions and **1** group of two questions.

There are **(t+2)(t+1)/2** ways to arrange the group. For **t+2** groups, you have to choose an answer to not be the same as the last one.

B) **t** groups of three questions and **2** group of one question.

There are **t+1** ways to arrange the group. For **t+1** groups, you have to choose an answer to not be the same as the last one.

Answer = **(t+1)k(k-1)^t + (t+2)(t+1)/2*k(k-1)^(t+1)**

A: Card Dealer Game (5/39)

Order of the deck doesn't matter. You just have to calculate that the blue card is pick even times.

You can calculate that by using dynamic programming.

```
DP[i] = DP[i-1]*p_i/(p_i+q_i) + (1-DP[i-1])*q_i/(p_i+q_i)
```

A: Card Dealer Game (cont.)

If ax = 1 mod p, then x = a^(p-2) mod p.

Or in python you can just use pow(a,-1,p)

I: Lulu and the Magical Array (5/39)

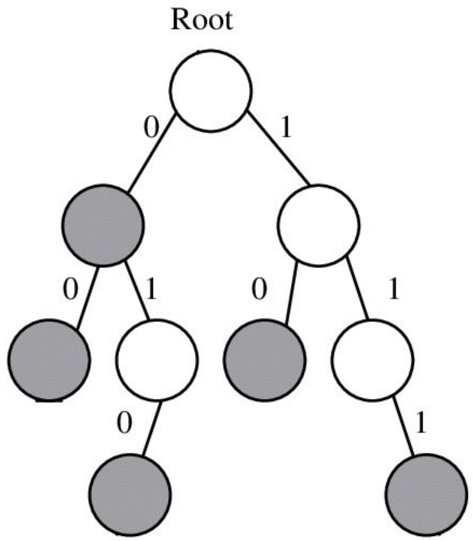
Creating new array b of length n-1 such that

b[i] = a[1] ⊕ a[i + 1] for i = 1 to n - 1

The answer is minimum of (minimum of array b and minimum between any XOR pair of array b)

I: Lulu and the Magical Array

You can do data structure like Trie to find minimum bitwise XOR pair in array b.



I: Lulu and the Magical Array

The simpler solution is just sort the array b and find

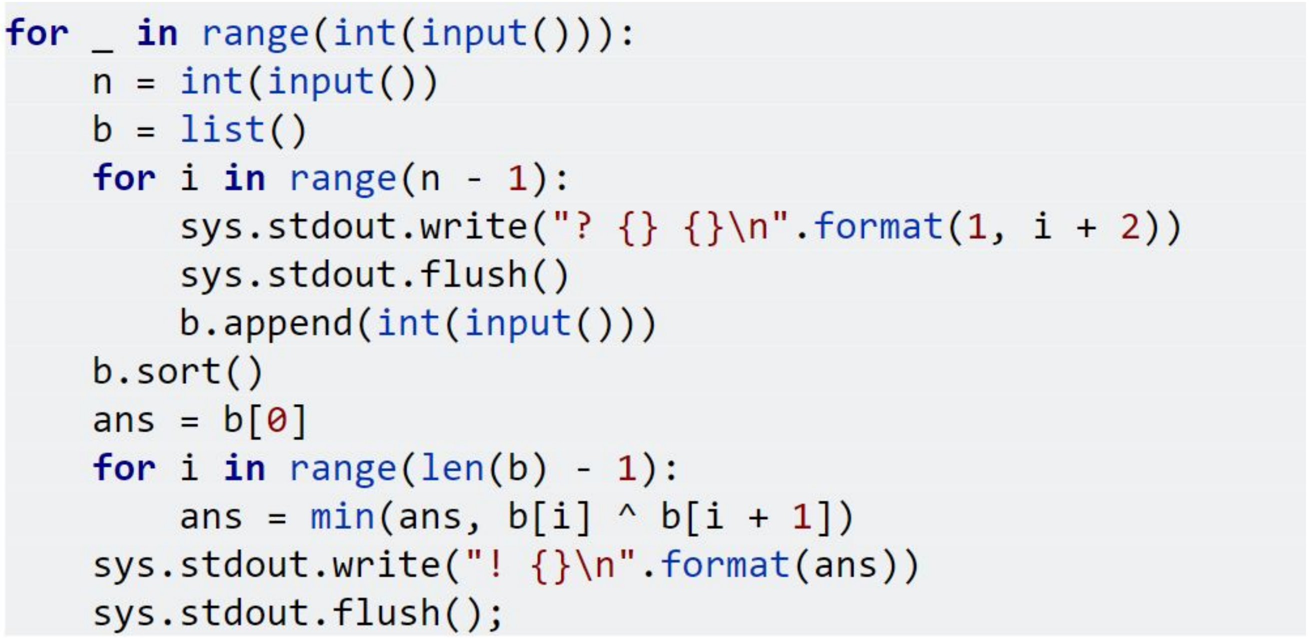
minimum of $b[i] \oplus b[i + 1]$ for $i = 1$ to $n - 2$

The intuition behind this is the adjacent pair in the sorted array has the most similar bit in significant order or just prove this equation

If $(a \le b \le c)$, $\min(a \oplus b, b \oplus c) \le a \oplus c$

I: Lulu and the Magical Array

The python code surely looks short, right?



```python
for _ in range(int(input())):
    n = int(input())
    b = list()
    for i in range(n - 1):
        sys.stdout.write("? {} {}\n".format(1, i + 2))
        sys.stdout.flush()
        b.append(int(input()))
    b.sort()
    ans = b[0]
    for i in range(len(b) - 1):
        ans = min(ans, b[i] ^ b[i + 1])
    sys.stdout.write("! {}\n".format(ans))
    sys.stdout.flush();
```

F: Portal Maintenance (1/39)

Given a weighted tree, you can add edge from u,v with weight dist(u,v)+c. Find the minimum cost to make the graph has euler cycle.

F: Portal Maintenance (1/39)

DFS on tree and match the nodes with odd degree to cover the whole tree.

Total cost = sum of all weight + c * # odd nodes / 2

B: Emma and the Pixie Dust (1/39)

Observe 1 : Combining k+1 elements together to get k pixie dust is the most effective way.

From Observe 1 : The problem becomes choose k+1 from m elements then find GCD of them, then add this to sum of the maximum n-k-1 elements from m-k-1 elements left

Observe 2 : Let a[1],a[2],...,a[k+1] be the **increasing** array of numbers that we want to merge and b[1],b[2],...,b[n-k-1] be the **increasing** array of numbers that we just add to the sum. The most effective way must have a[k]<b[1].

B: Emma and the Pixie Dust (Continue)

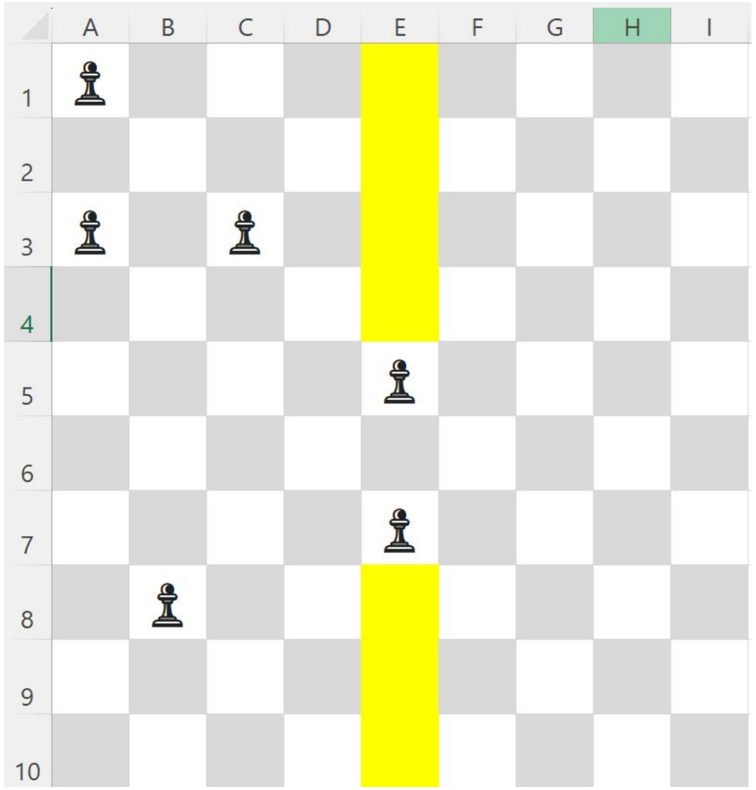
From Observe 2 : We divide into 2 cases.

Case 1 a[k+1]<b[1] : We know that array b must be the largest n-k-1 elements from original m elements.

Case 2 a[k+1]>b[1] : We know that array b must be the largest n-k elements that is not a[k+1]

From 2 cases, we just have to find the largest GCD of k+1 elements from the unchosen m-(n-k-1) elements. => O(nlogn)

J: Rook Placement (0/39)



Look at column E. The valid cells are prefix [1-4]E and suffix [8-10]E.
For the prefix and suffix, we need to count the number of empty rows. (Rows 2,4,9,10)

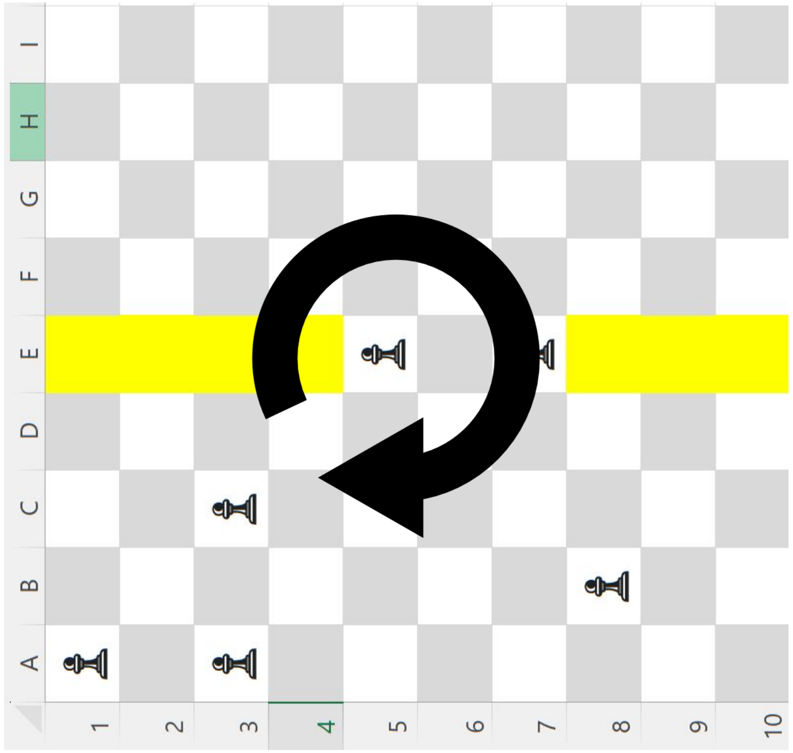
J: Rook Placement

- For each column, maintain empty prefix and suffix.
- maintain the index of empty rows.
- We need to update the answer when prefix and suffix changed or row became empty or non-empty.
- For example, when a row became non-empty, you need to subtract the answer equal to number of prefixes which covered that row and do the same for suffix.

J: Rook Placement

- To maintain prefix and suffix of each column, you simply need to maintain first and last elements in that column. (min, max) This can be done by std::set
- There are O(1) changes in prefix, suffix, and row. So you need a simple counting data structure to update the answer. e.g. (Fenwick Tree or Segment Tree)
- The range of input is large, so you may need to compress the coordinate of the input. But the well written solution with dynamic data structures can be passed as well.

J: Rook Placement



The previous explanation are for column counting, but row counting is symmetry. You can simply rotate the grid by 90 degree and run the same algorithm. (i.e. changing input from
(x, y) to (y, x)

J: Rook Placement

```
Even more simple, you can only consider prefix and
column counting only. Then transform the input in
4 cases and run the same algorithm on all of them.
-> r c (x, y)
-> r c (r - x + 1, y)
-> c r (y, x)
-> c r (c - y + 1, x)
Time complexity: O(n log(n))
```

D: Animal Circus (0/39)

If there is only one query you can solve this problem with binary search as you can check an answer if it valid or not because the more cage you use, the more animals you miss. Assume that our guessing for answer (number of cage) is **ANS** you can check validation of our guessing by doing following step.

D: Animal Circus

1. For every type of animal if the number of animals of that type **a_i** is greater than **ANS** can only take **ANS** animals of this type because there is only Ans cage we can use.

2. And for every type of animal the number of animals of that type is less than or equal to **ANS** can only take **a_i** animals.

D: Animal Circus

---

3. Let the number of animals we can take from the previous step equal **X**.

4. If X is greater than or equal to **K x ANS** We'll be sure that you can use **ANS** Cage.

5. Otherwise, you can't use **ANS** Cage

D: Animal Circus

These steps can be done naively in **O(N)** but in this problem, you must update several animals and answer multiple queries online (Because you need to XOR query by the previous answer).

Before that, you need to find out how to calculate **X** fast.

We know that if the number of animals exceeds **ANS** you can only take **ANS** animals so if we look closely, we'll have the equation of calculating **X**. Let **c** = number of **i** such that **a_i > ANS**, and **sum** = sum of **a_i** of animal type such that **a_i <= Ans** then **X = sum + c * Ans** we have the data structure that can quickly update and find range sum like segment tree. However, since the answer can be at most **2e14** our segment tree must be dynamic (i.e. you'll initialize new memory only if necessary).

D: Animal Circus

Then we can have time complexity for each query of **O(logMAX)** where **MAX** is the maximum possible answer which may be fast enough to pass but you can optimize time complexity to **O(logMAX)** by binary search and traverse in segment tree simultaneously. (There might be an integer overflow problem by using this method, so you need to set a bound when you traverse in the segment tree)

Authors

Problem	Author
A - Card Dealer Game	Mattanyu Tangngekkee
B - Emma and the Pixie dust	Poonyapat Sriroth
C - Chopsticks	Natapong Sriwatanasakdi
D - Animal Circus	Akarapon Watcharapalakorn
E - Hidden Project	Akarapon Watcharapalakorn
F - Portal Maintenance	Mattanyu Tangngekkee

Authors

Problem	**Author**
G - Ki Chang Jab Takkataen	Natapong Sriwatanasakdi
H - Final Quiz	Mattanyu Tangngekkee
I - Lulu and the Magical Array	Supakorn Kijwattanachai
J - Rook Placement	Mattanyu Tangngekkee Supakorn Kijwattanachai
K - A Potion Shopping On This Wonderful World!	Natapong Sriwatanasakdi