Phase 1 Remastered - Part 2

Progress Tracker

Leaderboard

Quiz Leaderboard

8.0 Hitting the Bits: And How to Talk to Computers

Expected Duration to Complete: 1 Week

Table of Contents

- Guess the Number
- Bear and Prime 100

Video Class

Contribution Technique

Contribution technique basically means calculating the contribution of each individual element to the answer. This technique is used in many problems, especially in problems related to counting or summing up some values. Let's see some examples to understand this technique better.

Sum of Pair Sums

Statement: Given an array a of size n, find $\sum_{i=1}^{n} \sum_{j=1}^{n} (a_i + a_j)$.

Constraints: $1 \leq n \leq 10^6$, $1 \leq a_i \leq 10^9$

Solution: Let's try to calculate the contribution of each element to the answer. For each element a_i, it will be added $2 \times n$ times to the answer. Once when a_i is the first element in the pair, and once when a_i is the second element in the pair. So, the contribution of a_i to the answer is $2 \times n \times a_i$. So, the answer is $\sum_{i=1}^{n} (2 \times n \times a_i) = 2 \times n \times \sum_{i=1}^{n} a_i$.

Sum of Subarray Sums

Statement: Given an array a of size n, find $\sum_{i=1}^{n} \sum_{j=i}^{n} (a_i + a_{i+1} + ... + a_j)$.

Constraints: $1 \leq n \leq 10^6$, $1 \leq a_i \leq 10^9$

Solution: Let's try to calculate the contribution of each element to the answer. For each element a_i, how many times will it be added to the answer? It is equal to the number of subarrays that contain a_i. And this is just $i \times (n - i + 1)$. So, the contribution of a_i to the answer is $a_i \times i \times (n - i + 1)$. So, the answer is $\sum_{i=1}^{n} (a_i \times i \times (n - i + 1))$.

Sum of Subset Sums

Statement: Given an array a of size n, find the sum of all subset sums of a. Output the answer modulo $10^9 + 7$.

Constraints: $1 \leq n \leq 10^6$, $1 \leq a_i \leq 10^9$

Solution: For each element a_i, it will be added 2^{n-1} times to the answer because there are 2^{n-1} subsets that contain a_i. So, the contribution of a_i to the answer is $2^{n-1} \times a_i$. So, the answer is $\sum_{i=1}^{n} (2^{n-1} \times a_i)$.

Product of Pair Products

Statement: Given an array a of size n, find $\prod_{i=1}^{n} \prod_{j=1}^{n} (a_i \times a_j)$. Output the answer modulo $10^9 + 7$.

Constraints: $1 \leq n \leq 10^6$, $1 \leq a_i \leq 10^9$

Solution: Let's try to calculate the contribution of each element to the answer. For each element a_i, it will be multiplied $2 \times n$ times to the answer. Once when a_i is the first element in the pair, and once when a_i is the second element in the pair. So, the contribution of a_i to the answer is $a_i^{2 \times n}$. So, the answer is $\prod_{i=1}^{n} (a_i^{2 \times n})$.

Notice that the product of all pair sums is hard to calculate because we can't just calculate the contributions directly. It's easier to calculate when the outer operation is the same as the operation on the elements i.e. sum of sums, product of products, XOR of XORs, etc.

XOR of Subarray XORs

Statement: Given an array a of size n, find $\bigoplus_{i=1}^{n} \bigoplus_{j=i}^{n} (a_i \bigoplus a_{i+1} \bigoplus ... \bigoplus a_j)$.

Constraints: $1 \leq n \leq 10^6$, $1 \leq a_i \leq 10^9$

Solution: Let's try to calculate the contribution of each element to the answer. For each element a_i, how many times will it be XORed to the answer? It is equal to the number of subarrays that contain a_i. And this is just $i \times (n - i + 1)$. Also, we know that $a_i \bigoplus a_i = 0$. So, the contribution of a_i to the answer is a_i if $i \times (n - i + 1)$ is odd, and 0 otherwise. So, we can get the answer by iterating over all elements and calculating the contribution of each element.

Sum of Max-Min over all Subsets

Statement: Given an array a of size n, find the sum of the difference between the maximum and minimum elements of all subsets of a. Output the answer modulo $10^9 + 7$.

Constraints: $1 \leq n \leq 2000$, $1 \leq a_i \leq 10^9$

Solution: First sort the array. Then, fix the minimum and maximum elements of the subset. The minimum element can be any element in the array, and the maximum element can be any element in the array that is greater than or equal to the minimum element. Let the minimum element be a_i and the maximum element be $a_j (j \geq i)$. Then,

the number of subsets that have a_i as the minimum element and a_j as the maximum element is 2^{j-i-1}. So, the contribution of a_i and a_j to the answer is $(a_j - a_i) \times 2^{j-i-1}$. So, the answer is $\sum_{i=1}^{n} \sum_{j=i+1}^{n} ((a_j - a_i) \times 2^{j-i-1})$.

Exercise: Try to solve the above problem when $n \leq 10^5$. Share your code with me after solving the problem.

Sum of Sum*Length over all Subarrays

Statement: Given an array a of size n, find the sum of (the sum of the subarray multiplied by the length of the subarray) over all subarrays of a. Output the answer modulo $10^9 + 7$.

Constraints: $1 \leq n \leq 10^6$, $1 \leq a_i \leq 10^9$

Solution: Let $S(l, r) = \sum_{i=l}^{r} i$, we can calculate it in $O(1)$. Let's try to calculate the contribution of each element to the answer. For each element a_i, let's add up the lengths of all subarrays that contain a_i. This is equal to $\sum_{j=1}^{i} \sum_{k=i}^{n} (k - j + 1) = \sum_{j=1}^{i} (\sum_{k=i}^{n} k + \sum_{k=i}^{n} -j + \sum_{k=i}^{n} 1) = \sum_{j=1}^{i} (S(i, n) - j \times (n - i + 1) + (n - i + 1)) = i \times S(i, n) - S(1, i) \times (n - i + 1) + i \times (n - i + 1)$. So, the contribution of a_i to the answer is $a_i \times (i \times S(i, n) - S(1, i) \times (n - i + 1) + i \times (n - i + 1))$. So, the answer is $\sum_{i=1}^{n} (a_i \times (i \times S(i, n) - S(1, i) \times (n - i + 1) + i \times (n - i + 1)))$.

Collapse With Style

Copy code

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e6 + 9, mod = 1e9 + 7;
int a[N];
int F(int n) { // 1 + 2 + ... + n
  return (1LL * n * (n + 1) / 2) % mod;
}
int S(int l, int r) { // l + (l + 1) + ... + r
  return (F(r) - F(l - 1) + mod) % mod;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }
  int ans = 0;
  // for (int i = 1; i <= n; i++) {
  //   int sum = 0;
  //   for (int j = i; j <= n; j++) {
  //     sum += a[j];
  //     ans += sum * (j - i + 1);
  //   }
```

```
  // }
  for (int i = 1; i <= n; i++) {
    int contrib = (1LL * S(i, n) * i % mod - 1LL * (n - i + 1) * S(1, i) % mod + mod) % mo
    contrib = (contrib + 1LL * (n - i + 1) * i % mod) % mod;
    ans += 1LL * a[i] * contrib % mod;
    ans %= mod;
  }
  cout << ans << '\n';
  return 0;
}
```

Sum of Number of Vowels in Substrings

Statement: Given a string s of size n, find the sum of the number of vowels in all substrings of s. Link to problem: link

Constraints: $1 \leq n \leq 10^5$, s_i is a lowercase English letter

Solution: Solve it yourself! Feel free to ask on Discord if you need help and please share your code with me after solving the problem.

Sum of Number of Inversions in Permutations

Statement: Given an integer n, find the sum of the number of inversions in all permutations of $[1, 2, ..., n]$. Output the answer modulo $10^9 + 7$.

Constraints: $1 \leq n \leq 10^5$

Solution: For each pair of elements $i < j$, how many permutations have i before j? Think about it like this: around 50% of the permutations have i before j, and around 50% of the permutations have j before i. So, for one pair the contribution is $\frac{n!}{2}$. And there are $\frac{n \times (n-1)}{2}$ pairs. So, the answer is $\frac{n \times (n-1) \times n!}{4}$. Pretty cool, right?

Sum of Number of Inversions in Subarrays

Statement: Given an array a of size n, find the sum of the number of inversions in all subarrays of a.

Constraints: $1 \leq n \leq 2000$, $1 \leq a_i \leq 10^9$

Solution: Solve it yourself! Feel free to ask on Discord if you need help and please share your code with me after solving the problem.

Bitwise Manipulation

We already learned about basic bitwise stuff (bitwise operations, how to set/unset/modify specific bits, bitsets, the xor trick) in previous classes. This time we will try to solve some harder problems.

The Bit Law

Always think bit by bit. If the solution is independent of the other bits, then we can solve the problem for each bit separately.

One good thing about this is that for each bit, the problem boils down to 0s and 1s and becomes much easier to solve.

Array Elimination

Problem Link: link

Think bit by bit to solve this problem.

Collapse With Style

Copy code

```
#include<bits/stdc++.h>
using namespace std;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int t; cin >> t;
  while (t--) {
    int n; cin >> n;
    vector<int> cnt(30, 0);
    int a[n + 1];
    for (int i = 1; i <= n; i++) {
      cin >> a[i];
      for (int j = 0; j < 30; j++) {
        if (a[i] >> j & 1) {
          cnt[j]++;
        }
      }
    }
    for (int k = 1; k <= n; k++) {
      bool ok = true;
      for (int j = 0; j < 30; j++) {
        if (cnt[j] % k != 0) {
          ok = false;
        }
      }
      if (ok) cout << k << ' ';
    }
    cout << '\n';
  }
  return 0;
}
```

Sum of All Numbers in an Array using The Bit Law

Statement: Given an array a of size n, find $\sum_{i=1}^{n} a_i$.

Constraints: $1 \leq n \leq 10^6, 0 \leq a_i \leq 10^9$

Solution: We can just add up all the numbers, but we will solve it using the bit law!

Any integer $\leq 10^9$ can be represented as $\sum_{i=0}^{30} (2^i \times b_i)$ where b_i is the i-th bit of the number.

For each bit k, we can calculate the number of numbers that have 1 in the k-th bit. Let's say there are $cnt_k[1]$ numbers with 1 in the k-th bit and $cnt_k[0]$ numbers with 0 in the k-th bit. Then, the sum of all numbers with 1 in the k-th bit is $cnt_k[1] \times 2^k$. So, the total sum of all numbers is $\sum_{k=0}^{30} (cnt_k[1] \times 2^k)$.

Collapse With Style Copy code

```
#include<bits/stdc++.h>
using namespace std;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  int ans = 0;
  vector<int> cnt(30, 0);
  for (int i = 1; i <= n; i++) {
    int x; cin >> x;
    for (int k = 0; k < 30; k++) {
      if (x >> k & 1) {
        cnt[k]++;
      }
    }
  }
  for (int k = 0; k < 30; k++) {
    ans += cnt[k] * (1 << k);
  }
  cout << ans << '\n';
  return 0;
}
```

Sum of Pair XORs

Statement: Given an array a of size n, find $\sum_{i=1}^{n} \sum_{j=1}^{n} (a_i \bigoplus a_j)$.

Constraints: $1 \leq n \leq 10^6, 0 \leq a_i \leq 10^9$

Solution: Notice that when we calculate xor of two numbers, the ith bit of the answer is 1 if the ith bit of the two numbers are different, and 0 otherwise. So the answer for the i-th bit is independent of other bits. And this is the case for other bitwise operations too. So, we can solve this problem bit by bit. For each bit k, we can calculate the number of pairs that have different bits in this position. Let's say there are $cnt_k[0]$ numbers in the array with 0 in the k-th bit, and $cnt_k[1]$ numbers with 1 in the k-th bit. Then, the number of pairs that have different bits in the k-th bit is $cnt_k[0] \times cnt_k[1]$. Also, any integer $\leq 10^9$ can be represented as $\sum_{i=0}^{30} (2^i \times b_i)$ where b_i is the i-th

bit of the number. So, in the total answer, the k-th bit will be set $cnt_k[0] \times cnt_k[1]$ times adding up to $cnt_k[0] \times cnt_k[1] \times 2^k$.

So, the contribution of the k-th bit to the answer is $cnt_k[0] \times cnt_k[1] \times 2^k$. So, the answer is $\sum_{k=0}^{30} (cnt_k[0] \times cnt_k[1] \times 2^k)$.

Collapse With Style

Copy code

```
#include<bits/stdc++.h>
using namespace std;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  int a[n + 1];
  int ans = 0;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }
  // for (int i = 1; i <= n; i++) {
  //   for (int j = 1; j <= n; j++) {
  //     ans += a[i] ^ a[j];
  //   }
  // }
  int cnt[30][2];
  memset(cnt, 0, sizeof cnt);
  for (int i = 1; i <= n; i++) {
    for (int k = 0; k < 30; k++) {
      if (a[i] >> k & 1) {
        cnt[k][1]++;
      }
      else {
        cnt[k][0]++;
      }
    }
  }
  for (int k = 0; k < 30; k++) {
    int contrib = cnt[k][0] * cnt[k][1] * 2;
    ans += contrib * (1 << k);
  }
  cout << ans << '\n';
  return 0;
}
```

Sum of Pair ANDs

Statement: Given an array a of size n, find $\sum_{i=1}^{n} \sum_{j=1}^{n} (a_i \& a_j)$.

Constraints: $1 \leq n \leq 10^6$, $0 \leq a_i \leq 10^9$

Solution: Notice that when we calculate AND of two numbers, the ith bit of the answer is 1 if the ith bit of the two numbers are both 1, and 0 otherwise. So the answer for the i-th bit is independent of other bits.

So, we can solve this problem bit by bit. For each bit k, we can calculate the number of pairs that have 1 in this position. Let's say there are $cnt_k[1]$ numbers in the array with 1 in the k-th bit. Then, the number of pairs that have 1 in the k-th bit is $cnt_k[1]^2$.

So, the contribution of the k-th bit to the answer is $cnt_k[1]^2 \times 2^k$. So, the answer is $\sum_{k=0}^{30} (cnt_k[1]^2 \times 2^k)$.

Collapse With Style

Copy code

```
#include<bits/stdc++.h>
using namespace std;

int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  int a[n + 1];
  int ans = 0;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }
  // for (int i = 1; i <= n; i++) {
  //   for (int j = 1; j <= n; j++) {
  //     ans += a[i] & a[j];
  //   }
  // }
  int cnt[30][2];
  memset(cnt, 0, sizeof cnt);
  for (int i = 1; i <= n; i++) {
    for (int k = 0; k < 30; k++) {
      if (a[i] >> k & 1) {
        cnt[k][1]++;
      }
      else {
        cnt[k][0]++;
      }
    }
  }
  for (int k = 0; k < 30; k++) {
    int contrib = cnt[k][1] * cnt[k][1];
```

```
    ans += contrib * (1 << k);
  }
  cout << ans << '\n';
  return 0;
}
```

Sum of Pair ORs

Statement: Given an array a of size n, find $\sum_{i=1}^{n} \sum_{j=1}^{n} (a_i | a_j)$.

Constraints: $1 \leq n \leq 10^6$, $0 \leq a_i \leq 10^9$

Solution: Solve it yourself!

Sum of Subset XORs

Statement: Given an array a of size n, find the sum of all subset XORs of a.

Constraints: $1 \leq n \leq 10^6$, $0 \leq a_i \leq 10^9$

Solution: Let's solve for each bit separately. For each bit k, we can calculate the number of subsets that have 1 in this position. Let's say there are $cnt_k[1]$ numbers in the array with 1 in the k-th bit.

A subset XOR will have 1 in the k-th bit if the number of elements in the subset with 1 in the k-th bit is odd. So, the number of subsets that have 1 in the k-th bit is = number of ways to choose an odd number of elements with 1 in the k-th bit $\times$ number of ways to choose elements with 0 in the k-th bit = $2^{cnt_k[1]-1} \times 2^{cnt_k[0]} = 2^{cnt_k[1]+cnt_k[0]-1}$.

So, the contribution of the k-th bit to the answer is $2^{cnt_k[1]+cnt_k[0]-1} \times 2^k$. So, the answer is $\sum_{k=0}^{30} (2^{cnt_k[1]+cnt_k[0]-1} \times 2^k)$.

Collapse With Style

Copy code

```
#include<bits/stdc++.h>
using namespace std;

const int mod = 1e9 + 7;
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  int a[n + 1];
  int ans = 0;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }
  int cnt[30][2];
  memset(cnt, 0, sizeof cnt);
  for (int i = 1; i <= n; i++) {
    for (int k = 0; k < 30; k++) {
```

```
      if (a[i] >> k & 1) {
        cnt[k][1]++;
      }
      else {
        cnt[k][0]++;
      }
    }
  }
  int pw2 = 1;
  for (int i = 1; i < n; i++) {
    pw2 = pw2 * 2 % mod;
  }
  for (int k = 0; k < 30; k++) {
    if (!cnt[k][1]) continue; // if no elements have 1 in the k-th bit then the contributi
    int contrib = pw2; // 2^(n - 1)
    ans += 1LL * contrib * (1 << k) % mod;
    ans %= mod;
  }
  cout << ans << '\n';
  return 0;
}
```

Sum of Subset ANDs

Statement: Given an array a of size n, find the sum of all subset ANDs of a.

Constraints: $1 \leq n \leq 10^6, 0 \leq a_i \leq 10^9$

Solution: Solve it yourself! Feel free to ask on Discord if you need help and please share your code with me after solving the problem.

Sum of Subset ORs

Statement: Given an array a of size n, find the sum of all subset ORs of a.

Constraints: $1 \leq n \leq 10^6, 0 \leq a_i \leq 10^9$

Solution: Solve it yourself! Feel free to ask on Discord if you need help and please share your code with me after solving the problem.

Number of Subarrays with XOR 0

Statement: Given an array a of size n, find the number of subarrays of a with XOR 0.

Constraints: $1 \leq n \leq 10^6, 0 \leq a_i \leq 10^9$

Solution: Compute the prefix XOR array p. Then, for each i, we need to find the number of $j < i$ such that $p_i = p_j$. We can use a map to store the number of times each value appears in the prefix XOR array. Then, for each i, we can get the number of j such that $p_i = p_j$ in $O(\log n)$ using the map. So, the total time complexity is $O(n \log n)$.

Expected Value (Basic)

Expected value is just the average of all possible outcomes. So basically, Expected Value = $\frac{total\ sum\ of\ all\ possible\ outcomes}{number\ of\ possible\ outcomes}$.

For example, if we roll a fair die, what is the expected value of the number we get? The possible outcomes are $1, 2, 3, 4, 5, 6$. So, Expected Value = $\frac{1+2+3+4+5+6}{6} = 3.5$.

There are wayyy more to expected values but as a beginner, you don't need to know more than this. Also expected values more related to probabilities that you have learned in school. You can learn about basic probabilities from here and here. Also, there are ways to calculate expected values without generating all possible outcomes by using probabilities and linearity of expectation. We will learn more about them in the future.

Heads in Two Coin Flips

Statement: If we roll a fair coin twice, what is the expected number of heads we get?

Solution: Let's try to generate all possible outcomes. The possible outcomes are HH, HT, TH, TT. So, Expected Value = $\frac{2+1+1+0}{4} = 1.0$.

A Random Number in an Array

Statement: Given an array a of size n, if we select a random number from the array, what is the expected value of the number we get?

Constraints: $1 \leq n \leq 10^6$, $0 \leq a_i \leq 10^9$

Solution: Let's try to generate all possible outcomes. The possible outcomes are $a_1, a_2, ..., a_n$. So, Expected Value = $\frac{a_1+a_2+...+a_n}{n}$.

Sum of a Subarray

Statement: Given an array a of size n, if we select a random subarray from the array, what is the expected value of the sum of the subarray?

Constraints: $1 \leq n \leq 10^6$, $0 \leq a_i \leq 10^9$

Solution: Let's try to generate all possible outcomes. The possible outcomes are all subarrays of a. So, Expected Value = $\frac{\sum_{i=1}^{n}\sum_{j=i}^{n}(a_i+a_{i+1}+...+a_j)}{\frac{n\times(n+1)}{2}}$. To calculate the sum of all subarrays, we can use the contribution technique we learned earlier.

Interactive Problems (Basic)

Tutorial: link

Interactive problems are problems where you have to interact with the judge to get the answer. For example, you might have to ask the judge questions and get a reply from the judge. Based on the reply, you can ask more questions. Or, you might have to submit an answer and get the final verdict from the judge. Let's see some examples.

Guess the Number

Problem Link: link

This is the most common interactive problem. You have to guess a hidden number between 1 and $n = 10^6$. You can ask the judge your guess and the judge will reply with one of the following:

- `<` if your guess is less than the number
- `>=` if your guess is greater than or equal to the number

Part 1: You can guess up to 10^6 times. Now solve the problem.

Strategy: Just guess all numbers from 1 to 10^6 one by one. When the judge replies with `<` for the first time when you guess x, then the hidden number is $x - 1$.

Collapse With Style Copy code

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    bool found = false;
    for (int i = 1; i <= 1000000; i++) {
        cout << i << endl;
        string s;
        cin >> s;
        if (s == "<") {
            cout << "! " << i-1 << endl;
            found = true;
            break;
        }
    }
    if (!found) {
        cout << "! 1000000" << endl;
    }
    return 0;
}
```

Let's say the answer is 4. Then, the interaction will be like this:

Collapse With Style Copy code

```
You: 1
Judge: >=
You: 2
Judge: >=
You: 3
Judge: >=
```

```
You: 4
Judge: >=
You: 5
Judge: <
You: ! 4
```

Things to keep in mind:

- Always print a newline after each guess.
- Don't forget to flush the output after each guess. In C++, you can do this by using `cout.flush()` or just printing `endl` will automatically flush the output.
- If you don't flush the output, you might get a wrong answer or runtime error or something like `Idleness Limit Exceeded` verdict.
- Flush the output even after printing the final output (i.e. `! 4` in the above example).
- The judge *normally* fixes the hidden number before the interaction starts. So, if you ask the judge the same question multiple times, you will get the same reply.
- If you ask too many questions, you might get a `Wrong Answer` verdict. So, be careful about that.
- If you ask invalid questions or questions in wrong format, you might get any verdict. So, be careful about that too. Even do not print any extra spaces or extra newlines.
- To test the code locally, you will have to act as the judge and you will have to reply to the guesses. For this, you will have to run the code in the terminal and interact with it.

Part 2: You can guess up to 25 times. Now solve the problem.

Strategy: We can use binary search to guess the number. The total number of guesses will be $\log_2 10^6 \approx 20$.

Collapse With Style Copy code

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int l = 1, r = 1000000, ans = -1;
    while (l <= r) {
        int mid = (l + r) / 2;
        cout << mid << endl;
        string s;
        cin >> s;
        if (s == "<") {
            r = mid - 1;
        } else {
            ans = mid;
            l = mid + 1;
        }
    }
    cout << "! " << ans << endl;
```

```
    return 0;
}
```

Let's say the answer is 4 and $n = 20$ (for example's sake). Then, the interaction will be like this:

Collapse With Style Copy code

```
// l = 1, r = 20, mid = 10
You: 10
Judge: <
// l = 1, r = 9, mid = 5
You: 5
Judge: <
// l = 1, r = 4, mid = 2
You: 2
Judge: >=
// l = 3, r = 4, mid = 3
You: 3
Judge: >=
// l = 4, r = 4, mid = 4
You: 4
Judge: >=
// l = 5, r = 4, l > r, exit loop
You: ! 4
```

Bear and Prime 100

Problem Link: link

Solve it yourself!

Collapse With Style Copy code

```
#include<bits/stdc++.h>
using namespace std;

bool query(int x) {
  cout << x << endl;
  string s; cin >> s;
  return s == "yes";
}
bool is_prime(int n) {
  for (int i = 2; i < n; i++) {
    if (n % i == 0) {
      return false;
    }
  }
```

```
  return true;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  if (query(2 * 2) or query(3 * 3) or query(5 * 5) or query(7 * 7)) {
    cout << "composite" << endl;
    return 0;
  }
  int prime_divs = 0;
  int cnt = 0;
  for (int i = 2; i <= 50; i++) {
    if (is_prime(i)) {
      cnt++;
      prime_divs += query(i);
    }
  }
  // cout << cnt << '\n'; // cnt = 15
  if (prime_divs <= 1) {
    cout << "prime" << endl;
  }
  else {
    cout << "composite" << endl;
  }
  return 0;
}
```