Tricks of the Trade in Combinatorics and Arithmetic

Zachary Friggstad

Programming Club Meeting

Fast Exponentiation

Given integers a, b with $b \geq 0$, compute a^b *exactly*.

Fast Exponentiation

Given integers a, b with $b \geq 0$, compute a^b *exactly*.

Naively, takes $b - 1$ multiplications. There is a way using only $O(\log b)$ multiplications.

Note, most applications of this have you take the answer modulo some value m so the numbers don't get too big. This is an essential subroutine in many ciphers, including RSA and Diffie-Hellman.

If $b = 2^k$ then just use repeated squaring:

$$a \to a^2 \to a^4 \to a^8 \to \ldots \to a^{2^i} \to \ldots \to a^{2^k}.$$

Takes $k = \log_2 b$ multiplications.

If $b = 2^k$ then just use repeated squaring:

$$a \to a^2 \to a^4 \to a^8 \to \ldots \to a^{2^i} \to \ldots \to a^{2^k}.$$

Takes $k = \log_2 b$ multiplications.

In general, write $b = \sum_{i=0}^{k} c_i \cdot 2^i$ where $c_i \in \{0, 1\}$. Use repeated squaring to iteratively compute a^{2^i}. If $c_i = 1$ then multiply a^{2^i} into the answer.

If $b = 2^k$ then just use repeated squaring:

$$a \to a^2 \to a^4 \to a^8 \to \ldots \to a^{2^i} \to \ldots \to a^{2^k}.$$

Takes $k = \log_2 b$ multiplications.

In general, write $b = \sum_{i=0}^{k} c_i \cdot 2^i$ where $c_i \in \{0, 1\}$. Use repeated squaring to iteratively compute a^{2^i}. If $c_i = 1$ then multiply a^{2^i} into the answer.

```
//compute a^b mod m
int fmodexp(int a, int b, int m) {
    int ans = 1, pow2 = a; //pow2 = a^{2^i} after i iterations
    while (b) {
        if (b & 1) ans = (ans * pow2) % m; //if c_i == 1
        pow2 = (pow2*pow2) % m;
        b >>= 1;
    }
    return ans;
}
```

Linear Recurrences (yawn)

Recall the Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, \ldots$.

Linear Recurrences (yawn)

Recall the Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, \ldots$.

$$\mathbf{fib}(n) = \begin{cases} n & \text{if } n \in \{0, 1\} \\ \mathbf{fib}(n-1) + \mathbf{fib}(n-2) & \text{if } n \geq 2 \end{cases}$$

Linear Recurrences (yawn)

Recall the Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, \ldots$.

$$\mathbf{fib}(n) = \begin{cases} n & \text{if } n \in \{0, 1\} \\ \mathbf{fib}(n-1) + \mathbf{fib}(n-2) & \text{if } n \geq 2 \end{cases}$$

This is a linear recurrence. Apart from some base cases, the recurrence is just a linear combination of some previous terms.

Linear Recurrences (yawn)

Recall the Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, \ldots$.

$$\mathbf{fib}(n) = \begin{cases} n & \text{if } n \in \{0, 1\} \\ \mathbf{fib}(n-1) + \mathbf{fib}(n-2) & \text{if } n \geq 2 \end{cases}$$

This is a linear recurrence. Apart from some base cases, the recurrence is just a linear combination of some previous terms.

Another example:

$$g(n) = \begin{cases} 1 & \text{if } n = 0 \\ -3 & \text{if } n = 1 \\ 7 & \text{if } n = 2 \\ 2f(n-1) - 3 \cdot f(n-3) & \text{if } n \geq 3 \end{cases}$$

Compute the n'th value of a recurrence!

Compute the n'th value of a recurrence!

Easy, in $O(k \cdot n)$ time where k is the "order" (i.e. number of terms back it looks). Use DP to build the table up to the n'th term.

Compute the n'th value of a recurrence!

Easy, in $O(k \cdot n)$ time where k is the "order" (i.e. number of terms back it looks). Use DP to build the table up to the n'th term.

We can do it **much** faster. Look!

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} fib(n) \\ fib(n+1) \end{pmatrix} = \begin{pmatrix} fib(n+1) \\ fib(n+2) \end{pmatrix}$$

Compute the n'th value of a recurrence!

Easy, in $O(k \cdot n)$ time where k is the "order" (i.e. number of terms back it looks). Use DP to build the table up to the n'th term.

We can do it **much** faster. Look!

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} fib(n) \\ fib(n+1) \end{pmatrix} = \begin{pmatrix} fib(n+1) \\ fib(n+2) \end{pmatrix}$$

Therefore,

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} fib(n) \\ fib(n+1) \end{pmatrix}$$

The column vector on the left are the **base cases**.

Compute the n'th value of a recurrence!

Easy, in $O(k \cdot n)$ time where k is the "order" (i.e. number of terms back it looks). Use DP to build the table up to the n'th term.

We can do it **much** faster. Look!

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} fib(n) \\ fib(n+1) \end{pmatrix} = \begin{pmatrix} fib(n+1) \\ fib(n+2) \end{pmatrix}$$

Therefore,

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} fib(n) \\ fib(n+1) \end{pmatrix}$$

The column vector on the left are the **base cases**.

Use fast exponentiation to compute $fib(n)$ in $O(\log n)$ arithmetic operations!

Recall this recurrence.

$$g(n) = \begin{cases} 1 & \text{if } n = 0 \\ -3 & \text{if } n = 1 \\ 7 & \text{if } n = 2 \\ 2f(n-1) - 3 \cdot f(n-3) & \text{if } n \geq 3 \end{cases}$$

Recall this recurrence.

$$g(n) = \begin{cases} 1 & \text{if } n = 0 \\ -3 & \text{if } n = 1 \\ 7 & \text{if } n = 2 \\ 2f(n-1) - 3 \cdot f(n-3) & \text{if } n \geq 3 \end{cases}$$

Then

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & 0 & 2 \end{pmatrix}^n \cdot \begin{pmatrix} 1 \\ -3 \\ 7 \end{pmatrix} = \begin{pmatrix} g(n) \\ g(n+1) \\ g(n+2) \end{pmatrix}$$

Recall this recurrence.

$$g(n) = \begin{cases} 1 & \text{if } n = 0 \\ -3 & \text{if } n = 1 \\ 7 & \text{if } n = 2 \\ 2f(n-1) - 3 \cdot f(n-3) & \text{if } n \geq 3 \end{cases}$$

Then

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & 0 & 2 \end{pmatrix}^n \cdot \begin{pmatrix} 1 \\ -3 \\ 7 \end{pmatrix} = \begin{pmatrix} g(n) \\ g(n+1) \\ g(n+2) \end{pmatrix}$$

Just use fast exponentiation again! Note, if they want the answer mod m, then make sure to take mods after each arithmetic step in the computation.

Recall this recurrence.

$$g(n) = \begin{cases} 1 & \text{if } n = 0 \\ -3 & \text{if } n = 1 \\ 7 & \text{if } n = 2 \\ 2f(n-1) - 3 \cdot f(n-3) & \text{if } n \geq 3 \end{cases}$$

Then

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & 0 & 2 \end{pmatrix}^n \cdot \begin{pmatrix} 1 \\ -3 \\ 7 \end{pmatrix} = \begin{pmatrix} g(n) \\ g(n+1) \\ g(n+2) \end{pmatrix}$$

Just use fast exponentiation again! Note, if they want the answer mod m, then make sure to take mods after each arithmetic step in the computation.

In general, number of arithmetic operations is $O(k^3 \log n)$ where k is the order of the recurrence (using $O(k^3)$ matrix multiplication).

Permutations

A permutation of a set is just a rearrangement of it. Let's just talk about permutations of $\{0, \ldots, n-1\}$.

One way to express a permutation π:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 4 & 1 & 6 & 7 & 5 & 0 & 2 \end{pmatrix}$$

Read this like "0 goes to 3" and "1 goes to 4", etc.

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 4 & 1 & 6 & 7 & 5 & 0 & 2 \end{pmatrix}$$

Can break a permutation down into cycles. Track the trajectory of an item as it repeatedly gets permuted.

$$0 \to 3 \to 6 \to 0$$

$$1 \to 4 \to 7 \to 2 \to 1$$

$$5 \to 5$$

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 4 & 1 & 6 & 7 & 5 & 0 & 2 \end{pmatrix}$$

Can break a permutation down into cycles. Track the trajectory of an item as it repeatedly gets permuted.

$$0 \to 3 \to 6 \to 0$$

$$1 \to 4 \to 7 \to 2 \to 1$$

$$5 \to 5$$

Write this compactly as

$$\pi = (0\ 3\ 6) \cdot (1\ 4\ 7\ 2) \cdot (5)$$

$$\pi = (0\ 3\ 6) \cdot (1\ 4\ 7\ 2) \cdot (5)$$

Permutations are naturally represented as an array

```
//initialization like this possible in c++11
vector<int> pi = {3, 4, 1, 6, 7, 5, 0, 2};
```

$$\pi = (0\ 3\ 6) \cdot (1\ 4\ 7\ 2) \cdot (5)$$

Permutations are naturally represented as an array

```
//initialization like this possible in c++11
vector<int> pi = {3, 4, 1, 6, 7, 5, 0, 2};
```

Can find all cycles in $O(n)$ time. Pick an item not in a marked cycle, and follow it through its cycle.

$$\pi = (0\ 3\ 6) \cdot (1\ 4\ 7\ 2) \cdot (5)$$

Permutations are naturally represented as an array

```
//initialization like this possible in c++11
vector<int> pi = {3, 4, 1, 6, 7, 5, 0, 2};
```

Can find all cycles in $O(n)$ time. Pick an item not in a marked cycle, and follow it through its cycle.

```
vector<bool> seen(n, false);
vector<vector<int>> cycles;
for (auto x : pi) {
    if (seen[x]) continue;
    vector<int> cyc;
    while (!seen[x]) { // follow x around the cycle
        cyc.push_back(x);
        seen[x] = true;
        x = perm[x];
    }
    cycles.push_back(cyc);
}
```

Tip!

When solving a problem involving a permutation, looking at the cycle decomposition may help!

Tip!

When solving a problem involving a permutation, looking at the cycle decomposition may help!

Problem

Given a permutation π, how many times do you have to apply π before everything ends up back in its original position?

Tip!

When solving a problem involving a permutation, looking at the cycle decomposition may help!

Problem

Given a permutation π, how many times do you have to apply π before everything ends up back in its original position?

Example

Suppose you have a shuffling machine that always permutes the cards the same way. How many times to you have to apply this shuffling until the deck returns to its original arrangement?

If π is just a cycle of length n, it takes n steps.

If π is just a cycle of length n, it takes n steps.

e.g. $\pi = (0\ 1\ 2\ 3\ 4\ 5)$. After 6 applications of π, every item returns back to its start location.

If π is just a cycle of length n, it takes n steps.

e.g. $\pi = (0\ 1\ 2\ 3\ 4\ 5)$. After 6 applications of π, every item returns back to its start location.

If π is the product of a bunch of cycles, the answer is the smallest integer m that is a multiple of all cycle lengths: the least-common multiple of all cycle lengths!

If π is just a cycle of length n, it takes n steps.

e.g. $\pi = (0\ 1\ 2\ 3\ 4\ 5)$. After 6 applications of π, every item returns back to its start location.

If π is the product of a bunch of cycles, the answer is the smallest integer m that is a multiple of all cycle lengths: the least-common multiple of all cycle lengths!

e.g. $\pi = (0\ 3\ 6) \cdot (1\ 4\ 7\ 2) \cdot (5)$.

If π is just a cycle of length n, it takes n steps.

e.g. $\pi = (0\ 1\ 2\ 3\ 4\ 5)$. After 6 applications of π, every item returns back to its start location.

If π is the product of a bunch of cycles, the answer is the smallest integer m that is a multiple of all cycle lengths: the least-common multiple of all cycle lengths!

e.g. $\pi = (0\ 3\ 6) \cdot (1\ 4\ 7\ 2) \cdot (5)$.

12 applications to get every item back to its starting position.

Another Way to Say It

$\pi^{12} = (0) \cdot (1) \cdot (2) \cdot (3) \cdot (4) \cdot (5) \cdot (6) \cdot (7)$

Suppose π, σ are permutations. Get a new permutation, denoted $\pi \circ \sigma$, by first permuting according to σ and then according to π.

Example:

- $\pi = (0\ 1\ 4) \cdot (2\ 3)$
- $\sigma = (0) \cdot (1\ 2\ 4\ 3)$
- $\pi \circ \sigma = (0\ 1\ 3\ 4\ 2)$

Suppose π, σ are permutations. Get a new permutation, denoted $\pi \circ \sigma$, by first permuting according to σ and then according to π.

Example:

- $\pi = (0\ 1\ 4) \cdot (2\ 3)$
- $\sigma = (0) \cdot (1\ 2\ 4\ 3)$
- $\pi \circ \sigma = (0\ 1\ 3\ 4\ 2)$

```
vector<int> pi, sigma, comp;
for (int i = 0; i < n; ++i)
    comp[i] = pi[sigma[i]];
```

Suppose π, σ are permutations. Get a new permutation, denoted $\pi \circ \sigma$, by first permuting according to σ and then according to π.

Example:

- $\pi = (0\ 1\ 4) \cdot (2\ 3)$
- $\sigma = (0) \cdot (1\ 2\ 4\ 3)$
- $\pi \circ \sigma = (0\ 1\ 3\ 4\ 2)$

```
vector<int> pi, sigma, comp;
for (int i = 0; i < n; ++i)
    comp[i] = pi[sigma[i]];
```

The $\circ$ operation on permutations is associative, so we can also use fast exponentiation!

i.e. computing π^k takes $O(k \log n)$ time where n is the number of items being permuted.

Finally, every permutation has an inverse. Just reverse the cycles!

Example

- $\pi = (0\ 1\ 4) \cdot (2\ 3)$
- $\pi^{-1} = (4\ 1\ 0) \cdot (3\ 2)$
- $\pi \circ \pi^{-1} = (0) \cdot (1) \cdot (2) \cdot (3) \cdot (4)$

```
vector<int> pi, inv;
for (int i = 0; i < n; ++i)
    inv[pi[i]] = i;
```

Counting Inversions

Problem: Given a permutation π, how many pairs of indices $i < j$ have $\pi[i] > \pi[j]$?

Trivial in $O(n^2)$ time.

Use **merge sort**: but when merging also count the number of numbers an item jumps over when being placed in the merged array.

i.e. if we have to merge sorted arrays `a` and `b`, if `a[0]` $>$ `b[0]` then add the length of `a` to the count of inversions.

Running Time: $O(n \log n)$

```
/* Count the number of inverted pairs i,j
   with lo <= i < j < hi. Here aux is an auxiliary array. */
long long count_inv(int* arr, int lo, int hi, int* aux) {
  if (lo+1 >= hi) return 0; // at most one element

  // split and recurse
  int mid = (lo+hi)/2;
  long long cnt = count_inv(a, lo, mid, aux);
  cnt += count_inv(a, mid, hi, aux);

  int il = lo, ir = mid, j = lo;
  while (il < mid || ir < hi)
    if (ir == hi || (il < mid && arr[il] <= arr[ir]))
      aux[j++] = arr[il++];
    else {
      // a[ir] hops the rest of the first half of a
      cnt += mid-il;
      aux[j++] = arr[ir++];
    }
  for (int k = lo; k < hi; ++k) arr[k] = aux[k];
  return cnt;
}
```

One quick slide on good-old counting!

\# of permutations of n items $= n! = n \cdot (n-1) \cdot \ldots \cdot 1$.

One quick slide on good-old counting!

\# of permutations of n items $= n! = n \cdot (n-1) \cdot \ldots \cdot 1$.

\# of size-k subsets of a size-n set $= \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$.

One quick slide on good-old counting!

\# of permutations of n items $= n! = n \cdot (n-1) \cdot \ldots \cdot 1$.

\# of size-k subsets of a size-n set $= \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$.

Also,

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{otherwise} \end{cases}$$

One quick slide on good-old counting!

\# of permutations of n items $= n! = n \cdot (n-1) \cdot \ldots \cdot 1$.

\# of size-k subsets of a size-n set $= \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$.

Also,

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \text{ or } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{otherwise} \end{cases}$$

Some problems need you to calculate these binomial coefficients.
Usually best to construct them using a table + dynamic programming.

Reminders/Scattered Facts:

- **Binomial Theorem**: $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$

Reminders/Scattered Facts:

- **Binomial Theorem**: $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$
- **Freshman Exponentiation**: $(a+b)^p \equiv a^p + b^p \mod p$ for prime p

Reminders/Scattered Facts:

- **Binomial Theorem**: $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$
- **Freshman Exponentiation**: $(a+b)^p \equiv a^p + b^p \mod p$ for prime p
- **Fibonacci gcd**: $\gcd(\mathbf{fib}(a), \mathbf{fib}(b)) = \mathbf{fib}(\gcd(a,b))$
  pause
- `next_permutation`: rearranges a vector/array to the lexicographically-next permutation in linear time.

UVa - 735 Dart-a-Mania (link)

UVa - 735 Dart-a-Mania (link)

No need to be clever. The input is so small that you can just try enumerating them all with three nested for loops.

UVa - 735 Dart-a-Mania (link)

No need to be clever. The input is so small that you can just try enumerating them all with three nested for loops.

Tip: Populate an array with all possible score values so you can iterate over it easily.

UVa - 735 Dart-a-Mania (link)

No need to be clever. The input is so small that you can just try enumerating them all with three nested for loops.

Tip: Populate an array with all possible score values so you can iterate over it easily.

Just remember the double and triple score regions, as well as the bullseye and zero regions.

UVa - 735 Dart-a-Mania (link)

No need to be clever. The input is so small that you can just try enumerating them all with three nested for loops.

Tip: Populate an array with all possible score values so you can iterate over it easily.

Just remember the double and triple score regions, as well as the bullseye and zero regions.

Tip: For each permutation, have it also contribute to the combination only if the permutation is in sorted order.

Open Kattis (namethatpermutation)

Open Kattis (namethatpermutation)

For a given number $1 \leq k \leq n$, how many permutations have k as the first element?

Open Kattis (namethatpermutation)

For a given number $1 \leq k \leq n$, how many permutations have k as the first element?

How many having a number $\leq k$ as the first element?

Open Kattis (namethatpermutation)

For a given number $1 \leq k \leq n$, how many permutations have k as the first element?

How many having a number $\leq k$ as the first element?

Use this to find the right k for the first element and repeat (some small details).

Open Kattis (namethatpermutation)

For a given number $1 \leq k \leq n$, how many permutations have k as the first element?

How many having a number $\leq k$ as the first element?

Use this to find the right k for the first element and repeat (some small details).

Big Issue!: Can you spot it?

Open Kattis (namethatpermutation)

For a given number $1 \leq k \leq n$, how many permutations have k as the first element?

How many having a number $\leq k$ as the first element?

Use this to find the right k for the first element and repeat (some small details).

Big Issue!: Can you spot it?

$50! - 1 \approx 3 \cdot 10^{64}$

Use Python (or Java, sigh...)

UVa - 11582 Colossal Fibonacci (link)

UVa - 11582 Colossal Fibonacci (link)

Numbers are too big even for the fast exponentiation trick, if a, b are as big as possible:

$$n = a^b \text{ so } \log_2 n = b \cdot \log_2 a \geq 2^{64}.$$

UVa - 11582 Colossal Fibonacci (link)

Numbers are too big even for the fast exponentiation trick, if a, b are as big as possible:

$$n = a^b \text{ so } \log_2 n = b \cdot \log_2 a \geq 2^{64}.$$

Look at the small modulus. Any repeating pattern?

UVa - 11582 Colossal Fibonacci (link)

Numbers are too big even for the fast exponentiation trick, if a, b are as big as possible:

$$n = a^b \text{ so } \log_2 n = b \cdot \log_2 a \geq 2^{64}.$$

Look at the small modulus. Any repeating pattern?

The sequence of pairs $(\textbf{fib}(i), \textbf{fib}(i+1))$ mod n must eventually repeat after $O(n^2)$ steps. Find the "cycle length" c (can show there is no "pre-period" to the cycle).

UVa - 11582 Colossal Fibonacci (link)

Numbers are too big even for the fast exponentiation trick, if a, b are as big as possible:

$$n = a^b \text{ so } \log_2 n = b \cdot \log_2 a \geq 2^{64}.$$

Look at the small modulus. Any repeating pattern?

The sequence of pairs $(\textbf{fib}(i), \textbf{fib}(i+1))$ mod n must eventually repeat after $O(n^2)$ steps. Find the "cycle length" c (can show there is no "pre-period" to the cycle).

Compute $x := a^b$ mod c and then find $\textbf{fib}(x)$ mod n.

Open Kattis (difficult)

Open Kattis (difficult)

Let $C_{1,2}$ be the number of pairs that agree between the 1st and 2nd list. Similarly consider $C_{2,3}$ and $C_{1,3}$.

Open Kattis (difficult)

Let $C_{1,2}$ be the number of pairs that agree between the 1st and 2nd list. Similarly consider $C_{2,3}$ and $C_{1,3}$.

Can we compute these?

Open Kattis (difficult)

Let $C_{1,2}$ be the number of pairs that agree between the 1st and 2nd list. Similarly consider $C_{2,3}$ and $C_{1,3}$.

Can we compute these? **Yes**: for any two (say lists 1 and 2) relabel the numbers so one is $1, 2, 3, \ldots, n$. Count # inversions in the other.

It is enough to know $C_{1,2}$, $C_{2,3}$ and $C_{1,3}$ to figure out the final answer.

Open Kattis (difficult)

Let $C_{1,2}$ be the number of pairs that agree between the 1st and 2nd list. Similarly consider $C_{2,3}$ and $C_{1,3}$.

Can we compute these? **Yes**: for any two (say lists 1 and 2) relabel the numbers so one is $1, 2, 3, \ldots, n$. Count # inversions in the other.

It is enough to know $C_{1,2}, C_{2,3}$ and $C_{1,3}$ to figure out the final answer. Let A be the answer. Each pair that is consistent in all 3 contributes once to each of $C_{1,2}, C_{2,3}$ and $C_{1,3}$.

Open Kattis (difficult)

Let $C_{1,2}$ be the number of pairs that agree between the 1st and 2nd list. Similarly consider $C_{2,3}$ and $C_{1,3}$.

Can we compute these? **Yes**: for any two (say lists 1 and 2) relabel the numbers so one is $1, 2, 3, \ldots, n$. Count # inversions in the other.

It is enough to know $C_{1,2}, C_{2,3}$ and $C_{1,3}$ to figure out the final answer. Let A be the answer. Each pair that is consistent in all 3 contributes once to each of $C_{1,2}, C_{2,3}$ and $C_{1,3}$.

Each pair that is inconsistent will contribute once to exactly one.

Open Kattis (difficult)

Let $C_{1,2}$ be the number of pairs that agree between the 1st and 2nd list. Similarly consider $C_{2,3}$ and $C_{1,3}$.

Can we compute these? **Yes**: for any two (say lists 1 and 2) relabel the numbers so one is $1, 2, 3, \ldots, n$. Count # inversions in the other.

It is enough to know $C_{1,2}, C_{2,3}$ and $C_{1,3}$ to figure out the final answer. Let A be the answer. Each pair that is consistent in all 3 contributes once to each of $C_{1,2}, C_{2,3}$ and $C_{1,3}$.

Each pair that is inconsistent will contribute once to exactly one.

So $3A + (\binom{n}{2} - A) = C_{1,2} + C_{2,3} + C_{1,3}$. Solve for A.

Total Time: $O(n \log n)$.