FFT

Polynomial :

$$A(x) \;=\; \sum_{j=0}^{n-1} a_j x^j$$

Coefficient form:

$$(a_0, a_1, \ldots, a_{n-1})$$

Point value form :

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

such that all of the x_k are distinct and

$$y_k = A(x_k)$$

Multiplication:

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$B(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$C(x) \;=\; A(x)B(x)$$

$$C(x) = \sum_{j=0}^{2n-2} c_j x^j ,$$

where

$$c_j = \sum_{k=0}^{j} a_k b_{j-k} .$$

$$C(x_k) = A(x_k)B(x_k)$$

Multiplying polynomials in point value form is O(n).

If you know the point value form, then coefficient form is unique.

Main idea behind fast polynomial multiplication:

$$\begin{tikzcd}
\boxed{\begin{matrix} a_0, a_1, \ldots, a_{n-1} \\ b_0, b_1, \ldots, b_{n-1} \end{matrix}} \arrow[r, "\text{Ordinary multiplication}", "\text{Time } \Theta(n^2)"'] \arrow[d, "\begin{matrix}\text{Evaluation} \\ \text{Time } \Theta(n \lg n)\end{matrix}"] & \boxed{c_0, c_1, \ldots, c_{2n-2}} & \left.\right\} \begin{matrix}\text{Coefficient} \\ \text{representations}\end{matrix} \\
\boxed{\begin{matrix} A(\omega_{2n}^0), B(\omega_{2n}^0) \\ A(\omega_{2n}^1), B(\omega_{2n}^1) \\ \vdots \\ A(\omega_{2n}^{2n-1}), B(\omega_{2n}^{2n-1}) \end{matrix}} \arrow[r, "\text{Pointwise multiplication}", "\text{Time } \Theta(n)"'] & \boxed{\begin{matrix} C(\omega_{2n}^0) \\ C(\omega_{2n}^1) \\ \vdots \\ C(\omega_{2n}^{2n-1}) \end{matrix}} \arrow[u, "\begin{matrix}\text{Interpolation} \\ \text{Time } \Theta(n \lg n)\end{matrix}"] & \left.\right\} \begin{matrix}\text{Point-value} \\ \text{representations}\end{matrix}
\end{tikzcd}$$

Four lemmas:

Lemma 1 — For any integer $n \geq 0$, $k \geq 0$ and $d \geq 0$, $w_{dn}^{dk} = w_n^k$

Lemma 2 — For any even integer $n > 0$, $w_n^{n/2} = w_2 = -1$

Lemma 3 — If $n > 0$ is even, then the squares of the n complex nth roots of unity are the (n/2) complex (n/2)th roots of unity, formally $(w_n^k)^2 = (w_n^{k+n/2})^2 = w_{n/2}^k$

Lemma 4 — For any integer $n \geq 0$, $k \geq 0$, $w_n^{k+n/2} = -w_n^k$
W_n^i = ith root of the total n (nth roots) of unity = $e^{2pi\ i/n}$

The n points that we are choosing for converting from coefficient form to point value form.
These n points are the n nth roots of unity.

$$\begin{aligned} y_k &= A(\omega_n^k) \\ &= \sum_{j=0}^{n-1} a_j \omega_n^{kj} \end{aligned}$$

Divide the coefficients into two parts -
1 part has only the odd coefficients and the other part has only even coefficients.

$$\begin{aligned} A^{[0]}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1} \\ A^{[1]}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1} \end{aligned}$$

$A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$

$A^{[0]}$ and $A^{[1]}$ at squares of the n roots.

So in all we are not evaluating at n nth roots of unity, we are instead evaluating at n/2 n/2 th roots of unity.

$X_i = W_n^i$

$A(x_k) = A^{[0]}(x_k^2) + x_k A^{[1]}(x_k^2)$, $k < n/2$
$A(x_{k+n/2}) = A^{[0]}(x_{k+n/2}^2) + x_{k+n/2} A^{[1]}(x_{k+n/2}^2)$
$= A^{[0]}(x_k^2) - x_k A^{[1]}(x_k^2)$
For each coefficient, it'll take O(1)
So total O(n) time for evaluating A using $A^{[0]}$ and $A^{[1]}$.

$T(n) = T(n/2) + O(n)$

$T(n) = n \log n$

At each step of the recursion, our n should be even.
N should be initially a power of 2.

(1,2,3) -> (1,2,3,0) // ensure that the total number of terms is a power of 2.

$$\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$\frac{1}{n} \begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^{-1} & w_n^{-2} & w_n^{-3} & \cdots & w_n^{-(n-1)} \\ w_n^0 & w_n^{-2} & w_n^{-4} & w_n^{-6} & \cdots & w_n^{-2(n-1)} \\ w_n^0 & w_n^{-3} & w_n^{-6} & w_n^{-9} & \cdots & w_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{-(n-1)} & w_n^{-2(n-1)} & w_n^{-3(n-1)} & \cdots & w_n^{-(n-1)(n-1)} \end{pmatrix}$$

$$a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}$$

$$\begin{aligned} y_k &= A(\omega_n^k) \\ &= \sum_{j=0}^{n-1} a_j \omega_n^{kj} \end{aligned}$$

Interpolation can be reduced to evaluation and also be done in O(n logn).

Q.) You are given the polynomial in this form

(x-a1)(x-a2)....(x-an)

You have to find the coefficients of this polynomial

It'll have n+1 coefficients.

```
Polynomial f( factors(1...n))
{
if(n==1) polynomial({-factor[1],1});
Mid = n/2;
Return f(factors(1...n/2)) * f(factors(n/2+1...n));
}
```

T(n) = T(n/2) + O(nlogn)
T(n) = n $\log^2 n$

P = $c2^k + 1$
NTT -> mod 98244353 = 119 * 2^{23} + 1

Primitive root g = 3
const int mod = 7340033;

```
const int root = 15311432;
const int root_1 = 469870224;
const int root_pw = 1 << 20;
```

Problem 1: CF - Placing Rooks:
https://codeforces.com/problemset/problem/1342/E

Either each row or each column should have exactly one rook.

Let say each row has exactly one rook.

If there are rooks in a column, a-1 pairs will attack each other.

Summation (a-1) = k;
Suppose there are m non empty columns

N - m = k => m = n-k

Select m columns, and distribute the rooks among them
A1, a2,..... Am

For a fixed set of (a1,a2,....am), what will be number of ways to distribute them?

Name the rooks like this -> its row number is its name
1111..222..33….mmm

For the fixed arrangement = n!/(a1!a2!....am!)

All possible (a1,a2,....am) such that a1+a2+a3...am = n
Ai >= 1

(x/1! + x^2/2! + x^3/3! ….. x^n/n!)^m
Coefficient of x^n in the above polynomial.

Binary exponentiation, with polynomial multiplication
O(n logn logm)

Problem 2 : CF - Thief in a Shop
https://codeforces.com/problemset/problem/632/E

Make a polynomial coefficient of x^i = 1 iff there exist an item with cost i

Let P(x) be that polynomial.
Find $P^k(x)$ and count number of non zero coefficients.

For detailed explanation of FFT & NTT

1. https://codeforces.com/blog/entry/43499
2. https://codeforces.com/blog/entry/48798

More practice problems on FFT

1. https://codeforces.com/problemset/problem/1342/E
2. https://codeforces.com/problemset/problem/300/D
3. https://codeforces.com/problemset/problem/632/E
4. https://www.codechef.com/problems/CHEFSSM
5. https://www.codechef.com/problems/SUMXOR2
6. https://www.codechef.com/problems/WEIRDMUL

SQRT Decomposition

Q. Given an array A of size N . You are given q queries of 2 types:
1. "Min L R " - Find the minimum in the range [L,R].
2. "Modify (i, val)" - Modify A[i] = val
(n <= 10^4, q<=10^4)

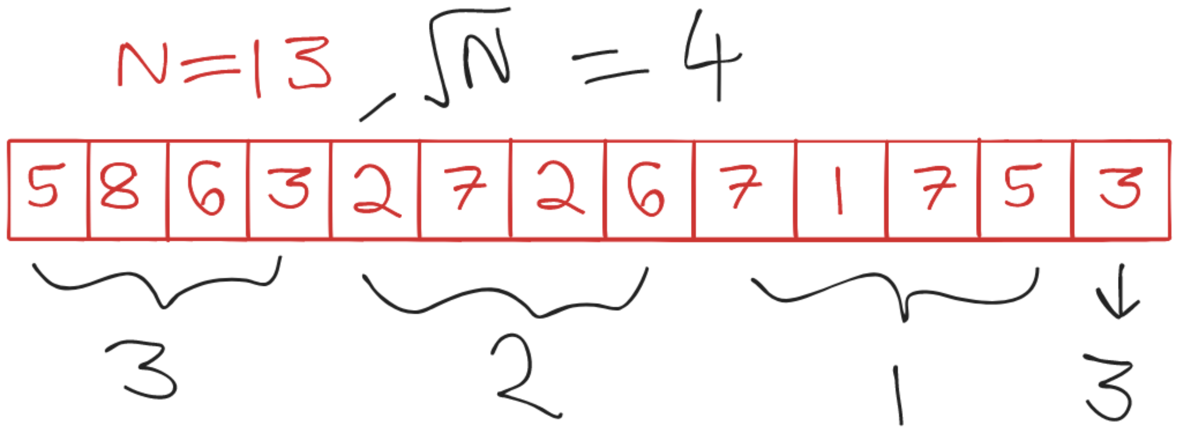
Approach:

Divide the array into blocks and calculate the minimum for each block.
If block size is B, no. of blocks = N/B

For performing a range query, Time complexity : O (N / B + B)

For performing a point update, Time complexity: O (B)

Optimal block size is sqrt(N) [You can take ceil of this value]



Pre-computation: O(N)

C++ Implementation is shown below:

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

int n,len;
vector<int> b,a;

const int inf=1e18;
void pre()
{
     len = (int) sqrt (n + .0) + 1; // size of the block and
the number of blocks
     b.assign(len,inf);

     for (int i=0; i<n; ++i)
          b[i / len] = min(b[i/len],a[i]);

}

int query(int l, int r)
```

```
{
    int ans = inf;
    int i=l;
    while(i<=r)
    {
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to [l,
r]
            ans = min(ans, b[i / len]);
            i += len;
        }
        else {
            ans = min(ans,a[i]);
            ++i;
        }
     }
     return ans;
}

int32_t main()
{
cin>>n;

a.resize(n);
for(int i=0; i<n; i++)
{
     cin>>a[i];
}

pre(); // pre-processing

int q;
cin>>q;

while(q--)
{
    int l, r;
     cin>>l>>r;

     // print the minimum for each query
     cout<<query(l,r)<<'\n';

     // For updating an element,
     // just recalculate b[i] value for its block
     // and also update that element in the array a
}

return 0;
}
```

Mo's Algorithm

- For static arrays
- When we can change the order of queries (Offline mode of answering queries)
Sort all the queries (L, R) in the following way:

(i) Sort in increasing order block number of L
(ii) If block number of L is the same for 2 queries, then sort in increasing order of R.

It maintains an active range for the query and when the range changes, it just adds or removes elements from the active range to calculate the new answer.

```
void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data
structure

int block_size;
```

```
struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l,
cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

For left border, O(Q * sqrt(N)) [For each query, left border changes in worst case by sqrt(N)]
For right border, O (N * sqrt(N)) [For each block, N times in worst case]

Time Complexity: O ((N + Q)*sqrt(N) * F)
where F is the complexity for add/remove

Note: Block size should be taken as constant (roughly sqrt(max value of N)), because division by constant is optimised by the compilers.

Q. You have an array A of N numbers from 0 to 1000000, You have Q queries [L,R]. For each query, you must print F(L,R) = $\sum_{i=0}^{1000000}$ count(i)^2 * i
where count(i) = no. of times i appears in A[L...R]
(1<= N , Q <=10^5)
Link: https://codeforces.com/problemset/problem/86/D

```
const int MAX = 1e6+1;
int freq[MAX];
int ans=0;
void add(int idx)
{
      int num = a[idx];
      ans -= freq[num]*freq[num]*num;
      freq[num]++;
      ans += freq[num]*freq[num]*num;
}

void remove(int idx)
{
     int num = a[idx];
     ans -= freq[num]*freq[num]*num;
     freq[num]--;
     ans += freq[num]*freq[num]*num;
}

int get_answer()
{
     return ans;
}
```

Q. You have an array A of N numbers from 0 to 10^9, You have Q queries [L,R]. For each query, you must print the element with the highest frequency in A[L...R] ($1 <= N , Q <= 10^5$)

Approach:
Maintain frequency of all elements using a map. And also take a set of pair (frequency, element) containing the first element with highest frequency.
Now, creating add(), remove() and get_answer() is easy
Apply Mo's algorithm
Time complexity: O((N+Q) sqrt(N) log N)

Q. Find inversions for each query in a static array.

Given a static array and queries of the form $[l, r]$, count the number of pairs (i, j) such that $a[i] > a[j]$.

Solution Approach:
Maintain Fenwick Tree for the frequency of elements present in active range.
Apply Mo's algorithm.
Time complexity: O((N+Q) sqrt(N) log N)

Another Approach (Optional): (Sweepline Mo's Algorithm)
Link to blog: https://codeforces.com/blog/entry/83248
Time complexity: O((N+Q) sqrt(N))

Practice Problems on Mo's Algorithm:
1. https://codeforces.com/problemset/problem/617/E
2. https://www.spoj.com/problems/DQUERY/

Another Application: (Combining Algorithms)
If, for a problem, we can solve it in Algo 1: O(k) as well as in Algo 2: O(n/k), then use a threshold (sqrt(N)), so that if k < threshold, use Algo 1, otherwise use Algo 2.
Reference: (See 27.1 from CPH book)
https://usaco.guide/CPH.pdf#page=262

2-SAT

- It is used to check satisfiability of boolean expression (check if it can be true)

$\vee$ – OR

$\wedge$ – AND

$\neg$ – NOT

We are given a boolean expression of this form: (2-SAT)

$$(a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge \cdots \wedge (a_m \vee b_m),$$

where a1 , b1, a2, b2,... am, bm are boolean variables or negation of boolean variable.
This is also called CNF (Conjunctive Normal Form).

If this expression is satisfiable, then we need to find the values of all the boolean variables that satisfy the expression.

Consider this expression:

$$L_1 = (x_2 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_4)$$

Let x_1 = false
x_2 = false
x_3 = true
x_4 = true
When we put these values in L_1, it is true (It means L_1 is satisfiable)

Each pair is of the form (a_i V b_i) in 2-SAT

It can be rewritten as :

$$(a_i \vee b_i)$$

$$\neg a_i \Rightarrow b_i$$

$$\neg b_i \Rightarrow a_i$$

Create a graph with 2*n nodes (One for each x_i and ¬ x_i)
We create edge from ¬ a_i to b_i and from ¬ b_i to a_i in a graph.

$$L_1 = (x_2 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_4)$$

$$\begin{tikzcd}
\neg x_3 \arrow[dr] & x_2 \arrow[l] \arrow[r] & \neg x_1 \arrow[r] \arrow[dr] & x_4 \\
\neg x_4 \arrow[r] & x_1 \arrow[u] \arrow[r] & \neg x_2 \arrow[u] & x_3 \arrow[l]
\end{tikzcd}$$

This is a directed graph.
Now, find SCCs.

We assign an id to each SCC.
(comp[v] = id of the SCC to which vertex v belongs)

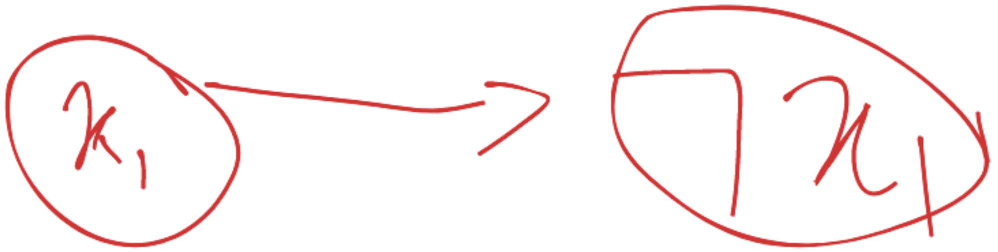
If comp[v] <= comp[u], it means there is a path from v to u

There are 3 cases:
1. comp[x_i] = comp[¬ x_i]

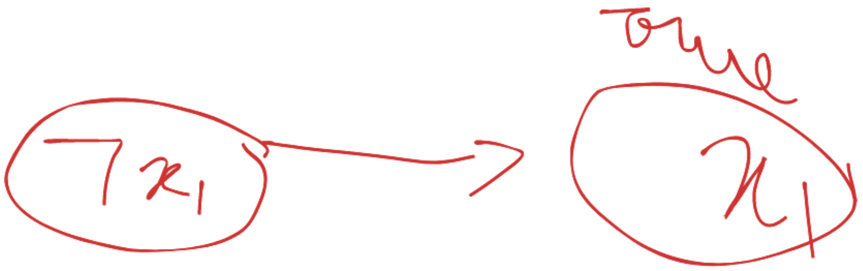
$$\begin{tikzcd}
x_1 \arrow[r] & \neg x_1 \arrow[l, bend left]
\end{tikzcd}$$

If x_i and ¬ x_i belong the same SCC, it is a contradiction. (No possible solution)
// No satisfiable

2. If comp [x_i] < comp [¬ x_i] ,
assign x_i as false



3. If comp [x_i] > comp [¬ x_i] ,
assign x_i as true



Note: Assignment to true in reverse topological order gives the correct solution.

Some boolean variable x_k can be represented as 2*k and 2*k+1 is not (x_k)

```
#include <bits/stdc++.h>

using namespace std;

int n;
vector<vector<int>> g, gt;
vector<bool> visited;
vector<int> order, comp;
vector<bool> assignment;

void dfs1(int v) {
    visited[v] = true;
    for (int u : g[v]) {
        if (!visited[u])
            dfs1(u);
    }
    order.push_back(v);
}

void dfs2(int v, int id) {
    comp[v] = id;
    for (int u : gt[v]) {
        if (comp[u] == -1)
            dfs2(u, id);
    }
}

/* This function will return true if there is a possible solution,
 * otherwise return false */
bool solve() {
     //2-SAT Algorithm
    visited.assign(n, false);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs1(i);
    }

     // Find SCCs (Give an id to each component)
    comp.assign(n, -1);
    reverse(order.begin(),order.end());
    int id=0;

    for (auto v: order) {
        if (comp[v] == -1)
            dfs2(v, id++);
    }
    assignment.assign(n / 2, false);
    for (int i = 0; i < n; i += 2) {
        if (comp[i] == comp[i + 1])
            return false; // No solution possible
        assignment[i / 2] = comp[i] > comp[i + 1];
    }

    return true;
}

int main()
{
     // Take n/2 (no. of boolean variables) as input

     // Take input of graph g with n vertices
     // For any variable x_k:
     // (i) vertex 2k => x_k
     // (i) vertex 2k+1 => not(x_k)

     // Store reverse of graph in gt
     solve();

     return 0;
}
```

Q. https://codeforces.com/problemset/problem/776/D

Solution Approach:

Let x1 , x2 be the 2 switches of a room, then there are 2 cases for intial state of the room:

Case $\textcircled{1}$ (locked) :-

$$(x_1 \land \neg x_2) \lor (\neg x_1 \land x_1)$$

Case $\textcircled{2}$ (unlocked):

$$(x_1 \land x_2) \lor (\neg x_1 \land \neg x_2)$$

For each room, there should be either of these form of expressions, take OR of all such expressions to convert it into a 2-SAT problem.

Practice Problems on 2-SAT:

1. https://cses.fi/problemset/task/1684
2. https://codeforces.com/problemset/problem/1215/F