

- 1 General
- 2 Algorithms
- 3 Data Structures
- 4 String
- 5 Math
- 6 Graph
- 7 2D Geometry
- 8 3D Geometry

1 General

Header

```
// use better compiler options
#pragma GCC optimize("Ofast","unroll-loops")
#pragma GCC target("avx2,fma")

// include everything
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <sys/resource.h>

// namespaces
using namespace std;
using namespace __gnu_cxx; // rope
using namespace __gnu_pbds; // tree/trie

// common defines
#define fastio ios_base::sync_with_stdio(0);
    cin.tie(0);
#define nostacklim rlimit RZ;getrlimit(3,&RZ);
    RZ.rlim_cur=-1;setrlimit(3,&RZ);
#define DEBUG(v) cout<<"DEBUG: "<<#v<<" = "<<v
    <<'\n';
#define ll long long
#define ull unsigned ll
#define i128 __int128
#define u128 unsigned i128
#define ld long double

// global variables
mt19937 rng((uint32_t)chrono::steady_clock::
    now().time_since_epoch().count());
```

Fast IO

```
void readn(unsigned int& n) {
    char c; n = 0;
    while ((c = getchar_unlocked()) != ' ' && c
        != '\n')
        n = n * 10 + c - '0';
}

void readn(int& n) {
    char c; n = 0; int s = 1;
    if ((c = getchar_unlocked()) == '-') s = -1;
```

```
    else n = c - '0';
    while ((c = getchar_unlocked()) != ' ' && c
        != '\n')
        n = n * 10 + c - '0';
    n *= s;
}

void readn(long double& n) {
    char c; n = 0;
    long double m = 0, o = 1; bool d = false; int
        s = 1;
    if ((c = getchar_unlocked()) == '-') s = -1;
    else if (c == '.') d = true;
    else n = c - '0';
    while ((c = getchar_unlocked()) != ' ' && c
        != '\n') {
        if (c == '.') d = true;
        else if (d) { m = m * 10 + c - '0'; o *=
            0.1; }
        else n = n * 10 + c - '0';
    }
    n = s * (n + m * o);
}

void readn(double& n) {
    long double m; readn(m); n = m;
}

void readn(float& n) {
    long double m; readn(m); n = m;
}

void readn(string& s) {
    char c; s = "";
    while((c = getchar_unlocked()) != ' ' && c !=
        '\n')
        s += c;
}

bool readline(string& s) {
    char c; s = "";
    while(c = getchar_unlocked()) {
        if (c == '\n') return true;
        if (c == EOF) return false;
        s += c;
    }
    return false;
}

void printn(unsigned int n) {
    if (n / 10) printn(n / 10);
    putchar_unlocked(n % 10 + '0');
}

void printn(int n) {
    if (n < 0) { putchar_unlocked('-'); n *= -1;
        }
    printn((unsigned int)n);
}
```

2 Algorithms

Min/Max Subarray

```
// max subarray - compare = a < b, reset = a <
    0
// min subarray - compare = a > b, reset = a >
    0
// returns {sum, {start, end}}
pair<int, pair<int, int>> ContiguousSubarray(
    int* a, int size, bool(*compare)(int, int),
    bool(*reset)(int), int defbest = 0) {
    int best = defbest, cur = 0, start = 0, end =
        0, s = 0;
    for (int i = 0; i < size; i++) {
        cur += a[i];
        if ((*compare)(best, cur)) { best = cur;
            start = s; end = i; }
        if ((*reset)(cur)) { cur = 0; s = i + 1; }
    }
    return {best, {start, end}};
}
```

Quickselect

```
#define QSNE -999999

int partition(int arr[], int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++)
        if (arr[j] <= x)
            swap(arr[i++], arr[j]);
    swap(arr[i], arr[r]);
    return i;

// find k'th smallest element in unsorted
    array, only if all distinct
int quickselect(int arr[], int l, int r, int k
    )
{
    if (!(k > 0 && k <= r - l + 1)) return QSNE;
    swap(arr[l + rng() % (r-l+1)], arr[r]);
    int pos = partition(arr, l, r);
    if (pos-l == k-1) return arr[pos];
    if (pos-l > k-1) return quickselect(arr, l,
        pos-1, k);
    return quickselect(arr, pos+1, r, k-pos+1-1);
}
```

// TODO: compare against std::nth_element()

Saddleback Search

```
// search for v in 2d array arr[x][y], sorted
    on both axis
pair<int, int> saddleback_search(int** arr,
    int x, int y, int v) {
```

```
    int i = x-1, j = 0;
    while (i >= 0 && j < y) {
        if (arr[i][j] == v) return {i, j};
        (arr[i][j] > v)? i--: j++;
    }
    return {-1, -1};
}
```

Ternary Search

```
// < for max, > for min, or any other unimodal
    func
#define TERNCOMP(a,b) (a)<(b)
int ternary_search(int a, int b, int (*f)(int)
    ) {
    while (b-a > 4) {
        int m = (a+b)/2;
        if (TERNCOMP((*f)(m), (*f)(m+1))) a = m;
        else b = m+1;
    }
    for (int i = a+1; i <= b; i++)
        if (TERNCOMP((*f)(a), (*f)(i)))
            a = i;
    return a;
}

#define TERNPREC 0.000001
double ternary_search(double a, double b,
    double (*f)(double)) {
    while (b-a > TERNPREC * 4) {
        double m = (a+b)/2;
        if (TERNCOMP((*f)(m), (*f)(m + TERNPREC))) a
            = m;
        else b = m + TERNPREC;
    }
    for (double i = a + TERNPREC; i <= b; i +=
        TERNPREC)
        if (TERNCOMP((*f)(a), (*f)(i)))
            a = i;
    return a;
}
```

3 Data Structures

Fenwick Tree

```
// Fenwick tree, array of cumulative sums - O(
    log n) updates, O(log n) gets
struct Fenwick {
    int n; ll* tree;

    void update(int i, int val) {
        ++i;
        while (i <= n) {
            tree[i] += val;
            i += i & (-i);
        }
    }
}
```

```
Fenwick(int size) {
    n = size;
    tree = new ll[n+1];
    for (int i = 1; i <= n; i++)
        tree[i] = 0;
}

Fenwick(int* arr, int size) : Fenwick(size) {
    for (int i = 0; i < n; i++)
        update(i, arr[i]);
}

~Fenwick() { delete[] tree; }

ll operator[](int i) {
    if (i < 0 || i > n) return 0;
    ll sum = 0;
    ++i;
    while (i>0) {
        sum += tree[i];
        i -= i & (-i);
    }
    return sum;
}

ll getRange(int a, int b) { return operator
↪ [](b) - operator[](a-1); }
};
```

Hashtable

```
// similar to unordered_map, but faster
struct chash {
    const uint64_t C = (1ll)(2e18 * M_PI) + 71;
    ll operator()(ll x) const { return
↪ __builtin_bswap64(x*C); }
};

int main() {
    gp_hash_table<ll,int,chasht> hashtable
↪ ({}},{},{},{},{1<<16});
    for (int i = 0; i < 100; i++)
        hashtable[i] = 200+i;
    if (hashtable.find(10) != hashtable.end())
        cout << hashtable[10];
}
```

Ordered Set

```
typedef tree<int,null_type,less<int>,
↪ rb_tree_tag,
↪ tree_order_statistics_node_update>
↪ ordered_set;

int main()
{
    ordered_set o_set;
    o_set.insert(5); o_set.insert(1); o_set.
↪ insert(3);

    // get second smallest element
    cout << *(o_set.find_by_order(1)) << '\n';
```

```
// number of elements less than k=4
    cout << o_set.order_of_key(4) << '\n';
}
```

Rope

```
// O(log n) insert, delete, concatenate
int main() {
    // generate rope
    rope<int> v;
    for (int i = 0; i < 100; i++)
        v.push_back(i);

    // move range to front
    rope<int> copy = v.substr(10, 10);
    v.erase(10, 10);
    v.insert(copy.mutable_begin(), copy);

    // print elements of rope
    for (auto it : v)
        cout << it << " ";
}
```

Segment Tree

```
//max(a,b), min(a,b), a+b, a*b, gcd(a,b), a^b
struct SegmentTree {
    typedef int T;
    static constexpr T UNIT = INT_MIN;
    T f(T a, T b) {
        if (a == UNIT) return b;
        if (b == UNIT) return a;
        return max(a,b);
    }
    int n; vector<T> s;
    SegmentTree(int n, T def=UNIT) : s(2*n, def),
↪ n(n) {}
    SegmentTree(vector<T> arr) : SegmentTree(arr.
↪ size()) {
        for (int i=0;i<arr.size();i++) update(i,arr[
↪ i]);
    }
    void update(int pos, T val) {
        for (s[pos += n] = val; pos /= 2;)
            s[pos] = f(s[pos * 2], s[pos*2+1]);
    }
    T query(int b, int e) { // query [b, e)
        T ra = UNIT, rb = UNIT;
        for (b+=n, e+=n; b<e; b/=2, e/=2) {
            if (b % 2) ra = f(ra, s[b++]);
            if (e % 2) rb = f(s[--e], rb);
        }
        return f(ra, rb);
    }
    T get(int p) { return query(p, p+1); }
};
```

Trie

```
typedef trie<string, null_type,
↪ trie_string_access_traits<>,
    pat_trie_tag, trie_prefix_search_node_update
↪ > trie_type;

int main() {
    // generate trie
    trie_type trie;
    for (int i = 0; i < 20; i++)
        trie.insert(to_string(i)); // true if new,
↪ false if old

    // print things with prefix "1"
    auto range = trie.prefix_range("1");
    for (auto it = range.first; it != range.
↪ second; it++)
        cout << *it << " ";
}
```

4 String

Aho Corasick

```
// range of alphabet for automata to consider
// MAXC = 26, OFFC = 'a' if only lowercase
↪ letters
const int MAXC = 256;
const int OFFC = 0;

struct aho_corasick {
    struct state
    {
        set<pair<int, int>> out;
        int fail; vector<int> go;
        state() : fail(-1), go(MAXC, -1) {}
    };
    vector<state> s;
    int id = 0;

    aho_corasick(string* arr, int size) : s(1) {
        for (int i = 0; i < size; i++) {
            int cur = 0;
            for (int c : arr[i]) {
                if (s[cur].go[c-OFFC] == -1) {
                    s[cur].go[c-OFFC] = s.size();
                    s.push_back(state());
                }
                cur = s[cur].go[c-OFFC];
            }
            s[cur].out.insert({arr[i].size(), id++});
        }

        for (int c = 0; c < MAXC; c++)
            if (s[0].go[c] == -1)
                s[0].go[c] = 0;
        queue<int> sq;
```

```
for (int c = 0; c < MAXC; c++) {
    if (s[0].go[c] != 0) {
        s[s[0].go[c]].fail = 0;
        sq.push(s[0].go[c]);
    }
}
while (sq.size()) {
    int e = sq.front(); sq.pop();
    for (int c = 0; c < MAXC; c++) {
        if (s[e].go[c] != -1) {
            int failure = s[e].fail;
            while (s[failure].go[c] == -1)
                failure = s[failure].fail;
            failure = s[failure].go[c];
            s[s[e].go[c]].fail = failure;
            for (auto length : s[failure].out)
                s[s[e].go[c]].out.insert(length);
            sq.push(s[e].go[c]);
        }
    }
}

// list of {start pos, pattern id}
vector<pair<int, int>> search(string text)
{
    vector<pair<int, int>> toret;
    int cur = 0;

    for (int i = 0; i < text.size(); i++) {
        while (s[cur].go[text[i]-OFFC] == -1)
            cur = s[cur].fail;
        cur = s[cur].go[text[i]-OFFC];

        if (s[cur].out.size())
            for (auto end : s[cur].out)
                toret.push_back({i - end.first + 1, end.
↪ second});
    }
    return toret;
}
};
```

Boyer Moore

```
struct defint { int i = -1; };
vector<int> boyermoore(string txt, string pat)
↪ {
    vector<int> toret; unordered_map<char, defint
↪ > badchar;
    int m = pat.size(), n = txt.size();
    for (int i = 0; i < m; i++) badchar[pat[i]].i
↪ = i;
    int s = 0;
    while (s <= n - m) {
        int j = m - 1;
        while (j >= 0 && pat[j] == txt[s + j]) j--;
        if (j < 0) {
```

```

    toret.push_back(s);
    s += (s + m < n) ? m - badchar[txt[s + m]].
    ↪ i : 1;
} else
    s += max(1, j - badchar[txt[s + j]].i);
}
return toret;
}

```

English Conversion

```

const string ones[] = {"", "one", "two", " "
    ↪ three", "four", "five", "six", "seven", " "
    ↪ eight", "nine"};
const string teens[] = {"ten", "eleven", " "
    ↪ twelve", "thirteen", "fourteen", "fifteen", " "
    ↪ "sixteen", "seventeen", "eighteen", " "
    ↪ nineteen"};
const string tens[] = {"twenty", "thirty", " "
    ↪ forty", "fifty", "sixty", "seventy", " "
    ↪ eighty", "ninety"};
const string mags[] = {"thousand", "million", " "
    ↪ "billion", "trillion", "quadrillion", " "
    ↪ quintillion", "sextillion", "septillion"};
string convert(int num, int carry) {
    if (num < 0) return "negative " + convert(-
    ↪ num, 0);
    if (num < 10) return ones[num];
    if (num < 20) return teens[num % 10];
    if (num < 100) return tens[(num / 10) - 2] +
    ↪ (num%10==0?"":" " ") + ones[num % 10];
    if (num < 1000) return ones[num / 100] + (num
    ↪ /100==0?"":" " ") + "hundred" + (num%100==0?
    ↪ "":" " ") + convert(num % 100, 0);
    return convert(num / 1000, carry + 1) + " " +
    ↪ mags[carry] + " " + convert(num % 1000,
    ↪ 0);
}

string convert(int num) {
    return (num == 0) ? "zero" : convert(num, 0);
}

```

Knuth Morris Pratt

```

vector<int> kmp(string txt, string pat) {
    vector<int> toret;
    int m = txt.length(), n = pat.length();

    int next[n + 1];
    for (int i = 0; i < n + 1; i++)
        next[i] = 0;

    for (int i = 1; i < n; i++) {
        int j = next[i + 1];
        while (j > 0 && pat[j] != pat[i])
            j = next[j];
        if (j > 0 || pat[j] == pat[i])
            next[i + 1] = j + 1;
    }
}

```

```

for (int i = 0, j = 0; i < m; i++) {
    if (txt[i] == pat[j]) {
        if (++j == n)
            toret.push_back(i - j + 1);
    } else if (j > 0) {
        j = next[j];
        i--;
    }
}
return toret;
}

```

Longest Common Prefix

```

string lcp(string* arr, int n) {
    if (n == 0) return "";
    sort(arr, arr + n);
    string r = ""; int v = 0;
    while (v < arr[0].length() && arr[0][v] ==
    ↪ arr[n-1][v])
        r += arr[0][v++];
    return r;
}

```

Longest Common Subsequence

```

string lcs(string a, string b) {
    int m = a.length(), n = b.length();

    int L[m+1][n+1];
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) L[i][j] = 0;
            else if (a[i-1] == b[j-1]) L[i][j] = L[i
            ↪ -1][j-1]+1;
            else L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }
    // return L[m][n]; // length of lcs

    string out = "";
    int i = m - 1, j = n - 1;
    while (i >= 0 && j >= 0) {
        if (a[i] == b[j]) {
            out = a[i--] + out;
            j--;
        }
        else if (L[i][j+1] > L[i+1][j]) i--;
        else j--;
    }

    return out;
}

```

Longest Common Substring

```

// l is array of palindrome length at that
↪ index

```

```

int manacher(string s, int* l) {
    int n = s.length() * 2;
    for (int i = 0, j = 0, k; i < n; i += k, j =
    ↪ max(j-k, 0)) {
        while (i >= j && i + j + 1 < n && s[(i-j)/2]
    ↪ == s[(i+j+1)/2]) j++;
        l[i] = j;
        for (k = 1; i >= k && j >= k && l[i-k] != j-
    ↪ k; k++)
            l[i+k] = min(l[i-k], j-k);
    }
    return *max_element(l, l + n);
}

```

Subsequence Count

```

// 0(m*n) - "banana", "ban" >> 3 (ban, ba..n,
    ↪ b..an)
ull subsequences(string body, string subs) {
    int m = subs.length(), n = body.length();
    if (m > n) return 0;
    ull** arr = new ull*[m+1];
    for (int i = 0; i <= m; i++) arr[i] = new ull
    ↪ [n+1];
    for (int i = 1; i <= m; i++) arr[i][0] = 0;
    for (int i = 0; i <= n; i++) arr[0][i] = 1;
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            arr[i][j] = arr[i][j-1] + ((body[j-1] ==
            ↪ subs[i-1])? arr[i-1][j-1] : 0);
    return arr[m][n];
}

```

5 Math

Catalan Numbers

```

ull* catalan = new ull[1000000];

void genCatalan(int n, int mod) {
    catalan[0] = catalan[1] = 1;
    for (int i = 2; i <= n; i++) {
        catalan[i] = 0;
        for (int j = i - 1; j >= 0; j--) {
            catalan[i] += (catalan[j] * catalan[i-j-1])
            ↪ % mod;
            if (catalan[i] >= mod)
                catalan[i] -= mod;
        }
    }
}

```

// TODO: consider binomial coefficient method

Combinatorics (nCr, nPr)

```

// can optimize by precomputing factorials,
↪ and fact[n]/fact[n-r]
ull nPr(ull n, ull r) {

```

```

    ull v = 1;
    for (ull i = n-r+1; i <= n; i++)
        v *= i;
    return v;
}

ull nPr(ull n, ull r, ull m) {
    ull v = 1;
    for (ull i = n-r+1; i <= n; i++)
        v = (v * i) % m;
    return v;
}

```

```

ull nCr(ull n, ull r) {
    long double v = 1;
    for (ull i = 1; i <= r; i++)
        v = v * (n-r+i) / i;
    return (ull)(v + 0.001);
}

// requires modulo math
// can optimize by precomputing mfac and minv-
↪ mfac
ull nCr(ull n, ull r, ull m) {
    return mfac(n, m) * minv(mfac(k, m), m) % m *
    ↪ minv(mfac(n-k, m), m) % m;
}

```

Chinese Remainder Theorem

```

bool ecrt(ll* r, ll* m, int n, ll& re, ll& mo)
    ↪ {
    ll x, y, d; mo = m[0]; re = r[0];
    for (int i = 1; i < n; i++) {
        d = egcd(mo, m[i], x, y);
        if ((r[i] - re) % d != 0) return false;
        x = (r[i] - re) / d * x % (m[i] / d);
        re += x * mo;
        mo = mo / d * m[i];
        re %= mo;
    }
    re = (re + mo) % mo;
    return true;
}

```

Count Digit Occurences

```

/*count(n,d) counts the number of occurrences
    ↪ of a digit d in the range [0,n]*/
ll digit_count(ll n, ll d) {
    ll result = 0;
    while (n != 0) {
        result += ((n%10) == d ? 1 : 0);
        n /= 10;
    }
    return result;
}

```

```

ll count(ll n, ll d) {

```

```

    if (n < 10) return (d > 0 && n >= d);
    if ((n % 10) != 9) return digit_count(n, d)
    ↪ + count(n-1, d);
    return 10*count(n/10, d) + (n/10) + (d > 0)
    ↪ ;
}

```

Discrete Logarithm

```

unordered_map<int, int> dlogc;
int discretelog(int a, int b, int m) {
    dlogc.clear();
    ll n = sqrt(m)+1, an = 1;
    for (int i = 0; i < n; i++)
        an = (an * a) % m;

    ll c = an;
    for (int i = 1; i <= n; i++) {
        if (!dlogc.count(c)) dlogc[c] = i;
        c = (c * an) % m;
    }

    c = b;
    for (int i = 0; i <= n; i++) {
        if (dlogc.count(c)) return (dlogc[c] * n - i
        ↪ + m - 1) % (m-1);
        c = (c * a) % m;
    }
    return -1;
}

```

Euler Phi / Totient

```

int phi(int n) {
    int r = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) r -= r / i;
        while (n % i == 0) n /= i;
    }
    if (n > 1) r -= r / n;
    return r;
}

```

```

#define n 100000
ll phi[n+1];
void computeTotient() {
    for (int i=1; i<=n; i++) phi[i] = i;
    for (int p=2; p<=n; p++) {
        if (phi[p] == p) {
            phi[p] = p-1;
            for (int i = 2*p; i<=n; i += p) phi
            ↪ [i] = (phi[i]/p) * (p-1);
        }
    }
}

```

Factorials

```

// digits in factorial

```

```

#define kamenetsky(n) (floor((n * log10(n /
    ↪ M_E)) + (log10(2 * M_PI * n) / 2.0)) + 1)

// approximation of factorial
#define stirling(n) ((n == 1) ? 1 : sqrt(2 *
    ↪ M_PI * n) * pow(n / M_E, n))

// natural log of factorial
#define lfactorial(n) (lgamma(n+1))

```

Prime Factorization

```

// do not call directly
ll pollard_rho(ll n, ll s) {
    ll x, y;
    x = y = rand() % (n - 1) + 1;
    int head = 1, tail = 2;
    while (true) {
        x = mult(x, x, n);
        x = (x + s) % n;
        if (x == y) return n;
        ll d = __gcd(max(x - y, y - x), n);
        if (1 < d && d < n) return d;
        if (++head == tail) y = x, tail <= 1;
    }

    // call for prime factors
    void factorize(ll n, vector<ll> &divisor) {
        if (n == 1) return;
        if (isPrime(n)) divisor.push_back(n);
        else {
            ll d = n;
            while (d >= n) d = pollard_rho(n, rand() % (
            ↪ n - 1) + 1);
            factorize(n / d, divisor);
            factorize(d, divisor);
        }
    }
}

```

Farey Fractions

```

// generate 0 <= a/b <= 1 ordered, b <= n
// farey(4) = 0/1 1/4 1/3 1/2 2/3 3/4 1/1
// length is sum of phi(i) for i = 1 to n

```

```

vector<pair<int, int>> farey(int n) {
    int h = 0, k = 1, x = 1, y = 0, r;
    vector<pair<int, int>> v;
    do {
        v.push_back({h, k});
        r = (n-y)/k;
        y += r*k; x += r*h;
        swap(x,h); swap(y,k);
        x = -x; y = -y;
    } while (k > 1);
    v.push_back({1, 1});
    return v;
}

```

Fast Fourier Transform

```

#define cd complex<double>
const double PI = acos(-1);

```

```

void fft(vector<cd>& a, bool invert) {
    int n = a.size();

    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1) j ^= bit;
        j ^= bit;

        if (i < j) swap(a[i], a[j]);
    }
}

```

```

for (int len = 2; len <= n; len <= 1) {
    double ang = 2 * PI / len * (invert ? -1 :
    ↪ 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
        cd w(1);
        for (int j = 0; j < len / 2; j++) {
            cd u = a[i+j], v = a[i+j+len/2] * w;
            a[i+j] = u + v;
            a[i+j+len/2] = u - v;
            w *= wlen;
        }
    }
}

```

```

if (invert)
    for (auto& x : a)
        x /= n;
}

```

```

vector<int> fftmult(vector<int> const& a,
    ↪ vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin
    ↪ (), b.end());
    int n = 1 << (32 - __builtin_clz(a.size() + b
    ↪ .size() - 1));
    fa.resize(n); fb.resize(n);
}

```

```

fft(fa, false); fft(fb, false);
for (int i = 0; i < n; i++) fa[i] *= fb[i];
fft(fa, true);

```

```

vector<int> toret(n);
for (int i = 0; i < n; i++) toret[i] = round(
    ↪ fa[i].real());
return toret;
}

```

Greatest Common Denominator

```

ll egcd(ll a, ll b, ll& x, ll& y) {
    if (b == 0) { x = 1; y = 0; return a; }
    ll gcd = egcd(b, a % b, x, y);
}

```

```

x -= a / b * y;
swap(x, y);
return gcd;
}

```

Josephus Problem

```

// 0-indexed, arbitrary k
int josephus(int n, int k) {
    if (n == 1) return 0;
    if (k == 1) return n-1;
    if (k > n) return (josephus(n-1,k)+k)%n;
    int res = josephus(n-n/k,k)-n%k;
    return res + ((res<0)?n:res/(k-1));
}

```

```

// fast case if k=2, traditional josephus
int josephus(int n) {
    return 2*(n-(1<<(32-__builtin_clz(n)-1)));
}

```

Least Common Multiple

```

#define lcm(a,b) ((a*b)/__gcd(a,b))

```

Modulo Operations

```

#define MOD 1000000007
#define madd(a,b,m) (a+b-((a+b-m)>=0)?m:0))
#define mult(a,b,m) ((ull)a*b%m)
#define msub(a,b,m) (a-b+((a<b)?m:0))

```

```

ll mpow(ll b, ll e, ll m) {
    ll x = 1;
    while (e > 0) {
        if (e % 2) x = (x * b) % m;
        b = (b * b) % m;
        e /= 2;
    }
    return x % m;
}

```

```

ull mfac(ull n, ull m) {
    ull f = 1;
    for (int i = n; i > 1; i--)
        f = (f * i) % m;
    return f;
}

```

```

// if m is not guaranteed to be prime
ll minv(ll b, ll m) {
    ll x = 0, y = 0;
    if (egcd(b, m, x, y) != 1) return -1;
    return (x % m + m) % m;
}

```

```

ll mdiv_compmo(int a, int b, int m) {
    if (__gcd(b, m) != 1) return -1;
    return mult(a, minv(b, m), m);
}

```

```
// if m is prime (like 10^9+7)
ll mdv_primemod (int a, int b, int m) {
    return mult(a, mpow(b, m-2, m), m);
}
```

Miller-Rabin Primality Test

```
// Miller-Rabin primality test - O(10 log^3 n)
bool isPrime(ull n) {
    if (n < 2) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;

    ull s = n - 1;
    while (s % 2 == 0) s /= 2;

    for (int i = 0; i < 10; i++) {
        ull temp = s;
        ull a = rand() % (n - 1) + 1;
        ull mod = mpow(a, temp, n);
        while (temp != n - 1 && mod != 1 && mod != n
            ↪ - 1) {
            mod = mult(mod, mod, n);
            temp *= 2;
        }
        if (mod != n - 1 && temp % 2 == 0) return
            ↪ false;
    }
    return true;
}
```

Sieve of Eratosthenes

```
bitset<1000000001> sieve;

// generate sieve - O(n log n)
void genSieve(int n) {
    sieve[0] = sieve[1] = 1;
    for (ull i = 3; i * i < n; i += 2)
        if (!sieve[i])
            for (ull j = i * 3; j <= n; j += i * 2)
                sieve[j] = 1;
}
```

```
// query sieve after it's generated - O(1)
bool querySieve(int n) {
    return n == 2 || (n % 2 != 0 && !sieve[n]);
}
```

Simpson's / Approximate Integrals

```
// integrate f from a to b, k iterations
// error <= (b-a)/18.0 * M * ((b-a)/2k)^4
// where M = max(abs(f''''(x))) for x in [a,b]
// "f" is a function "double func(double x)"
double Simpsons(double a, double b, int k,
    ↪ double (*f)(double)) {
    double dx = (b-a)/(2.0*k), t = 0;
```

```
for (int i = 0; i < k; i++)
    t += ((i==0)?1:2)*(*f)(a+2*i*dx) + 4 * (*f)(
        ↪ a+(2*i+1)*dx);
return (t + (*f)(b)) * (b-a) / 6.0 / k;
}
```

Common Equations Solvers

```
// ax^2 + bx + c = 0, find x
vector<double> solveEq(double a, double b,
    ↪ double c) {
    vector<double> r;
    double z = b * b - 4 * a * c;
    if (z == 0)
        r.push_back(-b/(2*a));
    else if (z > 0) {
        r.push_back((sqrt(z)-b)/(2*a));
        r.push_back((sqrt(z)+b)/(2*a));
    }
    return r;
}
```

```
// ax^3 + bx^2 + cx + d = 0, find x
vector<double> solveEq(double a, double b,
    ↪ double c, double d) {
    vector<double> res;
    long double a1 = b/a, a2 = c/a, a3 = d/a;
    long double q = (a1*a1 - 3*a2)/9.0, sq = -2*
        ↪ sqrt(q);
    long double r = (2*a1*a1*a1 - 9*a1*a2 + 27*a3
        ↪ )/54.0;
    long double z = r*r-q*q*q, theta;
    if (z <= 0) {
        theta = acos(r/sqrt(q*q*q));
        res.push_back(sq*cos(theta/3.0) - a1/3.0);
        res.push_back(sq*cos((theta+2.0*PI)/3.0) -
            ↪ a1/3.0);
        res.push_back(sq*cos((theta+4.0*PI)/3.0) -
            ↪ a1/3.0);
    }
    else {
        res.push_back(pow(sqrt(z)+fabs(r), 1/3.0));
        res[0] = (res[0] + q / res[0]) * ((r<0)
            ↪ ?1:-1) - a1 / 3.0;
    }
    return res;
}
```

```
// m = # equations, n = # variables, a[m][n+1]
    ↪ = coefficient matrix
// a[i][0]x + a[i][1]y + ... + a[i][n]z = a[i]
    ↪ ] [n+1]
const double eps = 1e-7;
bool zero(double a) { return (a < eps) && (a >
    ↪ -eps); }
vector<double> solveEq(double **a, int m, int
    ↪ n) {
    int cur = 0;
```

```
for (int i = 0; i < n; i++) {
    for (int j = cur; j < m; j++) {
        if (!zero(a[j][i])) {
            if (j != cur) swap(a[j], a[cur]);
            for (int sat = 0; sat < m; sat++) {
                if (sat == cur) continue;
                double num = a[sat][i] / a[cur][i];
                for (int sot = 0; sot <= n; sot++)
                    a[sat][sot] -= a[cur][sot] * num;
            }
            cur++;
            break;
        }
    }
    for (int j = cur; j < m; j++)
        if (!zero(a[j][n])) return vector<double>();
    vector<double> ans(n,0);
    for (int i = 0, sat = 0; i < n; i++)
        if (sat < m && !zero(a[sat][i]))
            ans[i] = a[sat][n] / a[sat++][i];
    return ans;
}
```

6 Graph

```
struct edge {
    int u,v,w;
    edge (int u, int v, int w) : u(u), v(v), w(
        ↪ w) {}
    edge () : u(0), v(0), w(0) {}
};
bool operator < (const edge &e1, const edge &
    ↪ e2) { return e1.w < e2.w; }
bool operator > (const edge &e1, const edge &
    ↪ e2) { return e1.w > e2.w; }
```

```
struct subset { int p, rank; };
```

Eulerian Path

```
#define edge_list vector<edge>
#define adj_sets vector<set<int>>

struct EulerPathGraph {
    adj_sets graph; // actually indexes incident
    ↪ edges
    edge_list edges; int n; vector<int> indeg;
    EulerPathGraph(int n): n(n) {
        indeg = *(new vector<int>(n,0));
        graph = *(new adj_sets(n, set<int>()));
    }

    void add_edge(int u, int v) {
        graph[u].insert(edges.size());
        indeg[v]++;
        edges.push_back(edge(u,v,0));
    }
}
```

```
bool eulerian_path(vector<int> &circuit) {
    if(edges.size()==0) return false;
    stack<int> st;
    int a[] = {-1, -1};
    for(int v=0;v<n;v++) {
        if(indeg[v]!=graph[v].size()) {
            bool b = indeg[v] > graph[v].size();
            if (abs(((int)indeg[v])-((int)graph[v].size
                ↪ ())) > 1) return false;
            if (a[b] != -1) return false;
            a[b] = v;
        }
    }
    int s = (a[0]!==-1 && a[1]!==-1 ? a[0] : (a
        ↪ [0]==-1 && a[1]==-1 ? edges[0].u : -1));
    if(s==-1) return false;
    while(!st.empty() || !graph[s].empty()) {
        if (graph[s].empty()) { circuit.push_back(s
            ↪ ); s = st.top(); st.pop(); }
        else {
            int w = edges[*graph[s].begin()].v;
            graph[s].erase(graph[s].begin());
            st.push(s); s = w;
        }
    }
    circuit.push_back(s);
    return circuit.size()-1==edges.size();
}
};
```

Minimum Spanning Tree

```
// returns vector of edges in the mst
// graph[i] = vector of edges incident to
    ↪ vertex i
// places total weight of the mst in &total
// if returned vector has size != n-1, there
    ↪ is no MST
vector<edge> mst(vector<vector<edge>> graph,
    ↪ ll &total) {
    total = 0;
    priority_queue<edge, vector<edge>, greater<
        ↪ edge>> pq;
    vector<edge> MST;
    bitset<20001> marked; // change size as
        ↪ needed
    marked[0] = 1;
    for (edge ep : graph[0]) pq.push(ep);
    while(MST.size()!=graph.size()-1 && pq.size
        ↪ ()!=0) {
        edge e = pq.top(); pq.pop();
        int u = e.u, v = e.v, w = e.w;
        if(marked[u] && marked[v]) continue;
        else if(marked[u]) swap(u, v);
        for(edge ep : graph[u]) pq.push(ep);
        marked[u] = 1;
        MST.push_back(e);
        total += e.w;
    }
}
```



```

    }
    return MST;
}

```

Union Find

```

int uf_find(subset* s, int i) {
    if (s[i].p != i) s[i].p = uf_find(s, s[i].p);
    return s[i].p;
}

```

```

void uf_union(subset* s, int x, int y) {
    int xp = uf_find(s, x), yp = uf_find(s, y);
    if (s[xp].rank > s[yp].rank) s[yp].p = xp;
    else if (s[xp].rank < s[yp].rank) s[xp].p =
    ↪ yp;
    else { s[yp].p = xp; s[xp].rank++; }
}

```

7 2D Geometry

```

#define point complex<double>
double dot(point a, point b) { return real(
    ↪ conj(a)*b); }
double cross(point a, point b) { return imag(
    ↪ conj(a)*b); }

```

```

struct line { point a, b; };
struct circle { point c; double r; };
struct triangle { point a, b, c; };
struct rectangle { point tl, br; };

```

```

struct convex_polygon {
    vector<point> points;
    convex_polygon(triangle a) {
        points.push_back(a.a); points.push_back(a.b)
        ↪ ; points.push_back(a.c);
    };
    convex_polygon(rectangle a) {
        points.push_back(a.tl); points.push_back({
        ↪ real(a.tl), imag(a.br)});
        points.push_back(a.br); points.push_back({
        ↪ real(a.br), imag(a.tl)});
    }
};

```

```

#define sq(a) ((a)*(a))
double circumference(circle a) { return 2 * a.
    ↪ r * M_PI; }
double area(circle a) { return sq(a.r) * M_PI;
    ↪ }
double intersection(circle a, circle b) {
    double d = abs(a.c - b.c);
    if (d <= b.r - a.r) return area(a);
    if (d <= a.r - b.r) return area(b);
    if (d >= a.r + b.r) return 0;
    double alpha = acos((sq(a.r) + sq(d) - sq(b.r)
    ↪ )) / (2 * a.r * d));
    double beta = acos((sq(b.r) + sq(d) - sq(a.r)

```

```

    ↪ ) / (2 * b.r * d));
    return sq(a.r) * (alpha - 0.5 * sin(2 * alpha
    ↪ )) + sq(b.r) * (beta - 0.5 * sin(2 * beta
    ↪ ));
}

```

```

double intersection(rectangle a, rectangle b)
    ↪ {
    double x1 = max(real(a.tl), real(b.tl)), y1 =
    ↪ max(imag(a.tl), imag(b.tl));
    double x2 = min(real(a.br), real(b.br)), y2 =
    ↪ min(imag(a.br), imag(b.br));
    return (x2 <= x1 || y2 <= y1) ? 0 : (x2-x1)*(
    ↪ y2-y1);
}

```

8 3D Geometry

```

struct point3d {
    double x, y, z;
    point3d operator+(point3d a) const { return {
    ↪ x+a.x, y+a.y, z+a.z}; }
    point3d operator*(double a) const { return {x
    ↪ *a, y*a, z*a}; }
    point3d operator-() const { return {-x, -y, -
    ↪ z}; }
    point3d operator-(point3d a) const { return *
    ↪ this + -a; }
    point3d operator/(double a) const { return *
    ↪ this * (1/a); }
    double norm() { return x*x + y*y + z*z; }
    double abs() { return sqrt(norm()); }
    point3d normalize() { return *this / this->
    ↪ abs(); }
};

```

```

double dot(point3d a, point3d b) { return a.x*
    ↪ b.x + a.y*b.y + a.z*b.z; }
point3d cross(point3d a, point3d b) { return {
    ↪ a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z, a.x*b
    ↪ .y - a.y*b.x}; }

```

```

struct line3d { point3d a, b; };
struct plane { double a, b, c, d; } // a*x + b
    ↪ *y + c*z + d = 0
struct sphere { point3d c; double r; };

```

```

#define sq(a) ((a)*(a))
#define cb(a) ((a)*(a)*(a))
double surface(circle a) { return 4 * sq(a.r)
    ↪ * M_PI; }
double volume(circle a) { return 4.0/3.0 * cb(
    ↪ a.r) * M_PI; }

```