Articulation Point: For Bridge(low[to] > tin[v] :

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1) IS_CUTPOINT(v);
}
void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

Bellman ford:

```
vector<int> d (n, INF);
d[v] = 0;
vector<int> p (n, -1);
for (;;)
{
    bool any = false;
    for (int j = 0; j < m; ++j)
        if (d[e[j].a] < INF)
            if (d[e[j].b] > d[e[j].a] + e[j].cost)
            {
                d[e[j].b] = d[e[j].a] + e[j].cost;
                p[e[j].b] = e[j].a;
                any = true;
            }
    if (!any)  break;
}
if (d[t] == INF)
    cout << "No path from " << v << " to " << t << ".";
else
{
    vector<int> path;
    for (int cur = t; cur != -1; cur = p[cur])
        path.push_back (cur);
    reverse (path.begin(), path.end());
    cout << "Path from " << v << " to " << t << ": ";
    for (size_t i=0; i<path.size(); ++i)
        cout << path[i] << ' ';
}
```

BIT:

```
const int maxn = 1e5; int arr[maxn + 100];
void update(int indx, int n, int add){
   for(int i = indx; i <= n; i = i + (i & -i)){
       arr[i]+= add;
   } return;
}
ll query(int x){
   ll sum = 0;
   for(int i = x; i > 0; i = i - (i & -i)){
       sum += arr[i];
   }
   return sum;
}
```

BITSET:

```
bitset<32> bset1;(default constructor initializes with all bits 0)
bitset<32> bset2(20);(bset2 is initialized with bits of 20)
bitset<32> bset3(string("1100"));(bset3 is initialized with bits of
specified binary string)
bitset<8> set8;// 00000000(declaring set8 with capacity of 8 bits)
set8[1] = 1; // 00000010 // setting first bit (or 6th index)
set8[4] = set8[1]; // 00010010
int numberof1 = set8.count();// count function returns number of set bits
in
bitset.size() function returns total number of bits
in bitset.so there difference will give us number of unset(0).
int numberof0 = set8.size() - numberof1; (bits in bitset)
set8.test(i)(test function return 1 if bit is set else returns 0)
(any() function returns true, if atleast 1 bit is set )
if (!set8.any()) cout << "set8 has no bit set.\n";
(none() function returns true, if none of the bit is set)
if (!bset1.none())cout << "bset1 has some bit set\n";
cout << set8.set() << endl; (bset.set() sets all bits)
cout<<set8.set(4,0)<< endl;(bset.set(pos, b) makes bset[pos]=b)
cout << set8.set(4) << endl;
(bset.set(pos) makes bset[pos] = 1
i.e. default is 1)
cout << set8.reset(2) << endl;//reset function makes all bits 0
cout << set8.reset() << endl;
// flip function flips all bits i.e.
1 <-> 0 and 0 <-> 1
cout << set8.flip(2) << endl; // bit.flip(pos) position flip
cout << set8.flip() << endl; // full flip
// Converting decimal number to binary by using bitset
int num = 100; bitset< (bitcount) >(num);
// Converting binary to decimal number by using bitset
bitset<32> binary(arr[i]); int value = binary.to_ulong(); // to_ullong()
to_string(); Convert to string . Ex: str = mybits.to_string();
bitset<4> bset1(9); // bset1 contains 1001
bitset<4> bset2(3); // bset2 contains 0011
(bset1 == bset2) (bset1 != bset2)(comparison operator)
(bset1 ^= bset2),(bset1 &= bset2),(bset1 |= bset2)
(bitwise operation and assignment)
(bset1 <<= 2),(bset1 >>= 1)(left and right shifting)
(~bset2) // not operator
(bset1 & bset2),(bset1 | bset2),(bset1 ^ bset2)(bitwise operator)
```

Key Bitmask Tricks:

```
void set(int & num,int pos){
    num |= (1 << pos);
}
void unset(int &num,int pos){
   num &= (~(1 << pos));
}
void toggle(int &num,int pos) {
   num ^= (1 << pos);
}
bool at_position(int num,int pos) {
   bool bit = num & (1<<pos);
   return bit;
}
1) Clear all bits from LSB to ith bit
mask = ~((1 << i+1 ) - 1);
x &= mask;

2) Clearing all bits from MSB to i-th bit
mask = (1 << i) - 1;
x &= mask;

3)Count set bits in integer
int countSetBits(int x){
   int count = 0;
   while (x){
        x = x & (x-1);
        count++;
   }
   return count;
}
```

Dljkstra(Not the best):

```
const int maxn = 1e5;
struct node{
   int id, cost;
   node(){}
   node(int _id, int _cost){
       id = _id, cost = _cost;
   }
   bool operator>(const node& a) const {
       return cost > a.cost;
   }
};
priority_queue<node, vector<node>, greater<node> > pq;
vi graph[maxn + 5], w[maxn + 5];
int visit[maxn + 5], n, m, tc = 1;
std::vector<pii> travecst;

void digkstra(){
   pq.push(node(1, 0));
   while(!pq.empty()){
       node now = pq.top();
       pq.pop();
       if(visit[now.id] < 2){
           visit[now.id]++;
           for(int i = 0; i < graph[now.id].size(); i++){
               int next = graph[now.id][i];
               if(visit[next] < 2){
                   if(travecst[next].ff > now.cost + w[now.id][i] ||
travecst[next].ss > now.cost + w[now.id][i]){
                       if(travecst[next].ff > now.cost + w[now.id][i]){
                           travecst[next].ss = travecst[next].ff;
                           travecst[next].ff = now.cost + w[now.id][i];
                           pq.push(node(next, travecst[next].ff));
                           pq.push(node(next, travecst[next].ss));
                       }
                       else if(travecst[next].ss > now.cost + w[now.id][i]
&&
now.cost + w[now.id][i] != travecst[next].ff){
                           travecst[next].ss = now.cost + w[now.id][i];
                           pq.push(node(next, travecst[next].ss));
                       }
                   }
               }
           }
       }
       else{
           if(now.id == n) return;
       }
   }
   return;
}
```

DSU:

```
int par[maxn + 5];
int get(int src){
   if(par[src] < 0) return src;
   else return par[src] = get(par[src]);
}

void union_set(int a, int b){
   if(-par[b] > -par[a]) par[a] = b, par[b]--;
   else par[a] -= par[b], par[b] = a;
   return;
}
```

Geomatory:

/// Geometry - MSA
/* ---------- Basics ----------
----- Conversions -----
Degree to Radian:
Radian to Degree:
Degree to ArcMinute:
ArcMinute to Degree:
----
rad=deg*PI/180.0
deg=rad*180.0/PI
arcM=deg*60.0
deg=arcM/60.0
----- polygon -----
Area of a regular polygon:-
1) Given the side length s(The base of the triangle formed with each side with center):
Area of polygon = s*s*n/(4*tan(t))
// n=number of sides, t=PI/n
2) Given the radius r(distance from the center to any vertex):
Area of polygon = 1.0/2 * r*r*n * sin(2.0*PI/n) // n=number of sides
3) Given the apothem a(a line from the center to the midpoint of a side):
Area of polygon = a*a*n * tan(2*PI/n)
// n=number of sides

----- Circle -----
The area A of the circular segment is equal to the area of the circular sector minus the area of the triangular portion
Area of circular segment A=R*R/2.0*(Theta-sin(Theta))

----- Triangles -----
Side lengths: a, b, c
Semiperimeter: p = (a + b + c)/2
Area: A=sqrt(p*(p-a)*(p-b)*(p-c))

Circumradius: R=abc/4A
Inradius: r=A/p, A=rp
Law of sines: sin(alpha)/a = sin(beta)/b = sin(gama)/c = 1/2R
Law of cosines: a*a=b*b + c*c - 2bc cos(alpha)
Law of tangents: (a+b)/(a-b) = (tan(alpha+beta)/2.0) / (tan(alpha-beta)/2.0)Area of triangle given all medians m1, m2, m3: 4/3*Area(m1, m2, m3)
Median Mc of a triangle with length a, b, c: Mc=1/2*sqrt(2*a*a+2*b*b-c*c)
Sidelength 'c' of a triangle with medians Ma, Mb, Mc:
c
= 2/3*sqrt(2*Ma*Ma+2*Mb*Mb-Mc*Mc)
= sqrt(2*(b*b+a*a)-4Mc*Mc)
= sqrt(b*b/2-a*a+2*Mb*Mb)
Denoting the altitudes of any triangle from sides a, b, and c respectively as ha, hb, and hc,D l U _ B e l i 3 v 3 r s | 43
and denoting the semi-sum of the reciprocals of the altitudes as H=(1/ha+1/hb+1/hc)/2,
we have, area A = 1 / (4 *sqrt(H * (H-(1/ha)) * (H-(1/hb)) * (H-(1/hc))))

----- Conic/Cylinder -----
Cylinderarea: A = Pl*r*r*h
Conic area: A = 1/3 * Pl*r*r*h
Frustum area: fA = Pl*h/3 * (R*R + R*r + r*r)

----- Pick's Theorem: -----
// Only for integer points
I = area + 1 – B/2(integer division)
Where
I = number of points inside a polygon
B = number of points on the border

------- Trapezoid -------
Given Parallel side lengths a, b and height h between them, Area = (a+b)/2*h
Given four side lengths a, b, c, d where a and b are parallel, Area = (a+b)/abs(a-b)*sqrt((s-a)(s-b)(s-b-c)(s-b-d))

***/**
Formulas and Results of Straight Lines
Consider two points P(x1,y1) and Q(x2,y2), then:

01. The distance formula

$$|PQ| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

02. The midpoint formula

$$\overline{PQ} = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

03. The point R(x,y) dividing P Q (straight line) in the ratio k1 / k2 is

$$x = \frac{k_1 x_2 + k_2 x_1}{k_1 + k_2}, \quad y = \frac{k_1 y_2 + k_2 y_1}{k_1 + k_2}$$

04. The slope of P Q is =>

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

05. The slope of the x-axis = zero
06. The slope of a line parallel to the x-axis = zero
07. The slope of the y-axis is not defined, i.e. ∞
08. The slope of a line parallel to the y-axis is not defined, i.e. ∞
09. The equation of the x-axis is y = 0
10. The equation of the y-axis is x = 0
11. The equation of the line parallel to the x-axis and at a distance a is y = a.
12 . The equation of the line parallel to the y-axis and at a distance b is x = b.
13. The equation of the line with slope m and y-intercept c is y = mx + c, which is called the slope – intercept form.
14. The equation of the line passing through (x1,y1) and having the slope is y–y1 = m (x–x1), which is called the slope – point form.
15. The equation of the line passing through two points (x1,y1) and (x2,y2) is (y–y1) / (y2–y1) = (x–x1) / (x2–x1)
16. The equation of the line having a and bas the x – intercept and y – intercept is (x / a)+(y / b)=1 and is called the equation of the line in intercept form.
17. The normal form of the s traight line is x cosα + y sinα = p , where p is the length of the perpendicular from O(0,0) to the line, and α is the inclination of the perpendicular.
18. The general form of the equation of a straight line is ax+by+c=0. Consider two lines l1 and l2 having the slopes m1 and m2 , respectively.
19. If two lines l1 and l2 are parallel , then m1 = m2.
20. If two lines l1 and l2 are perpendicular , then m1 × m2 = –1 .
21. The angle θ from l1 to l2 is tanθ = (m2 – m) / (1 + m1 * m2)

22. The distance of point (x1,y1) from the line ax + by + c = 0 is ⇒ $\frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$

23. If ax+by+c=0 with b>0 is the equation of the line l, the P(x1,y1) lies:
(1) Above the line l if ax1 + by1 + c > 0
(2) Below the line l i f ax1 + by1 + c < 0
(3) On the line
l if ax1 + by1 + c = 0
24. Three lines a1x + b1y + c1 = 0, a2x + b2y + c2 = 0, a3x +b3y + c3 = 0 are concurrent (সহগামী) if

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = 0$$

Hash(Code):

```
#define base 31
#define type 'a'
#define NUM_OF_HASH 1
#define MAXN (int)1e6
ll Hash[NUM_OF_HASH][MAXN + 5];
ll power[NUM_OF_HASH][MAXN + 5];
ll HMOD[] = {1000000007, 1000000009, 998244353,
1000000037,1000000021,1000000003,1000005133};
string str;
int len;

void pre(){
   for(int i=0;i<NUM_OF_HASH;i++){
       power[i][0] = 1;
       Hash[i][0] = str[0]-type+1;
       for(int j=1;j<=len;j++) {
           power[i][j] = (power[i][j-1]*base)%HMOD[i];
           if(j != len)
           Hash[i][j] = (Hash[i][j-1]*base + str[j]-type+1)%HMOD[i];
        }
   }
   return;
}
ll getHash(int i,int j,int k){
   if(!i) return Hash[k][j] % HMOD[k];
   return ((Hash[k][j] - (Hash[k][i - 1] * power[k][j - i + 1]) % HMOD[k])
+ HMOD[k] + HMOD[k] + HMOD[k] + HMOD[k] + HMOD[k]) % HMOD[k];
}

ll cngHash(int pos, char ch, int len, int k){
   return (((Hash[k][pos - 1] * base + ch - type + 1) * power[k][len - pos
- 1]) % HMOD[k] + getHash(pos + 1, len - 1, k) ) % HMOD[k];
}

ll singleHash(string &str, int k){
   ll hashv = str[0] - type + 1;
   for(int i = 1; i < str.size(); i++){
       hashv = (hashv * base + str[i] - type + 1) % HMOD[k];
   }
   return hashv;
}
```

KMP:

```
void KMPSearch(char* pat, char*
txt)
{
   int M = strlen(pat);  int N =
strlen(txt);
   // create lps[] that will hold
the longest prefix suffix
   int lps[M];
   computeLPSArray(pat, M, lps);
   int i = 0; // txt[]
   int j = 0; // pat[]
   while (i < N) {
       if (pat[j] == txt[i]) {
           j++,  i++;
       }
       if (j == M) {
           printf("Found pattern at
index %d ", i - j);
           j = lps[j - 1];
       }
       else if (i < N && pat[j] !=
txt[i]) {
           if (j != 0)  j = lps[j -
1];
           else
               i = i + 1;
       }
   }
}
// Fills lps[] for given patttern
pat[0..M-1]
void computeLPSArray(char* pat, int
M, int* lps){
   int len = 0;
   lps[0] = 0; // lps[0] is always
0
   int i = 1;
   while (i < M) {
       if (pat[i] == pat[len]) {
           len++;
           lps[i] = len;
           i++;
       }
       else{
           if (len != 0) {
               len = lps[len - 1];
           }
           else {
               lps[i] = 0;
               i++;
           }
       }
   }
}
```

KMP(Shafin Vai):

```
for (int R = 1; R < len; R++)
   {
       int L = pi[R-1];
       while(L > 0 && S[L] != S[R])
       {
           L = pi[L-1];
       }
       pi[R] = L + (S[L] == S[R]);

       if (pi[R] == n){
printf("%d\n", R-(n+n)); flag =
true; }
   }
```

LCA Using Sparse Table:

```
// Sparse Matrix DP approach to find LCA of
two nodes
#include <bits/stdc++.h>
using namespace std;
#define MAXN 100000
#define level 18

vector <int> tree[MAXN];
int depth[MAXN];
int parent[MAXN][level];
// pre-compute the depth for each node and
their
// first parent(2^0th parent)
// time complexity : O(n)
void dfs(int cur, int prev)
{
   depth[cur] = depth[prev] + 1;
   parent[cur][0] = prev;
   for (int i=0; i<tree[cur].size(); i++)
   {
      if (tree[cur][i] != prev)
         dfs(tree[cur][i], cur);
   }
}
// Dynamic Programming Sparse Matrix
Approach
// populating 2^i parent for each node
// Time complexity : O(nlogn)
void precomputeSparseMatrix(int n)
{
   for (int i=1; i<level; i++)
   {
      for (int node = 1; node <= n; node++)
      {
         if (parent[node][i-1] != -1)
            parent[node][i] =
               parent[parent[node][i-1]][i-1];
      }
   }
}

// Returning the LCA of u and v
// Time complexity : O(log n)
int lca(int u, int v)
{
   if (depth[v] < depth[u])
      swap(u, v);

   int diff = depth[v] - depth[u];

   // Step 1 of the pseudocode
   for (int i=0; i<level; i++)
      if ((diff>>i)&1)
         v = parent[v][i];
   // now depth[u] == depth[v]
   if (u == v)
      return u;

   // Step 2 of the pseudocode
   for (int i=level-1; i>=0; i--)
      if (parent[u][i] != parent[v][i])
      {
         u = parent[u][i];
         v = parent[v][i];
      }

   return parent[u][0];
}

void addEdge(int u,int v)
{
   tree[u].push_back(v);
   tree[v].push_back(u);
}

// driver function
int main()
{
   memset(parent,-1,sizeof(parent));
   int n = 8;
   addEdge(1,2);
   addEdge(1,3);
   addEdge(2,4);
   addEdge(2,5);
   addEdge(2,6);
   addEdge(3,7);
   addEdge(3,8);
   depth[0] = 0;

   // running dfs and precalculating depth
   // of each node.
   dfs(1,0);

   // Precomputing the 2^i th ancestor for
evey node
   precomputeSparseMatrix(n);

   // calling the LCA function
   cout << "LCA(4, 7) = " << lca(4,7) << endl;
   cout << "LCA(4, 6) = " << lca(4,6) << endl;
   return 0;
}
```

MO'S ON DSU:

```
const int maxn = 5e4;
int par[maxn + 5];
stack< pair<pii,int> > updates;
vector<pii>edge;
vector<pair<pii, int>> qrs[230];
int ans[maxn + 5];
int get(int x){
   return par[x] < 0 ? x : get(par[x]);
}

int union_set(int x, int y){
   x = get(x), y = get(y);
   if(x == y){
       updates.push({{-1, -1}, -1});
       return 0;
   }
   if(par[x] > par[y]) swap(x, y);
   updates.push({ {x, y}, par[y]} );
   par[x] += par[y], par[y] = x;
   return 1;
}
int rollback(){
   int cnt = 0;
   while(!updates.empty()){
       pair<pii, int> now;
       now = updates.top();
       updates.pop();
       if(now.ff.ff == -2) break;
       if(now.ff.ff != -1){
           cnt++;
           par[now.ff.ff] -= now.ss;
           par[now.ff.ss] = now.ss;
       }
   }
   return cnt;
}
bool cmp(pair<pii, int> &a, pair<pii, int> &b){
   return a.ff.ss < b.ff.ss ||
    (a.ff.ss == b.ff.ss && a.ff.ff < b.ff.ff);
}



void solve(){
   int n, m, u, v, q, l, r; sc2(n, m); edge.resize(m);
   for(int i = 0; i < m; i++){
       sc2(u, v);
       edge[i] = {u, v};
```

```
    }
    sc1(q);  int sq = sqrt(m);
    for(int i = 0; i < q; i++){
        sc2(l , r);  l--, r--;
        qrs[l / sq].pb({{l, r}, i});
    }
    for(int i = 0; i < sq + 1; i++) sort(all(qrs[i]), cmp);

    for(int i = 0; i <= sq + 1; i++){
        for(int t = 0; t <= n; t++) par[t] = -1;

        while(!updates.empty()) updates.pop();
        int res = n, rn = (i + 1) * sq;
        for(int j = 0; j < qrs[i].size(); j++){
            int low = i * sq, high = (i + 1) *  sq - 1;

            if(low <= qrs[i][j].ff.ff  && qrs[i][j].ff.ss <= high){
                for(int x = qrs[i][j].ff.ff; x <= qrs[i][j].ff.ss; x++){
                    res -= union_set(edge[x].ff, edge[x].ss);
                }
                ans[qrs[i][j].ss] = res;
                res += rollback();
            }
            else{
                pair<pii, int> now = qrs[i][j];
                while(rn <= now.ff.ss){
                    res -= union_set(edge[rn].ff, edge[rn].ss);  rn++;
                }
                updates.push({{-2, -2}, -2});
                int ln = (i + 1) * sq - 1;
                while(ln >= now.ff.ff){
                    res -= union_set(edge[ln].ff, edge[ln].ss); ln--;
                }
                ans[now.ss] = res;
                res += rollback();
            }
        }
    }
    for(int i = 0; i < q; i++){
        printf("%d\n", ans[i]);
    }
    return;
}
```

Number theory:

Topic Modular multiplicative inverse:(x/a)%m and Fermat's little theorem:

Works when m is prime;

If we know m is prime, then we can also use Fermats's little theorem to find the inverse.

a^m-1 ≅ 1 (mod m)

If we multiply both sides with a-1, we get

a^-1 ≅ a^m-2 (mod m)

(X/a) % m = x%m * (a^-1)%m == ((x % m) * bigmod(a, m - 2)) % m;

This is also true: (a * a^-1) % m == 1

Fermat's little theorem:

Fermat's little theorem states that

if p is a prime number, then for any integer a, the number a^p − a is an integer multiple of p. Here p is a prime number
a^p ≡ a (mod p).
Special Case: If a is not divisible by p,
Fermat's little theorem is equivalent to the statement that a^(p-1)-1 is an integer multiple of p.
a^(p-1) ≡ 1 (mod p)
OR
a^(p-1) % p = 1
Here a is not divisible by p.

Prime Count(10 - 10^26):

4 | 25 | 168 | 1,229 | 9,592 | 78,498 | 664,579 | 5,761,455 | 50,847,534 | 455,052,511 | 4,118,054,813 | 37,607,912,018 | 346,065,536,839 | 3,204,941,750,802 | 29,844,570,422,669 | 279,238,341,033,925 | 2,623,557,157,654,233 | 24,739,954,287,740,860 | 234,057,667,276,344,607 | 2,220,819,602,560,918,840 | 21,127,269,486,018,731,928 | 201,467,286,689,315,906,290 | 1,925,320,391,606,803,968,923 | 18,435,599,767,349,200,867,866 | 176,846,309,399,143,769,411,680

NCR:

```
ll fact[maxn + 5];
void factgen(){
   fact[0] = 1;
   FOR(i, 1, maxn + 2){
       fact[i] = (fact[i - 1] * i) % MOD;
   }
   return;
}
ll ncr(ll n, ll r){
   ll ret = (fact[n] * bigmod( ( fact[r] * fact[n - r]) % MOD , MOD - 2)) %
MOD;
   return ret;
}
```

Key Formula OF Number Theory:

1.No divisor / factors of n = (1 + a)(1 + b)(1 + c)...(a + r) [n = p1 ^ a * p2^b...pn ^ r]

2.Sum of divisor of n = (1 + p1 + p1^2 + ..p1^a)(1 + p2 + p2^2 + ..p2^b)...(1 + pn + pn^2 + ..pn^r)
n = (p1 ^ (a + 1) - 1) / (p1 - 1)...(pn ^ (r + 1) - 1) / (pn - 1) [1 + x + x^2...xn = (x ^ (n + 1) - 1) / (x - 1)]

3.phi(n) = n - 1 [If n is prime]
4.phi(p^a) = p^a ((p - 1) / p) [p is prime]
5.phi(n) = n * ((p1 - 1) / p1) * ((p2 - 1) / p2)...((pn - 1) / pn)
6.phi(nm) = (phi(n) * phi(m) * d) / phi(d) [d = gcd(n, m)]
7.If d1, d2, d3..dn is the divisor of n Then [n = phi(d1) + phi(d2) … + phi(dn)

Prime Sieve:

```
const int maxn = 5e6 + 5;
ll factof[maxn + 5];
vi primes;
void factgen(){
   for(int i = 1; i <= maxn; i++){
       if( !(i & 1) ) factof[i] = 2;
       else factof[i] = i;
   }

   for(int i = 3; i * i <= maxn; i+= 2){
       if(factof[i] == i){
           for(int j = i * i; j <= maxn; j += i){
               if(factof[j] == j) factof[j] = i;
           }
       }
   }
   for(int i = 2; i <= maxn; i++){
       if(factof[i] == i) primes.pb(i);
   }
   return;
}
```

Phi in Linear time:

```
const int N = 10000000;
int lp[N + 1];
int phi[N + 1];
vector<int> pr;

void calc_sieve()
{
    phi[1] = 1;
    for (int i = 2; i <= N; ++i)
    {
        if (lp[i] == 0)
        {
            lp[i] = i;
            phi[i] = i - 1;
            pr.push_back(i);
        }
        else
        {
            //Calculating phi
            if (lp[i] == lp[i / lp[i]])
                phi[i] = phi[i / lp[i]] * lp[i];
            else
                phi[i] = phi[i / lp[i]] * (lp[i] - 1);
        }
        for (int j = 0; j < (int)pr.size() && pr[j] <= lp[i] && i * pr[j] <= N; ++j)
            lp[i * pr[j]] = pr[j];
    }

}
```

Sieve in Linear Time Complexity:

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
   if (lp[i] == 0) {
       lp[i] = i;
       pr.push_back (i);
   }
   for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N; ++j)
       lp[i * pr[j]] = pr[j];
}
```

Miller Robbin for big number:

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64
mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result *
base % mod;
        base = (u128)base * base %
mod;
        e >>= 1;
    }
 return result;
}
bool check_composite(u64 n, u64 a,
u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};
bool MillerRabin(u64 n, int iter=5)
{ // returns true if n is probably
prime, else returns false.
    if (n < 4)
        return n == 2 || n == 3;
    int s = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        s++;
    }
    for (int i = 0; i < iter; i++) {
        int a = 2 + rand() % (n -
3);
        if (check_composite(n, a, d,
s))
            return false;
    }


   return true;
}
```

//Miller Rabin 100%(Accuracy 32 - 64 bit int) 32(Take first 4prime) - 64(Take first 12 Prime)

```
bool MillerRabin(u64 n) { //
returns true if n is prime,
else returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7,
11, 13, 17, 19, 23, 29, 31,
37}) {
        if (n == a)
            return true;
        if (check_composite(n,
a, d, r))
            return false;
    }
    return true;
}
```

Segment Tree:

```
const int maxn = 1e5;
int seg[4 * maxn + 5], n, m;
vector<int> v;

void build(int indx, int l, int r){
   if(l == r){
       //inti
       return;
   }
   int mid = l + (r - l) / 2;
   build(indx << 1, l, mid);
   build( (indx << 1) + 1, mid + 1, r);
   //seg[indx] = seg[indx << 1] * seg[ (indx << 1) + 1];     //marge
   return;
}

void update(int indx, int l, int r, int i, int value){
   if(l > i || r < i) return;
   if(l == r && l == i){
       //change;
       return;
   }
   int mid = l + ( r - l ) / 2;
   update(indx << 1, l, mid, i, value);
   update((indx << 1) + 1, mid + 1, r, i, value);
   //seg[indx] = seg[indx << 1] * seg[(indx << 1) + 1]; // marge
   return;
}
int query(int indx, int l, int r, int i, int j){
   if(r < i || j < l) return 1; //invalid
   if( i <= l && r <= j) return seg[indx]; // insegment

   int mid = l + (r - l) / 2;
   int a = query(indx << 1, l, mid, i, j);
   int b = query((indx << 1) + 1, mid + 1, r, i, j);
   return a * b; // marge and return;
}
```

SOS DP:

```
for(int i = 0; i < p; i++){
   for(int mask = 0; mask < (1 << p); mask++){
       if( !(mask & (1 << i)) ){
               dp[mask] += dp[mask | (1 << i)];
       }
   }
}
```

Sparse Table:

```
const int maxn = 1e5;
int table[maxn + 5][32], n, m;
vi v(maxn + 5);
void preprocess(){
    for(int i = 0; i < n; i++){
        table[i][0] = v[i];
    }
    for(int j = 1; j < 32; j++){
        int lim = 1 << j;
        if(lim > n) break;
        for(int i = 0; i < n; i++){
            if(i + lim > n) break;
            int a = table[i][j - 1],
b = table[i + (lim >> 1)][j - 1];
            table[i][j] = min(a, b);
        }
    }
}

int query(int l, int r){
    int dist = r - l + 1;
    ll res = INF;
    for(int i = 0; i < 32; i++){
        if( dist & (1 << i)){
            res = min(res, (ll)
table[l][i]);
            l += (1 << i);
        }
    }
    return res;
}
```

SQRT Decomposition:

```
const int maxn = 1e3;
int seg[maxn + 5];
void pri_process(int n, vi
&v){//initialize the segments for the
first time from v;
    int sq = sqrt(n);
    for(int i = 0; i < maxn; i++)
seg[i] = inf;
    for(int i = 0; i < n; i++){
        seg[i / sq] = min(v[i], seg[i /
sq]); // do someting;
    }
    return;
}
void update(int n, vi &v, int indx,
int x){ // Chanaging to v[indx] = x;
    v[indx] = x;
    int sq = sqrt(n), st = (indx / sq)
* sq, ed  = min(st + sq - 1, n - 1);
    FOR(i, st, ed + 1) seg[i / sq] =
min(v[i], seg[i / sq]);
    return;
}
int query(int n, vi &v, int l, int r){
// l to r Inclusive
    int sq = sqrt(n), st = (l / sq) *
sq, ed = min( st + sq - 1, r);
    int sum = inf;
    for(int i = l; i <= ed && i < n;
i++){ // first segment
        sum =  min(v[i], sum);
    }
    st = ed;
    for(int i = st + sq; i < n && i <=
r; i+= sq){ // middle segmetns;
        sum =  min(sum, seg[i / sq]);
        st = i;
    }
    st++;
    for(int i = st; i < n && i <= r;
i++) sum =  min(v[i], sum); // last
segment;
    return sum;
}
```

STL:

Vector: vector <T> v; (Dynamic array.)

Some Facilities :
std::vector<int>::iterator it; (iterator = pointer for container)
Operator[] Access element. Vector support Random access..
Ex: v[i] access the value of i th index.
front(); Access first element .Ex : v.front();
back(); Access last element. Ex : v.back();
assign();, Assign vector content. Ex; v1 = v2, second.assign (it,first.end()-1); , third.assign (myints,myints+3);
push_back(); Add element at the end. Ex: v.push_back(value);
pop_back(); Delete last element. Ex: v.pop_back();

upper_bound(); It returns an iterator pointing to the first element in the range [first, last) that is greater than value, or last if no such element is found. Ex: upper1 = upper_bound(v.begin(), v.end(), value);
lower_bound();The lower_bound() method in C++ is used to return an iterator pointing to the
first element in the range [first, last) which has a value not less than val. This means that the
function returns the index of the next smallest number just greater than that number.
Ex:lower_bound(v.begin(), v.end(), value);

insert(); Insert elements. Ex: it = myvector.insert (it , cnt , value);
myvector.insert (it+ x
,anothervector.begin(),anothervector.end());(inserting another vector value to my
vector at position (x +1) 1 base index for 0 base index at position (x))
erase() Erase elements.
Ex: myvector.erase (myvector.begin()+5); (erase the 6th element);
myvector.erase(myvector.begin(),myvector.begin()+3); erase the first 3 elements:
swap() Swap content. Ex: v1.swap(v2); (v1 <====> v2)
clear() Clear content. Ex: myvector.clear();
size() Returns the number of elements in the vector. Ex: myvector.size();

Map: map<key,value> name; (array with container index)

Some Facilities : It has some pair< T , T > p; facilities
std::map<key,value>::iterator it; (iterator = pointer for container)
it->first returns the key . it->second returns the value. (for pointer use -> for normal . (dot))
Operator[] Access element (public member function). Map does not support Random access..
But it supports sequential access. Ex: mp[i] access the value of i th index.
insert() . Ex: mp.insert ({'a',100});
,mp2.insert(mp.begin(),mp.find('c')); (inserting value to mp2 from mp range (from first to till key 'c' -1 inclusively)).

```
find() Get iterator to element / key (public member function ) Ex : it =
mymap.find('b');
erase() Erase elements (public member function).
Ex: mymap.erase (it); (erasing by iterator) , mymap.erase ('c'); (erasing
by key)
mymap.erase ( it, mymap.end() );    // erasing by range
lower_bound() Return iterator to lower bound . Ex: itlow=mymap.lower_bound
('b');
upper_bound() Return iterator to upper bound .Ex:itup = mymap.upper_bound
('d');  // itup points to e (not d)  // print range [itlow,itup):
for (it=itlow; it!=itup; ++it)  std::cout << (*it).first << " => " <<
(*it).second << '\n';
count() Count elements with a specific key . Ex : mymap.count('c')
```

Set: set< T > st;

```
Sets are containers that store unique elements following a specific order.
Some Facilities :
Support functions (insert(), find(), erase(), lower_bound(), upper_bound(),
::iterator)
Can use like map( stl ) . Ex: st.insert(value);

Stack: stack<T> st;
Stacks are a type of container adaptor, specifically designed to operate in
a LIFO context (last-in first-out), where elements are inserted and
extracted only from one end of the container.
Some Facilities :
top() Access next / top / newest  element. Ex: mystack.top() += 10 ;
// increasing 10 to the top element.
push() Insert element. Ex: for (int i=0; i<5; ++i) mystack.push(i);
pop() Remove top element. Ex: mystack.pop();
```

std::multiset<T> mymultiset;

```
Multisets are containers that store elements following a specific order,
and where multiple elements can have equivalent values.
gquiz2.erase(gquiz2.begin() , gquiz2.find(30) // with value 30 in gquiz2
auto it = mymultiset.lower_bound (30);
mymultiset.count(value) Count elements with a specific key.
```

Order Set: ordered_set o_set;

//Must include

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type,less<int>,
rb_tree_tag,tree_order_statistics_node_update>
//-----------------------------------------------------------------------
```

int : It is the type of the data that we want to insert (KEY).It can be integer, float or pair of int etc.
null_type : It is the mapped policy. It is null here to use it as a set.If we want to get map but not the set, as the second argument type must be used mapped type.
less : It is the basis for comparison of two functions.
rb_tree_tag : type of tree used. It is generally Red black trees because it takes log(n) time for insertion and deletion while other take linear time such as splay_tree.
tree_order_statistics_node__update : It is included in tree_policy.hpp and contains various operations for updating the node variants of a tree-based container, so we can keep track of metadata like the number of nodes in a subtree

Additional functions in the ordered set other than the set

o_set.order_of_key (k) : Number of items strictly smaller than k .
o_set.find_by_order(k) : K-th element in a set (counting from zero).

Queue: queue<T> myqueue;

queues are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.
Some Facilities :
front() Access next / front / oldest element. Ex: myqueue.front()
back() Access last / back / newest element. Ex: myqueue.back() -= myqueue.front();
push() Inserts a new element at the end of the queue, after its current last element.
EX: myqueue.push (5);
pop() Removes the next element in the queue, effectively reducing its size by one. Ex. Q.pop()

Priority Queue: priority_queue< T> mypq;

Priority queues are a type of container adaptors, specifically designed such that its first element is always the greatest of the elements it contains, according to some strict weak ordering criterion.
Some Facilities:
top() Access top element. Ex: mypq.top();
push() Inserts a new element in the priority_queue. Ex: mypq.push(5);
pop() Removes the element on top of the priority_queue, effectively reducing its size by one. The element removed is the one with the highest value. Ex: mypq.pop();
less<int> (50 (priority_queue top element),25,12 , 1)
Ex: priority_queue<long long,vector<long long>,less<long long>> mypq;
greater<int> (1 (priority_queue top element),12,25,50)
Ex: priority_queue<long long,vector<long long>,greater<long long>> mypq;

Deque: deque< T > mydeque;.Double ended queue

deque (usually pronounced like "deck") is an irregular acronym of double-ended queue. Double-ended queues are sequence containers with

```
dynamic sizes that can be expanded or contracted on both ends (either its
front or its back). (two way vector)
Some Facilities : similar with vector but it is two way.
Operator[]  Access element. deque support Random access..
Ex: mydeque[i] access the value of i th index.
front() Access first element.Ex : dq.front();
back() Access last element. Ex : dq.back();
push_back() Add element at the end. Ex: dq.push_back(value);
pop_back() Delete last element. Ex: dq.pop_back();
push_front() Insert element at beginning. Ex: dq.push_front(value);
pop_back() Delete last element. Ex: dq.pop_front();
pop_front() Delete first element. Ex;dq.pop_front();
```

Other functions are like vector.

```
Some common function for all stl:
Example for vector it will work for all other type:
vector<int>::iterator it;
begin() Return iterator to beginning. Ex: it = myvector.begin();
end() Return iterator to end.Ex: it = myvector.begin();
vector <int>:: reverse_iterator rit; (for using rbegin(),rend() we must use
reverse_iterator)
rbegin() Return reverse iterator to reverse beginning.Ex: rit =
myvector.rbegin();                rend() Return reverse iterator to reverse
end.Ex: rit = myvector.rend();
erase() Erase elements.
Ex: myvector.erase (myvector.begin()+5); (erase the 6th element);
myvector.erase(myvector.begin(),myvector.begin()+3);  erase the first 3
elements:
swap() Swap content. Ex: v1.swap(v2);  (v1 <====> v2)
clear()  Clear content. Ex: myvector.clear();
size() Returns the number of elements in the vector. Ex: myvector.size();
-------------------------------------------------------------------------
---
std::reverse() C++ :  It reverses the order of the elements in the range
[first, last) of any
Container.
For all the container:  reverse(v.begin() + 5, v.begin() + 8);
For array: reverse(std::begin(a), std::end(a));
```

My Code Base:

```
#include <bits/stdc++.h>
using namespace std;
//_____________________________________________________________________
_____________________________________________
#define PI  2*acos(0.0)
#define pf  printf
#define sc  scanf
#define ff  first
#define ss  second
#define pb  push_back
typedef long long ll;
typedef unsigned long long ull;
typedef std::vector<int> vi;
typedef vector<long long> vll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
#define sc1(n)          sc("%d",&n)
#define sc2(n, m)       sc("%d%d", &n, &m)
#define sc3(m, n, o)    sc("%d%d%d", &m, &n, &o)
#define scl(n)          sc("%lld", &n)
#define scll(n, m)      sc("%lld%lld", &n, &m)
#define sclll(n, m, o) sc("%lld%lld%lld", &n, &m, &o)
#define scf(f)          sc("%lf",&f);
#define pn(n)           pf("%d\n", n);
#define FOR(i,a,n)   for( int i = a; i < n; i++)
#define all(x) (x).begin(), (x).end()
#define FastIO ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int inf = 0x3f3f3f3f;
const int MOD =  1e9 + 7;
const long double EPS = 1e-9;
template <class T> inline T gcd(T a,T b){if(b==0)return a;return
gcd(b,a%b);}
template <class T> inline double my_sqrt(T n) { double high = n + 5, low =
0, mid, ans; int cnt = 100; while(cnt--) { mid = low + (high - low) / 2;
if(mid * mid <= n) ans = mid, low = mid; else high = mid; } return ans; }
template <class T> inline T bigmod(T b, T p){ if(p <= 0 || b == 0) return
1; ll x = b; if(p & 1) return (x * bigmod(b, p - 1)) % MOD; x = bigmod(b, p
>> 1); return (x * x) % MOD;}
#ifdef PARTHO
#define dbg(x) cout << __LINE__ << " says: " << #x << " = " << x << "\n"
#else
#define dbg(x)
#endif
//_____________________________________________________________________
_____________________________________________

void solve(){

   return;
}
int main() {
```

```
    #ifdef PARTHO
        freopen("/mnt/Stable/Dropbox/IO/input.txt","r",stdin);
        freopen("/mnt/Stable/Dropbox/IO/output.txt","w",stdout);
        int start_time = clock();
    #endif
    //FastIO;

    int test = 1;
    sc1(test);
    while(test--){
        solve();
    }
    #ifdef PARTHO
        int end_time = clock();
        printf("Time = %.4f\n",(end_time-start_time+0.0)/CLOCKS_PER_SEC);
    #endif

    return 0;
}
///Before submit=>
///    *check for integer overflow,array bounds
///    *check for n=1
```

Rand:

```
const int N = 300;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count()); ///
MUST ADD
double average_distance(const vector<int> &permutation){
    double distance_sum = 0;
    for (int i = 0; i < N; i++)
        distance_sum += abs(permutation[i] - i);
    return distance_sum / N;
}
int getRandom(int L,int R) /// generate random numbers in range [L,R] {
    return rng()%(R-L+1) + L;
}
int cal(int x,int d){
 return (x +  ceil((float)d/(x+1)));
}
```

//File Compair:

```
int main(){
 char file1[100] = "/mnt/Stable/Dropbox/IO/output.txt ";
 char file2[100] = "/mnt/Stable/Dropbox/IO/coutput.txt";
 char command[100] = "diff -s ";
 strcat(file1, file2);
 strcat(command, file1);
 system(command);
 return 0;
}
```