Mental Freeze Reference:

Template:

```
#include <bits/stdc++.h>
#define endl "\n"
#define debug(a) cout << #a << ": " << a << endl
#define debLine() cout << "==============" << endl
#define test() int t; cin >> t; while(t--)
#define all(a) a.begin(), a.end()
#define fillWith(a, b) memset(a, b, sizeof(a))
#define Mod 1000000007
#define F first
#define S second
#define pb push_back
#define goFast() ios::sync_with_stdio(0); cin.tie(0);
cout.tie(0)
#define files(x) freopen(x, "r", stdin)
typedef long long ll;
typedef long double ld;
using namespace std;

int main()
{
    goFast();
}
```

Graph Algorithms:

```
int n, m, visited[200200];
vector<int> myGraph[200200];
```

```
void addWeightedEdge(int u, int v, int w){
    myGraph[u].push_back({w,v});
    myGraph[v].push_back({w,u});
}
```

DFS:

```
void dfs(int u){
    if (visited[u])
        return;
    visited[u] = 1;
    for (int v : myGraph[u])
        dfs(v);
}
```

BFS:

```
void bfs(int u){
    queue<int> q;
    q.push(u);
    visited[u] = true;
    while(!q.empty()){
        int s = q.front();
        q.pop();
        for(int v : myGraph[s]){
            if(!visited[v]){
                visited[v] = true;
                q.push(v);
            }
        }
    }
}
```

BFS ShortestPath:

```
vector<int> BFSshortestPath(int start){
    queue<int> q;
    vector<int> distance(n + 1, 1e8);
    q.push(start);
    distance[start] = 0;
    while (!q.empty()){
        int parent = q.front();
        q.pop();
        for (int son : myGraph[parent]){
            if (distance[son] > distance[parent] + 1){
                distance[son] = distance[parent] + 1;
                q.push(son);
            }
        }
    }
    return distance;
}
```

01BFS:

```
vector<int> 01Bfs(int start){
    deque<int> q;
    vector<int> distance(n + 1, 1e8);
    q.push_front(start);
    distance[start] = 0;
    while (!q.empty()){
        int parent = q.front();
        q.pop_front();
        for (auto pairSon : MyGraph[parent]){
            int son = pairSon.first;
            int weight = pairSon.second;
            if (distance[son] > distance[parent] + weight){
                distance[son] = distance[parent] + weight;
                if (weight == 0)
                    q.push_front(son);
                else
                    q.push_back(son);
            }
        }
    }
    return distance;
}
```

Dijkstra:

```
vector<ll> Dijkstra(int source){
    priority_queue<pair<ll, int>> q;
    vector<ll> Dist(n + 1, 1e18);
    Dist[source] = 0;
    q.push({Dist[source], source});
    while (!q.empty()){
        pair<ll, int> Top = q.top();
        q.pop();
        int Parent = Top.second;
        ll DistParent = (-1) * Top.first;
        if (DistParent > Dist[Parent])
            continue; // Very Important
        for (auto PairSon : MyGraph[Parent]){
            int son = PairSon.second;
            ll Weight = PairSon.first;
            if (DistParent + Weight < Dist[son]){
                Dist[son] = DistParent + Weight;
                q.push({(-1) * Dist[son], son});
            }
        }
    }
    return Dist;
}
```

Floyd:

```
int Distance[555][555];
void Initiate(){
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            Distance[i][j] = 1e9;
    for(int i = 1; i <= n; i++)
        Distance[i][i] = 0;
}
void Floyd(){
    for(int k = 1; k <= n; k++)
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                Distance[i][j] = min(Distance[i][j],
Distance[i][k] + Distance[k][j]);
}
```

DSU:

```
int parent[200200], sizee[200200], numberOfComponents = n;
int root(int x){
    if (x == parent[x])
        return x;
    return parent[x] = root(parent[x]);
}
void Union(int x, int y){
    int rx = root(x);
    int ry = root(y);
    if (rx != ry){
        parent[rx] = ry;
        sizee[ry] += sizee[rx];
        numberOfComponents--;
    }
}
for (int i = 1; i <= n; i++){ // In Main
     parent[i] = i;
     sizee[i] = 1;
}
```

Bipartite Graph:

```
int color[200005];
bool isBipartite = true;
void dfsIsBipartite(int u, int c) // c is 0 initially{
    color[u] = c;
    for (int v : myGraph[u]){
        if(color[v] == -1)
            dfsIsBipartite(v, 1 - c); // 1 -> 0 or 0->1
        else
            if(color[u] == color[v])
                isBipartite = false;
    }
}
dfsIsBipartite(1, 0); // In Main
```

MST:

```
int a[200200], pa[200200], n, fix, t;
int root(int x){
    return pa[x] == x ? x : pa[x] = root(pa[x]);
}
ll Mst(vector<pair<int, pi>> v){
    ll ans = 0;
    sort(v.begin(), v.end());
    for(int i = 1; i <= n; i++)
        pa[i] = i;
    for(pair<int, pi> xx : v){
        int w = xx.F;
        int x = xx.S.F;
        int y = xx.S.S;
        x = root(x);
        y = root(y);
        if (x != y)
        {
            pa[x] = y;
            ans += w;
        }
    }
    return ans;
}
int main(){
    vector<pair<int, pi>> v;
    cout << Mst(v) << endl;
}
```

Graph On Grid:

```
int N;
bool vis[N][N];
// direction vectors:
int dr = {-1, 1, 0, 0};
int dc = {0 , 0, 1, -1};
void dfsGrid(int i, int j){
    if(vis[i][j]) return;
    vis[i][j] = true;
    for(int k=0;k<4;i++){
        int r = i + dr[k];
        int c = j + dc[k];
        if(r < 0 || c < 0 || r > N || c > N) continue;
        dfsGrid(r,c);
    }
}
```

1D - Prefix Sum:

```
int n;
int arr[500005];
ll prefixSum[500005];

// !: ZERO - BASED
void generatePrefixSumArray(){
    for(int i = 0; i < n; i++)
        prefixSum[i] = (i == 0) ? arr[i] : arr[i] + prefixSum[i
- 1];
}
ll getPrefixSumQRY(int L, int R){
    if(L == 0)
        return prefixSum[R];
    return prefixSum[R] - prefixSum[L - 1];
}
```

2D - Prefix Sum:

```
// *: Contains the prefix sum of each element in 2D.
int prefixSum[N][N]; // !: Initialized with zeros.
// *: Calculates the sum of the rectangle with the given indexes.
int sumQuery(int i, int j, int k, int l){
    return prefixSum[k][l] - prefixSum[i - 1][l] - prefixSum[k][j - 1] + prefixSum[i - 1][j - 1];
}
int main(){
    goFast();
    int n;
    cin >> n;
    //*: Taking input.
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            cin >> prefixSum[i][j];
    // *: Accumulating rows.
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            prefixSum[i][j] += prefixSum[i][j - 1];
    // *: Accumulating Columns.
    for(int j = 1; j <= n; j++)
        for(int i = 1; i <= n; i++)
            prefixSum[i][j] += prefixSum[i - 1][j];
    // *: Generating 4 points for the rectangle and saving the maximum sum.
    int maxi = 0;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
```

```
            for(int k = i + 1; k <= n; k++)
                for(int l = j + 1; l <= n; l++)
                    maxi = max(maxi, sumQuery(i, j, k, l));
}
```

Sorting:

```
bool sortBySecond(const pair<int, int> &a, const pair<int, int> &b){
    // if (a.second < b.second)
    //     return true;
    // else
    //     return false;
    return (a.second < b.second); // MUST BE LIKE THIS
}
sort(_.begin(), _.end(), sortBySecond); // In Main
```

```
set<int, greater<int>> st; // Set in Descending order
priority_queue<int, vector<int>, greater<int>> qu;
```

Math Algorithms:

Power:

```
int fastestPower(int x, int y){
    if(y == 0)
        return 1;
    if(y & 1)
        return x * fastestPower(x, y - 1);
    int z = fastestPower(x, y / 2);
    return z * z;
}
```

LOG:

```
// *: How to calculate the log of a custom base?
int number = 256, base = 4;
int x = log(number) / log(base);
// logx -> e based , log10x -> 10 based , log2x -> 2 based
```

GCD - LCM:

```
int LCM(int a, int b) // O( log (a * b) ){
    return (a / __gcd(a, b)) * b;
}
```

Divisors:

```
vector<ll> divisors(ll x) // O( sqrt (x) ){
    vector<ll> v;
    for (ll i = 1; i * i <= x; i++){
        if (x % i == 0){
            v.push_back(i);
            if (i != x / i)
                v.push_back(x / i);}
    }
    return v;
}
```

Factorization:

```
vector<ll> factorization(ll x) // O( sqrt (x) ){
    vector<ll> v;
    for (ll i = 2; i * i <= x; i++){
        while (x % i == 0){
            x /= i;
```

```
                v.push_back(i);
            }
        }
        if (x > 1)
            v.push_back(x);
        return v;
}
```

Primes:

```
bool isPrime(ll x) // O(sqrt (x)){
    // check if prime or not if the number is > 1e8
    if (x == 1)
        return 0;
    for (ll i = 2; i * i <= x; i++)
        if (x % i == 0)
            return 0;
    return 1;
}
```

Sieve:

```
const int MAX = 1e8;
bool prime[MAX];
void sieveOfEratosthenes(int n){
    for (int p = 2; p * p <= n; p++){
        if (prime[p] == true){
            for (int i = p * p; i <= n; i += p)
                prime[i] = false;
        }
    }
}
fillWith(prime, true); // In Main
```

```
sieveOfEratosthenes(n);
```

Combinations:

```
int dp[100][100];
int combination(int n, int r){ // nCr
    if(n == r || r == 0)
        return 1;
    if(dp[n][r] != -1)
        return dp[n][r];
    return dp[n][r] = combination(n - 1, r - 1) + combination(n
- 1, r);
}
fillWith(dp, -1); // In Main
```

Grapeee’s Combinations:

```
const int mod = 1e9 + 7;
int fa[100100];
int mul(int x, int y){
    return (ll) x * y % mod;
}
int po(int x, int y){
    if (!y)
        return 1;
    if (y & 1)
        return mul(x, po(x, y - 1));
    int z = po(x, y / 2);
    return mul(z, z);
}
int inv(int x){
    return po(x, mod - 2);
}
```

```
int C(int x, int y){
    if (y > x)
        return 0;
    return mul(mul(fa[x], inv(fa[y])), inv(fa[x - y]));
}
int main(){
    goFast();
    fa[0] = 1;
    for (int i = 1; i <= 1e5; i++)
        fa[i] = mul(fa[i - 1], i);
}
```

Double Precision:

```
cout.precision(10);
cout << fixed;
cout << (double) 10 << endl;
```

```
Epsilon: 1e-6
```

Counting principles:

```
F(2) = n/2 -> number of numbers in n that are divisible by 2.
F(2,3) = n/(2*3) number of numbers in n that are divisible by 2 and 3.
F(2,3,5) = n/(2*3*5) number of numbers in n that are divisible by 2 and 3 and 5.
- Number of number that are divisible by 2 or 3 or 5:
F(2) + F(3) + F(5) - F(2,3) - F(2,5) - F(3,5) + F(2,3,5)
    {we include odd subsets, exclude even subsets}
```

Round to multiple of a specified amount:

```
round(int x, int m) = (x / m) * m;      // round(48, 15) = 45
```

Divisors in a range:

How to get the number of numbers that divide m (with remainder 0) in a range from L to R:

```
if(L % m == 0)
    count = (R / m) - (L / m) + 1;
else count = (R / m) - (L / m);
```

Segment Tree:

```
int n;
int arr[100005];
int segment[4 * 100005];

void Build(int p = 1, int L = 1, int R = n){
    if (L == R){
        segment[p] = arr[L];
        return;
    }
    int Mid = (L + R) / 2;
    Build(2 * p, L, Mid);
    Build(2 * p + 1, Mid + 1, R);
    segment[p] = segment[2 * p] + segment[2 * p + 1];
}
int getQRY(int l, int r, int p = 1, int L = 1, int R = n){
    if (l > R || L > r)
        return 0; // Neutral
    if (l <= L && r >= R)
        return segment[p];
    int Mid = (L + R) / 2;
    return getQRY(l, r, 2 * p, L, Mid) + getQRY(l, r, 2 * p + 1,
Mid + 1, R);
}
void update(int pos, int newValue, int p = 1, int L = 1, int R =
n){
    if (L == R){
        segment[p] = newValue;
        return;
    }
    int Mid = (L + R) / 2;
    if (pos <= Mid)
        update(pos, newValue, p * 2, L, Mid);
    else
        update(pos, newValue, p * 2 + 1, Mid + 1, R);
    segment[p] = segment[p * 2] + segment[p * 2 + 1];
}
```

Strings:

Find function:

```
    string str = "x";
    size_t found = str.find("x");
    if(found != string::npos)
        cout << "Found at index: " << found;
```

LCS:

```
string x1, x2;
int dp[3005][3005];
int LCS(int i = 0, int j = 0){
    if(i == x1.size() || j == x2.size())
        return 0;
    if(dp[i][j] != -1)
        return dp[i][j];
    int call1 = 0, call2 = 0, call3 = 0;
    if(x1[i] == x2[j])
        return dp[i][j] = call1 = 1 + LCS(i + 1, j + 1);
    call2 = LCS(i + 1, j);
    call3 = LCS(i, j + 1);
    return dp[i][j] = max(call2, call3);
}
void path(int i = 0, int j = 0){
    if(i == x1.size() || j == x2.size())
        return;
    int call = LCS(i, j);
    if(x1[i] == x2[j]){
        cout << x1[i];
        path(i + 1, j + 1);
        return;
    }
    if(call == LCS(i + 1, j))
        path(i + 1, j);
    else
        path(i, j + 1);
}
```

Manacher’s Algorithm:

```
struct Manacher
{
    string s;
    vector<int> rad;
    int n;
    void build(string t)
    {
        s = t;
        n = 2 * s.size();
        rad.clear();
        rad.resize(n, 0);
        for (int i = 0, j = 0, k; i < n; i += k, j = max(j - k, 0))
        {
            for (; i >= j && i + j + 1 < n && s[(i - j) / 2] == s[(i + j + 1) / 2]; ++j);
            rad[i] = j;
            for (k = 1; i >= k && rad[i] >= k && rad[i - k] != rad[i] - k; ++k)
                rad[i + k] = min(rad[i - k], rad[i] - k);
        }
    }
    bool is_palindrome(int l, int r)
    {
        return l >= 0 && r < s.size() && rad[l + r] >= r - l + 1;
    }
    // Longest odd palindrome with center at ith position
    int longestOdd(int i) { return rad[i * 2]; }
    // Longest even palindrome with center between ith and (i+1)th position
    int longestEven(int i) { return rad[2 * i + 1]; }
};
Manacher man;
```

Standard Binary Search:

```
int binarySearch(){
    int L = 0, R = 1e18, Mid;
    while(L <= R){
        Mid = (L + R) / 2;
        if(can(Mid))
            R = Mid - 1;
        else
            L = Mid + 1;
    }
    if(can(Mid))
        return Mid;
    else return Mid + 1;
}
```

Standard Two Pointers:

```
int R = 0, L = 0;
while (L <= R && R != N){
    if (canTakeR)
        R++;
    else if (L == R)
        L++, R++;
    else
        L++;
}
```

Standard Bitmask:

```
int bruteForce(int i = 0, int MSK = 0){
    if(i == N){
        if(WhatEver)
            return 0;
        else return 1e9;
    }
    if(dp[i][MSK] != -1)
        return dp[i][MSK];
    int call1 = 1 + bruteForce(i + 1, MSK | (1 << i));
    int call2 = bruteForce(i + 1, MSK);
    return dp[i][MSK] = min(call1, call2);
}
```

Additional Functions:

```
 __builtin_popcount(n)
```

```
void printOnBits(ll x){
    for(int i = 0; i < 31; i++)
        if ((x & (1LL << i)) != 0)
            cout << i << " ";
}
```

```
stoi(string, 0, base);
```

```
string toBinary(int n){
    string r = "";
    while(n != 0){
        r += (n % 2 == 0 ? '0' : '1');
        n /= 2;
    }
    return r;
}
```

```
cout << M_PI << endl;
```

```
(a + b) % Mod = ((a % Mod) + (b % Mod)) % Mod
(a * b) % Mod = ((a % Mod) * (b % Mod)) % Mod
(a - b) % Mod = ((a % Mod) - (b % Mod) + Mod) % Mod
(a / b) % Mod = (a * (pow(b, Mod - 2) % Mod)) % Mod
```

```
x |= (1 << i); // Turn On
```

```
x ^= (1 << i); // Turn Off only if it was on.
```

```
test()
{
    cin >> ws;
    string x;
    getline(cin, x);
    cout << x << endl;
 }
```

Lazy Propagation Segment Tree:

```
int n;
int a[200005], seg[200005], lazy[200005];
void check(int p, int s, int e){
    if (lazy[p] != 0){
        seg[p] += lazy[p];
        if (s != e){
            lazy[2 * p] += lazy[p];
            lazy[2 * p + 1] += lazy[p];
        }
        lazy[p] = 0;
    }
}
void build(int p, int s, int e){
    check(p, s, e);
    if (s == e){
        seg[p] = a[s];
        return;
    }
    build(2 * p, s, (s + e) / 2);
    build(2 * p + 1, (s + e) / 2 + 1, e);
    seg[p] = min(seg[2 * p], seg[2 * p + 1]);
```

```
}
void update(int p, int s, int e, int i, int v){
    check(p, s, e);
    if (s == e){
        seg[p] = v;
        return;
    }
    if (i <= (s + e) / 2)
        update(2 * p, s, (s + e) / 2, i, v);
    else
        update(2 * p + 1, (s + e) / 2 + 1, e, i, v);

    seg[p] = min(seg[2 * p], seg[2 * p + 1]);
}

void update(int p, int s, int e, int a, int b, int v){
    check(p, s, e);
    if (s >= a && e <= b){
        seg[p] += v;
        if (s != e){
            lazy[2 * p] += v;
            lazy[2 * p + 1] += v;
        }
        return;
    }
    if (s > b || e < a)
        return;
    update(2 * p, s, (s + e) / 2, a, b, v);
    update(2 * p + 1, (s + e) / 2 + 1, e, a, b, v);
    seg[p] = min(seg[2 * p], seg[2 * p + 1]);
}
int get(int p, int s, int e, int a, int b){
    check(p, s, e);
    if (s >= a && e <= b)
```

```
        return seg[p];
    if (s > b || e < a)
        return 1e9;
    return min(get(2 * p, s, (s + e) / 2, a, b), get(2 * p + 1,
(s + e) / 2 + 1, e, a, b));
}
```