/* stuff you should look for:

 * 1. Runtime error -> int overflow, array bounds, assign values; int x = -1;

 * 2. special cases (n=1?)

 * 3.*** use of "long long" and "double" (if needed)

4.1 bruteforce works
4.2 presentation error

 * 4. sorting works

 * 5.Fix one value and check other

 * 6. in bitwise check for individual bits

 * 7. shift by one to the left or right

 * 8. try recursion eqn manipulate if you can

 * 9. reverse of the problem recursion

 */10. use 1e18 in binary search

11. use brackets (a*b*c)/d
12.have different approach mapped out
13. If cannot prove an approach don't code for it
14. bitwise complexity -> 32* n
15. be careful about nested loops -> don't use i instead of j
16. using variable to check is faster than sorting

1. read first paragraph of each problem... pick the easiest.. avoid hard ones.

2. Understanding the problem -> write the problem down(formalize).

3. find max allowed complexity

4. model the problem ->

minm operations -> greedy/ bfs/ dp

montonic -> binary search

possible -> 2 pointer/prefix sums

5. think simple solution first -> break the problem down bruteforce then identify why wrong

6. list approaches

7. edge cases handle -> generate all types of testcases

8. outline approach in paper

2. Understand test cases - > generate your own test cases.

6. DON'T GET STUCK ON ONE APPROACH

Figuring out the Nature of an Optimal Solution. One of the most important thing in problems imo.

Solving subtasks of the original problem and then trying to extend/generalize your solution.

Fixing a parameter and then trying to maximise the result with respect to that fixed parameter.

Finding necessary and sufficient conditions. Sometimes, your necessary conditions themselves become sufficient, or vice versa.

Identifying Lower and Upper bounds, and constructing them.

Reducing a problem into smaller subproblems, commonly used with Dynamic Programming or Inductive Proofs.

Considering the Decision Version of the Problem instead. Especially useful in Counting problems, where you need to count number of good subsequences for example.

Formalizing the Problem

```
/* double precision cout << fixed << std::setprecision(10) */

/* min heap priority_queue<int, vector<int> , greater<int>> */

/*
```

**#pragma GCC optimize("03")**

```
/*

  NO NEED TO CODE WITHOUT A COMPLETE ALGORITHM!!!

*/

  **** Observations:

*/
```

```
#include<bits/stdc++.h>
using namespace std;
```

```
void __print(int x) {cerr << x;}
void __print(long x) {cerr << x;}
void __print(long long x) {cerr << x;}
void __print(unsigned x) {cerr << x;}
void __print(unsigned long x) {cerr
<< x;}
void __print(unsigned long long x)
{cerr << x;}
void __print(float x) {cerr << x;}
void __print(double x) {cerr << x;}
void __print(long double x) {cerr <<
x;}
void __print(char x) {cerr << '\'' << x
<< '\'';}
void __print(const char *x) {cerr <<
'\"' << x << '\"';}
void __print(const string &x) {cerr <<
'\"' << x << '\"';}
void __print(bool x) {cerr << (x ?
"true" : "false");}

template<typename T, typename V>
void __print(const pair<T, V> &x)
{cerr << '{'; __print(x.first); cerr << ',';
__print(x.second); cerr << '}';}
template<typename T>
void __print(const T &x) {int f = 0;
cerr << '{'; for (auto &i: x) cerr << (f++
? "," : ""), __print(i); cerr << "}";}
void _print() {cerr << "]\n";}
template <typename T, typename...
V>
void _print(T t, V... v) {__print(t); if
(sizeof...(v)) cerr << ", "; _print(v...);}
#ifndef ONLINE_JUDGE
#define debug(x...) cerr << "[" << #x
<< "] = ["; _print(x)
#else
#define debug(x...)
#endif



//#define int long long
#define fr(i,a,b) for (int i = a; i <= b;
i++)
#define pb push_back
#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()
#define endl '\n'
#define sp " "
#define mp make_pair
#define mod 1000000007
#define inf 1e18
#define pi pair<int, int>
#define vi vector<int>
#define vecvec(type, name, n, m,
value) vector<vector<type>>
name(n, vector<type> (m, value))
```

```
#define asort(ar,n) sort(ar,ar+n)
#define vsort(v)
sort(v.begin(),v.end())
#define vrsort(v)
sort(v.rbegin(),v.rend())
#define srev(st)
reverse(st.rbegin(),st.rend())
#define YES cout<<"YES\n"
#define NO cout<<"NO\n"

template<class T>
void print_vec(const vector<T>& v,
char s=' '){
    for(size_t i=0; i<v.size(); i++){
        cout << v[i] << (i+1<v.size()? s:
'\n');
    }
}


int32_t main()
{
    ios_base:: sync_with_stdio(0);
    cin.tie(0);

}
```

```
int t, cs = 1; cin >> t;

while(t--)

{



}
```

**;VECTORS**

```
  vector<vector<int>>
arr(rows, vector<int>(cols,
0));


vector<vector<int>> v(N);


vector<int> v = {1, 2, 3};

// Insert
v.push_back(4);

// Iterate
for (int x : v) cout << x << " ";
for (int i = 0; i < v.size(); i++) cout <<
v[i] << " ";

// Sort
sort(v.begin(), v.end()); // Ascending
sort(v.rbegin(), v.rend()); //
Descending

// Find
if (find(v.begin(), v.end(), 3) !=
v.end()) cout << "Found";

auto it = find(v.begin(), v.end(), 2);
if (it != v.end())
    v.erase(it);



// Erase
v.erase(v.begin() + 1); // remove 2nd
element
```

```
v.erase(unique(v.begin(), v.end()),
v.end()); // remove duplicates from
sorted

// Lower/Upper Bound
int lb = lower_bound(v.begin(),
v.end(), 3) - v.begin();
int ub = upper_bound(v.begin(),
v.end(), 3) - v.begin();
```

**;Sets and Multisets**

```
set<int> s;
s.insert(3); s.insert(1); s.insert(2);

if (s.count(2)) cout << "Found";
s.erase(3);

for (int x : s) cout << x << " ";

auto it = s.lower_bound(2); // >= 2
auto it2 = s.upper_bound(2); // > 2

multiset<int> ms;
ms.insert(2); ms.insert(2);
ms.erase(ms.find(2)); // Only one
occurrence
```

**;Maps and Unordered Maps**

```
map<string, int> mp;
mp["apple"] = 5;
cout << mp["apple"];
```

```
for (auto [key, val] : mp) cout << key
<< " " << val << endl;

or


for (auto& p : m)
        cout << p.first << " " <<
p.second
        << "\n";

 for (auto it = m.begin(); it != m.end();
++it)
        cout << it->first << " " <<
it->second
        << endl;

erase:
  m.erase(2);



unordered_map<int, int> ump;
ump[1] = 10;
if (ump.find(1) != ump.end()) cout <<
"Exists";


 mp.clear();
```

**;STRING**

```
string s = "abcde";
reverse(s.begin(), s.end());
transform(s.begin(), s.end(),
s.begin(), ::toupper);
s.substr(1, 3); // "bcd"

string result = s1 + s2;
s1 += " world";
s1.append(" more text");
s1.append(s2, 0, 3); // first 3 chars of
s2


   string s = "geeksforgeeks";

    for (auto& x : s) {
        x = toupper(x);
    }


size_t pos = s1.find("lo");
size_t last = s1.rfind("l");

size_t firstOf = s1.find_first_of("abc");
size_t lastOf = s1.find_last_of("abc");

string sub = s1.substr(2, 3);  // from
index 2, length 3
```

```
    std::string str_num =
std::to_string(num);


string str = to_string(1234);
int n = stoi("456");
long long l = stoll("9876543210");
float f = stof("3.14");
double d = stod("2.718");

const char* cstr = str.c_str();  //
C-string

//Remove Specific Character
s1.erase(remove(s1.begin(),
s1.end(), ' '), s1.end());  // remove all
spaces


string s = "hello world";
size_t pos = s.find("world");
if (pos != string::npos)
    cout << "Found at index: " << pos;
// Outputs 6
```

**;BITS**

```
for long long
__builtin_popcountll(n)
__builtin_clzll(n);



__builtin_popcount(7);        // Count
1s in binary
__builtin_ctz(8);             // Count
trailing zeros
__builtin_clz(16);            // Count
leading zeros



// Checks if kth bit of x is set (1) or
not (0)
int check_kth_bit(int x, int k) {
  return (x >> k) & 1;
}

// Prints the positions of all set (1)
bits in binary representation of x
void print_on_bits(int x) {
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      cout << k << ' '; // prints the
position of the set bit
    }
  }
  cout << '\n';
}

// Returns the count of set (1) bits in
binary representation of x
int count_on_bits(int x) {
  int ans = 0;
  for (int k = 0; k < 32; k++) {
    if (check_kth_bit(x, k)) {
      ans++;
    }
  }
  return ans;
}

// Checks if x is even or odd
bool is_even(int x) {
  if (x & 1) {
    return false;
  }
  else {
    return true;
  }
}

// Sets the kth bit of x to 1 and
returns the result
int set_kth_bit(int x, int k) {
  return x | (1 << k);
}

// Sets the kth bit of x to 0 and
returns the result
int unset_kth_bit(int x, int k) {
  return x & (~(1 << k));
}

// Toggles the kth bit of x and returns
the result
int toggle_kth_bit(int x, int k) {
  return x ^ (1 << k);
}

// Checks if x is a power of 2
bool check_power_of_2(int x) {
  return count_on_bits(x) == 1;
}

int main() {
  // Bitwise AND (&)
  int and_result = 12 & 25;  // 12
(binary 1100) & 25 (binary 11001) = 8
(binary 1000)
  cout << "AND result: " <<
and_result << '\n'; // Output: 8

  // Bitwise OR (|)
  int or_result = 12 | 25;  // 12 (binary
1100) | 25 (binary 11001) = 29
(binary 11101)
  cout << "OR result: " << or_result
<< '\n'; // Output: 29

  // Bitwise XOR (^)
  int xor_result = 12 ^ 25;  // 12
(binary 1100) ^ 25 (binary 11001) =
21 (binary 10101)
  cout << "XOR result: " << xor_result
<< '\n'; // Output: 21

  // Bitwise NOT (~)
  int not_result = ~12;
  cout << "NOT result: " << not_result
<< '\n'; // Output: -13

  // Left shift (<<)
  int left_shift_result = 3 << 2;  // 3
(binary 11) << 2 = 12 (binary 1100)
  cout << "Left shift result: " <<
left_shift_result << '\n'; // Output: 12

  // Right shift (>>)
  int right_shift_result = 12 >> 2;  //
12 (binary 1100) >> 2 = 3 (binary 11)
  cout << "Right shift result: " <<
right_shift_result << '\n'; // Output: 3

  // Difference between 1 << x and
1LL << x
  int x = 31;
  long long res1 = 1 << x;  // This can
lead to overflow if x is large
  long long res2 = 1LL << x;  // This
avoids overflow since we're shifting
on a long long
  cout << "1 << x result: " << res1 <<
'\n';  // Output: -2147483648 (due to
overflow)
  cout << "1LL << x result: " << res2
<< "\n\n";  // Output: 2147483648
(correct value)


  x = 11; // binary representation:
1011

  cout << "Check 2nd bit of 11: " <<
check_kth_bit(x, 2) << '\n'; // Output:
0
  cout << "Set bits in 11 are at
positions: ";
  print_on_bits(x); // Output: 0 1 3
  cout << "Number of set bits in 11: "
<< count_on_bits(x) << '\n'; // Output:
3
  cout << "Is 11 even? " <<
is_even(x) << '\n'; // Output: 0 (false)
  cout << "11 after setting 2nd bit: "
<< set_kth_bit(x, 2) << '\n'; // Output:
15
  cout << "15 after unsetting 2nd bit: "
<< unset_kth_bit(15, 2) << '\n'; //
Output: 11
  cout << "11 after toggling 3rd bit: "
<< toggle_kth_bit(x, 3) << '\n'; //
Output: 3
  cout << "Is 8 a power of 2: " <<
check_power_of_2(8) << '\n'; //
Output: 1 (true)

  return 0;
}
```

**;SORT**

```
bool cmp(pair<int,int>& a,
pair<int,int>& b) {
    return a.second < b.second;
}
sort(v.begin(), v.end(), cmp);
```

**;BINARY SEARCH**

```
int n, q; cin >> n ;
  int a[n];
  for (int i = 0; i < n; i++) {
    cin >> a[i];
  }
  // a is already sorted
  int k; cin >> k;
  sort(a , a + n );
  int index = lower_bound(a , a + n ,
k) - a; // index(0 based)of first
element greater than or equal to k
  int index2 = upper_bound(a , a + n ,
k) - a; // index(0 based) of first
element greater than k

  cout << index + 1 << " " << index2 +
1 << endl;// 1 based index
```

**;Binary search on answer**

```
bool good(int w, int n, int c,
vector<int>& a)
{

}

        int l = 0;
                // try l =-1 if not ac
        int r = 2;
        while(!good(r, n, c, a)) r *= 2;

        while(r > l + 1)
        {
            int m = (l + r) / 2;
            if(good(m, n, c, a)) r = m;
            else l = m;
        }

        cout << r << '\n';
```

**;BFS**

```
const int inf = 1e9;
const int N = 1e5 + 9;
vector<int> g[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  for (int i = 1; i <= m; i++) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  queue<int> q;
  vector<int> d(n + 1, inf), par(n + 1,
-1);
  q.push(1);
  d[1] = 0;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (auto v: g[u]) {
      if (d[u] + 1 < d[v]) {
        d[v] = d[u] + 1;
        par[v] = u;
        q.push(v);
      }
    }
  }
  if (d[n] == inf) {
    cout << "IMPOSSIBLE\n";
    return 0;
  }
  cout << d[n] + 1 << '\n';
  int cur = n;
  vector<int> path;
  while (cur != -1) {
    path.push_back(cur);
    cur = par[cur];
  }
  reverse(path.begin(), path.end());
  for (auto u: path) {
    cout << u << ' ';
  }
  cout << '\n';
  return 0;
}
// https://cses.fi/problemset/task/1667
```

**;DIVISORS OF N**

```
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; cin >> n;
  vector<int> divs;
  for (int i = 1; i * i <= n; i++) {
    if (n % i == 0) {
      divs.push_back(i);
      if (i != n / i) divs.push_back(n / i);
    }
  }
  sort(divs.begin(), divs.end());
  for (auto x: divs) cout << x << ' ';
  return 0;
}
```

**;SIEVE**

```
const int N = 1e7 + 9;
bool f[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);


  int n = N - 9;
  vector<int> primes;
  f[1] = true;
  for (int i = 2; i <= n; i++) {
    if (!f[i]) {
      primes.push_back(i);
      for (int j = i + i; j <= n; j += i) {
        f[j] = true;
      }
    }
  }
  cout << primes.size() << '\n';
  return 0;
}
```

**;SPF**

```
const int N = 1e6 + 9;
int spf[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  for (int i = 2; i < N; i++) {
    spf[i] = i;
  }
  for (int i = 2; i < N; i++) {
    for (int j = i; j < N; j += i) {
      spf[j] = min(spf[j], i);
    }
  }
  int q; cin >> q; // queries q <= 1e6
  while (q--) {
    int n; cin >> n; // find prime
factorization of n <= 1e6
    vector<int> ans;
    while (n > 1) {
      ans.push_back(spf[n]);
      n /= spf[n];
    }
    for (auto x: ans) cout << x << ' ';
cout << '\n';
  }
  return 0;
}
```

**;skeleton view**

```
for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      while (n % i == 0) {
        v.push_back(i);
        n /= i;
      }
    }
  }
  if (n > 1) v.push_back(n);
  for (auto x: v) cout << x << ' ';
```

**;TWO POINTERS**

```
using ll = long long;
const int N = 1e5 + 9;
int a[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n; ll s; cin >> n >> s;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }
  ll ans = 0;
  int r = 1;
  ll sum = 0;
  for (int l = 1; l <= n; l++) {
    while (r <= n and sum + a[r] <= s) {
      sum += a[r];
      r++;
    }
    // r - 1 is the maximum index i
such that sum of a[l...i] is <= s
    ans += r - l;
    sum -= a[l];
  }
  cout << ans << '\n';
  return 0;
}
//  total number of subarrays whose
sum ≤ s.
```

**;GCD and LCM**

```
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int t; cin >> t;
  while (t--) {
    int a, b; cin >> a >> b;
    int GCD = __gcd(a, b);
    long long LCM = 1LL * a * b /
GCD;
    cout << LCM << ' ' << GCD << '\n';
  }
  return 0;
}
```

**;exclusion DP**

```
const int N = 1e6 + 5;


int32_t main()
{
    ios_base:: sync_with_stdio(0);
    cin.tie(0);
    int n; cin >> n;
    vi freq(N, 0);
    int maxm = 0;
    fr(i, 0, n - 1)
    {
        int x; cin >> x;
        freq[x]++;
        maxm = max(maxm, x);
    }

    vi cnt(maxm + 1, 0);
    fr(i, 1, maxm)
    {
        for(int j = i; j <= maxm; j += i)
        {
            cnt[i] += freq[j];
        }
    }

    vi f(maxm + 1, 0);
    vi g(maxm + 1, 0);

    fr(i, 1, maxm)
    {
        f[i] = (cnt[i] * (cnt[i] - 1) * (cnt[i] - 2)) / (3 * 2 );


         //coprime triplets for unordered (repition not allowed) cntiC3
         //repitiion allowed  (cnti + 1)C3
        //ordered pair cnti ^ 3


        //coprime triplets for unordered (repition not allowed) cntiCk
         //repitiion allowed  (cnti + 1)Ck
        //ordered pair cnti ^ k

    }

    for(int i = maxm; i >= 1; i--)
    {
        g[i] = f[i];
        for(int j = i + i; j <= maxm; j += i)
        {
            g[i] -= g[j];
        }



    }

    cout << g[1] << endl;


}
```

**;MAX SUBARRAY SUM**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int maxSubarraySum(vector<int> &arr) {

    // Stores the result (maximum sum found so far)
    int res = arr[0];

    // Maximum sum of subarray ending at current position
    int maxEnding = arr[0];

    for (int i = 1; i < arr.size(); i++) {

        // Either extend the previous subarray or start
        // new from current element
        maxEnding = max(arr[i], maxEnding + arr[i]);

        // Update result if the new subarray sum is larger
        res = max(res, maxEnding);
    }
    return res;
}

int main() {
    vector<int> arr = {2, 3, -8, 7, -1, 2, 3};
    cout << maxSubarraySum(arr);
    return 0;
}
```

**;prefix sum**

```
const int N = 2e5 + 9;
long long pref_sum[N];
int a[N];
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, q; cin >> n >> q;
  for (int i = 1; i <= n; i++) {
    cin >> a[i];
  }
  for (int i = 1; i <= n; i++) {
    pref_sum[i] = pref_sum[i - 1] + a[i];
  }
  while (q--) {
    int l, r; cin >> l >> r;
    cout << pref_sum[r] - pref_sum[l - 1] << '\n';
  }
  return 0;
}
```

**;2D prefix sum**

```
 int n; cin >>  n;
    vector<vector<int>> d(n + 1, vector<int>(n + 1, 0));
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            cin >> d[i][j];
        }
    }

    vector<vector<int>> pref_sum(n + 1, vector<int>(n + 1, 0));
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            pref_sum[i][j] = d[i][j] + pref_sum[i - 1][j] + pref_sum[i][j - 1] - pref_sum[i - 1][j - 1];
        }
    }

    vector<int> max_deliciousness(n * n + 1, 0);

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            for(int k = i; k <= n; k++)
            {
                for(int l = j; l <= n; l++)
                {
                    int sum = pref_sum[k][l] - pref_sum[i - 1][l] - pref_sum[k][j - 1] + pref_sum[i - 1][j - 1];
                    max_deliciousness[(k - i + 1) * (l - j + 1)] = max(max_deliciousness[(k - i + 1) * (l - j + 1)], sum);
                }
            }
        }
    }
```

**:GET ALL SUBSETS**

```
  for(int i = 1; i < (1 << n); i++)
    {
        vector<int> subset;
        for(int element = 0; element < n; element++)
        {
           if( i & (1 << element))
           {
               subset.push_back(element + 1);
           }
        }

}
```

;BITMASKS

```
int32_t main()
```

```
{
    ios_base:: sync_with_stdio(0);
    cin.tie(0);
    int t, cs = 1; cin >> t;
    while(t--)
    {
        int n; cin >> n;
        int k; cin >> k;
        int even = 0, cnt = 0;
        vi v(n + 1);

        for(int i = 1; i <= n; i++)
        {
            int x; cin >> x;
            v[i] = x;


        }

        fr(i, 0, 63)
        {
            int mask = 1LL << i;
            int f = 0;
            fr(j, 1, n)
            {

                if(k < mask)
                {
                    f = 1;
                    break;

                }
                if((mask & v[j]) == 0 )
                {
                    k -= mask;
                    v[j] = v[j] | mask;
                }


            }
            if(f) break;

        }

          for(int i = 1; i <= n; i++)
        {

            cnt +=
__builtin_popcountll(v[i]);


        }

        cout << cnt << endl;

    }

}
```

**;Sumofdigitsfrom1toN**

```
int sumOfDigitsFrom1ToNUtil(int n,
vector<int> &a) {
    if (n < 10)
        return (n * (n + 1) / 2);

    int d = (int)(log10(n));
    int p = (int)(ceil(pow(10, d)));
    int msd = n / p;

    return (msd * a[d] + (msd * (msd -
1) / 2) * p
            + msd * (1 + n % p) +
sumOfDigitsFrom1ToNUtil(n % p, a));
}

int sumOfDigits(int n) {
    int d = max(((int)(log10(n))), 1LL);
    vector<int> a(d + 1);
    a[0] = 0;
    a[1] = 45;

    for (int i = 2; i <= d; i++)
        a[i] = a[i - 1] * 10 + 45 *
(int)(ceil(pow(10, i - 1)));

    return
sumOfDigitsFrom1ToNUtil(n, a);
}
```

**;DFS**

```
//clear
for (int i = 1; i <= n; i++) {
        g[i].clear();
        vis[i] = false
 }

const int N = 1e5 + 9;
vector<int> g[N];
bool vis[N];
void dfs(int u) {
  vis[u] = true;
  for (auto v: g[u]) {
    if (!vis[v]) {
      dfs(v);
    }
  }
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  int n, m; cin >> n >> m;
  while (m--) {
    int u, v; cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  vector<int> components;
  for (int u = 1; u <= n; u++) {
    if (!vis[u]) {
      components.push_back(u);
      dfs(u);
    }
  }
  cout << (int)components.size() - 1
<< '\n';
  for (int i = 0; i + 1 <
(int)components.size(); i++) {
    cout << components[i] << ' ' <<
components[i + 1] << '\n';
  }
  return 0;
}
```

Given a sorted array arr (sorted in ascending order) and a target, find if there exists any pair of elements (arr[i], arr[j]) such that their sum is equal to the target.
/

```
bool twoSum(vector<int> &arr, int
target){

    int left = 0, right = arr.size() - 1;
    while (left < right){
        int sum = arr[left] + arr[right];

        if (sum == target)
            return true;

        // Move toward a higher sum
        else if (sum < target)
            left++;

        // Move toward a lower sum
        else
            right--;
    }

    // If no pair found
    return false;
}

int main(){
    vector<int> arr = {-3, -1, 0, 1, 2};
    int target = -2;
    if (twoSum(arr, target))
        cout << "true";
    else
        cout << "false";

    return 0;
}
```

;RECURSION

```
1. What is the
simplest possible
input?
2. play around with
examples and
visualize
3. Relate larger
cases to smaller
ones
4.generalize the
pattern
5. write code by
combining recursive
pattern with base
case
```

;dp

```
// C++ implementation to find the
minimum cost path
// using Space Optimization
#include <bits/stdc++.h>
using namespace std;

int minCost(vector<vector<int>>
&cost) {

    int m = cost.size();
    int n = cost[0].size();

    // 1D dp array to store the
minimum cost
                // of the current row
    vector<int> dp(n, 0);

    // Initialize the base cell
    dp[0] = cost[0][0];

    // Fill the first row
    for (int j = 1; j < n; j++) {
        dp[j] = dp[j - 1] + cost[0][j];
    }

    // Fill the rest of the rows
    for (int i = 1; i < m; i++) {

        // Store the previous value of
dp[j-1]
                // (for diagonal handling)
        int prev = dp[0];

        // Update the first column (only
depends on
                // the previous row)
        dp[0] = dp[0] + cost[i][0];

        for (int j = 1; j < n; j++) {

            // Store the current dp[j]
before updating it
            int temp = dp[j];

            // Update dp[j] using the
minimum of the
                // top, left, and diagonal
cells
            dp[j] = cost[i][j] + min({dp[j],
dp[j - 1], prev});

            // Update prev to be the old
dp[j] for the
                // diagonal calculation in
the next iteration
            prev = temp;
        }
    }

    // The last cell contains the
                // minimum cost path
    return dp[n - 1];
}

int main() {
    vector<vector<int>> cost = {{1, 2,
3}, {4, 8, 2}, {1, 5, 3}};

    cout << minCost(cost) << endl;
    return 0;
}
```

;1D

```
const int N = 1e5 + 9;
int n;
int dp[N], h[N];

int f(int i) {
    if (i == n) return 0;
    if(i == n - 1) return abs(h[i] - h[i +
1]);
    if(dp[i] != -1) return dp[i];

    int left = f(i + 1) + abs(h[i] - h[i +
1]);
    int right = f(i + 2) + abs(h[i] - h[i +
2]);
    return dp[i] = min(left, right);

}

int32_t main()
{
    ios_base:: sync_with_stdio(0);
    cin.tie(0);
    cin >> n;
    memset(dp, -1, sizeof(dp));
    for(int i = 1; i <= n; i++) {
        cin >> h[i];
    }
    cout << f(1) << endl;


}
```

;2D

```
const int N = 1e5 + 9;
int n;
int dp[N][3];
int arr[N][3];

int f(int i, int activity) {
    if(i == n + 1) return 0;
    if(dp[i][activity] != -1) return
dp[i][activity];

    int ans = 0;

    fr(k, 0, 2)
    {
        if(activity != k)
        {
            ans = max(ans, f(i + 1, k) +
arr[i][k]);
        }
    }

    return dp[i][activity] = ans;

}



int32_t main()
{
    ios_base:: sync_with_stdio(0);
    cin.tie(0);
    cin >> n;

    memset(dp, -1, sizeof(dp));
    fr(i,1,n)
    {
        fr(j,0,2)
        {
            cin >> arr[i][j];
        }
    }

    cout << f(1, -1) << endl;


}
```

;subset sum equal to n

```
//C++ implementation for subset sum
// problem using memoization
#include <bits/stdc++.h>
using namespace std;

// Recursive function to check if a
subset
// with the given sum exists
bool isSubsetSumRec(vector<int>&
arr, int n, int sum,
                    vector<vector<int>>
&memo) {

    // If the sum is zero, we found a
subset
    if (sum == 0)
        return 1;

    // If no elements are left
    if (n <= 0)
        return 0;

    // If the value is already
            // computed, return it
    if (memo[n][sum] != -1)
        return memo[n][sum];

    // If the last element is greater than
            // the sum, ignore it
    if (arr[n - 1] > sum)
        return memo[n][sum] =
isSubsetSumRec(arr, n - 1, sum,
memo);
    else {

        // Include or exclude the last
element
        return memo[n][sum] =
isSubsetSumRec(arr, n - 1, sum,
memo) ||

isSubsetSumRec(arr, n - 1, sum -
arr[n - 1], memo);
    }
}

// Function to initiate the subset sum
check
bool isSubsetSum(vector<int>&arr,
int sum) {
   int n = arr.size();

    vector<vector<int>> memo(n + 1,
vector<int>(sum + 1, -1));
    return isSubsetSumRec(arr, n,
sum, memo);
}

int main() {

    vector<int>arr = {1, 5, 3, 7, 4};
    int sum = 12;

    if (isSubsetSum(arr, sum)) {
        cout << "True" << endl;
    }
    else {
        cout << "False" << endl;
    }

    return 0;
}
```

;MOD

```
ans = ( (1LL * A )% mod) *(1LL *
B)% mod) );


//subtraction
int ans = (a - b) % mod;
if(ans < 0) ans += mod;

//divison where P is a prime mod
int divide(int a, int b)
{
return a * power(b, P - 2) % P;

}
```

;Counting

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e6, mod = 1e9 + 7;

int power(long long n, long long k) {
  int ans = 1 % mod; n %= mod; if (n
< 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n
% mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
int f[N], invf[N];
int nCr(int n, int r) {
  if (n < r or n < 0) return 0;
  return 1LL * f[n] * invf[r] % mod *
invf[n - r] % mod;
}
int nPr(int n, int r) {
  if (n < r or n < 0) return 0;
  return 1LL * f[n] * invf[n - r] % mod;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  f[0] = 1;
  for (int i = 1; i < N; i++) {
    f[i] = 1LL * i * f[i - 1] % mod;
  }
  invf[N - 1] = power(f[N - 1], mod -
2);
  for (int i = N - 2; i >= 0; i--) {
    invf[i] = 1LL * invf[i + 1] * (i + 1) %
mod;
  }
  cout << nCr(6, 2) << '\n';
  cout << nPr(6, 2) << '\n';
  return 0;
}
```

**;MISC**

```
vector<int> v;

    // Assign a vector of size 3 with
value 9
    v.assign(3, 9);
*max_element(numbers.begin(),
numbers.end())
```

- cout << setw(2) << setfill('0') << x << endl;

- cout << fixed << setprecision(6) << pi << "\n";  // prints 3.141593

- vector<vector<pair<char, int>>> arr(h + 1, vector<pair<char, int>>(w + 1, {'a', 0}));
- memset(dp, 0, sizeof(dp));

```
    tuple<int, string, double> t1(1,
"Hello", 3.14);

    // Using make_tuple
    auto t2 = make_tuple(42, "World",
2.71);

    // Accessing elements (std::get)
    cout << get<0>(t1) << "\n";  // 1
    cout << get<1>(t1) << "\n";  // Hello
    cout << get<2>(t1) << "\n";  // 3.14


sort(arr, arr + 9, greater<int>());

 cout << *max_element(v.begin(), v.end()) << endl;

 std :: string A = "SILENT";
    std :: string B = "LISTEN";

    /*Checking if B is a permutation of A*/
    if ( is_permutation ( A.begin(), A.end(), B.begin() ) )
    {
        std :: cout << "Anagrams" ;
    }



    vector<int> v = {1, 2, 3};

    // Printing all the greater permutations
    // of the current vector



    do {
        for (auto i: v) cout << i << " ";
        cout << endl;
    } while (next_permutation(v.begin(), v.end()));
```

**-countvector**

```
 cout << count(v.begin(), v.end(), 2);
```

Stack (stack)
• Operations: Push/Pop/Top (O(1)).
• Note: stack is implemented as a deque (double-ended queue).

```
#include <stack>
std::stack<int> st;
st.push(10); // Push an element - `O(1)`
st.pop(); // Pop an element - `O(1)`
st.top(); // Get the top element - `O(1)`
st.size(); // Get the number of elements - `O(1)`
st.empty(); // Check if the stack is empty - `O(1)`
st.clear(); // Remove all elements - `O(n)`
```

Queue (queue)
• Operations: Enqueue/Dequeue/Front (O(1)).
• Note: queue is implemented as a deque (double-ended queue).

```
#include <queue>
std::queue<int> q;
q.push(10); // Enqueue an element - `O(1)`
q.pop(); // Dequeue an element - `O(1)`
q.front(); // Get the front element - `O(1)`
q.size(); // Get the number of elements - `O(1)`
q.empty(); // Check if the queue is empty - `O(1)`
q.clear(); // Remove all elements - `O(n)`
```

;codeblocks
F9 -> build and run
ctrl + space -> autocomplete
F2 -> show build logs
setting in top -> compiler -? set g++ version

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    // fast input using scanf
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);  // read integers
    }

    // fast output using printf
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}

int x; long long y; double z;
scanf("%d %lld %lf", &x, &y, &z);
printf("%d %lld %.6f\n", x, y, z);
```

Geometry

- sum of interior angles of a polygon with n sides = (n - 2) * 180”
- sum of exterior angles = always 360 degrees

- regular polygon - all interior angles and sides are equal

**1. Points and Vectors (Foundation of everything)**

- **Point representation:** coordinates.
- **Vector representation:** difference between two points, .
- You can think of points and vectors as the same data structure — but points represent locations, vectors represent displacement.
- **Why needed:** All geometry calculations boil down to vector operations.

**Used in:** All problems — Draw a Square, Amr and Pins, Line Segments, etc.

---

## 2. Distance and Squared Distance

- **Formula:**

  dist2=(x2−x1)2+(y2−y1)2\text{dist}^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2
  Use squared distance to avoid floating point errors.
- Always store in in C++ if coordinates can be large (up to 10910^9 in CF).

**Used in:** Draw a Square, Amr and Pins, Line Segments, Maximal Area Quadrilateral.

---

## 3. Dot Product

- **Formula:**
  a · b=axbx+ayby a \cdot b = a_x b_x + a_y b_y
- **Uses:**
  - **Angle check:** If → 90° (orthogonal).
  - Projection lengths.
- Essential for checking **right angles** in square/rectangle problems.

**Used in:** Draw a Square, Square of Rectangles.

---

## 4. Cross Product (2D "pseudo" cross)

- **Formula (scalar in 2D):**
  a×b=axby−aybxa \times b = a_x b_y - a_y b_x
- **Meaning:**
  - **Sign** → orientation:

    → counter-clockwise (CCW),
    → clockwise (CW), → collinear.
  - **Magnitude** → twice the area of triangle formed by points.

**Used in:** Maximal Area Quadrilateral (area computation), Line Segments (intersection check).

---

## 5. Orientation Test

Using cross product:

orient(a, b, c) = cross(b-a, c-a)

- Tells if is to the left/right/on the line .
- Fundamental for intersection checks, convexity checks, and polygon algorithms.

**Used in:** Line Segments, Maximal Area Quadrilateral.

---

## 6. Triangle Inequality

- In triangles:
  AB+BC>AC AB + BC > AC for all sides.
- Used for deciding if three lengths can form a triangle.
- Extended to polygon problems to check feasibility.

**Used in:** The Third Side.

---

## 7. Polygon Area (Shoelace Formula)

- Formula for simple polygon with vertices (xi,yi)(x_i, y_i):

  Area=12∣∑i=0n−1(xiyi+1−xi+1yi)∣\text{Area} = \frac{1}{2} \left| \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \right|
- Always iterate cyclically (last vertex → first vertex).
- Works for both convex and concave polygons.

**Used in:** Maximal Area Quadrilateral.

---

## 8. Segment Intersection

- Two segments and intersect if:
  1. The orientations are different in both comparisons.
  2. Special case: collinear & overlapping segments.
- Requires both **orientation test** and **on-segment check**.

**Used in:** Line Segments.

---

## 9. Integer Geometry Practices

- **Avoid floating point** unless problem specifically needs angles.
- Compare squared lengths instead of lengths.
- When dividing in area formulas, keep numerator as integer until the end.

---

### Mapping Concepts to Problems

| Problem | Concepts Needed |
|---|---|
| Draw a Square | Distance², dot product, integer geometry |
| Square of Rectangles | Distance², orthogonality, integer geometry |
| Two Regular Polygons | Basic angle reasoning (modulus), symmetry |
| The Third Side | Triangle inequality |
| Line Segments | Distance², cross product, orientation test, segment intersection |
| Amr and Pins | Distance², ceil division (math trick) |
| Maximal Area Quadrilateral | Cross product (area), iteration over points |
| Rectangle/Shape Checks | Distance², dot product, orientation |

---

If you want, I can now make you a **compact C++ geometry template** containing:

- Point struct
- dist², dot, cross
- orientation
- on-segment
- segment intersection
- polygon area
  …so you can drop it into any Codeforces problem and just call the functions without rewriting.

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

using namespace std;



#define double int

struct Point {

    double x, y;

};
```

```
bool onSegment(Point p, Point q,
Point r) {

    if (q.x <= max(p.x, r.x) && q.x >=
min(p.x, r.x)

        && q.y <= max(p.y, r.y) && q.y
>= min(p.y, r.y))

        return true;

    return false;

}




int orientation(Point p, Point q, Point
r) {

    double val = (q.y - p.y) * (r.x - q.x)

                - (q.x - p.x) * (r.y - q.y);

    if (val == 0)



        return 0;



    return (val > 0) ? 1 : 2;



//1 r is clockwise to PQ

//2 counter clockwise

}



bool doIntersect(Point p1, Point q1,
Point p2, Point q2,

                Point& res) {

    int o1 = orientation(p1, q1, p2);

    int o2 = orientation(p1, q1, q2);

    int o3 = orientation(p2, q2, p1);

    int o4 = orientation(p2, q2, q1);



    if (o1 != o2 && o3 != o4) {



        double a1 = q1.y - p1.y;

        double b1 = p1.x - q1.x;

        double c1 = a1 * p1.x + b1 *
p1.y;



        double a2 = q2.y - p2.y;

        double b2 = p2.x - q2.x;

        double c2 = a2 * p2.x + b2 *
p2.y;



        double determinant = a1 * b2 -
a2 * b1;



        if (determinant != 0) {

            res.x

                = (c1 * b2 - c2 * b1) /
determinant;

            res.y

                = (a1 * c2 - a2 * c1) /
determinant;

            return true;

        }

    }

    if (o1 == 0 && onSegment(p1, p2,
q1))

        return true;

    if (o2 == 0 && onSegment(p1, q2,
q1))

        return true;

                if (o3 == 0 &&
onSegment(p2, p1, q2))

        return true;

    if (o4 == 0 && onSegment(p2, q1,
q2))

        return true;



    return false;

}



int main() {



    int ax, ay, bx, by;

    cin >> ax >> ay >> bx >> by;

    Point p1 = { ax, ay }, q1 = { bx, by
};



    int n; cin >> n;

    vector<Point> points(n);

    for (int i = 0; i < n; i++) {

        cin >> points[i].x >> points[i].y;

    }



    int cnt = 0;

    for(int i = 0; i < n; i++)
    {

        cnt += doIntersect(p1, q1,
points[i], points[(i + 1) % n], points[i]);

    }



    cout << cnt / 2 + 1 << endl;



}
```