Template CP:

```cpp
// #include <stdio.h>
#include <bits/stdc++.h>
using namespace std;
#define fastio                                                          \
   ios_base::sync_with_stdio(false);                                    \
   cin.tie(nullptr);                                                    \
   cout.tie(nullptr);

#define F first
#define S second
using ll = long long;
using ull = unsigned long long;
const ll LM = LONG_LONG_MAX;
const int N = 2e6 + 5, M = INT32_MAX;
// int arr[N];

int main(void) {
   // freopen("in.txt","r",stdin);
   fastio; // disable with 'printf() , scanf()'
   int t = 1;
   //cin >> t;
   while (t--) {

   }
    return 0;
}
```

Problem constraints Hints:

N	complexity	Possible Algorithms & Techniques
10^18	O(log(N))	Binary & Ternary Search / Matrix Power / Cycle Tricks / Big Simulation Steps / Values ReRank
100 000 000	O(N)	A Linear Solution - May be a greedy algorithm
40 000 000	O(N log N)	linear # calls to Binary & Ternary Search / Pre-processing & Querying / D & C
10 000	O(N2)	adhock / DP / Greedy / D & C / B & B
500	O(N3)	adhock / DP / Greedy / ..
90	O(N4)	adhock / DP / Greedy / ...
30-50		Search with pruning - branch and bound
40	O(2^N/2)	Meet in Middle
20	O(2^N)	Backtracking / Generating 2^N Subsets
11	O(N!)	Factorial / Permutations / Combination Algorithm

STD lib functions:

Standard C++ header Equivalent in previous versions <iostream> <limits> <utility> <cmath> <ctype> <algorithm> <bitset> <deque> <iterator> <list> <map> <queue> <set> <stack> <unordered_map> <unordered_set> <vector>

STLs:

- **Complexity of stack vs queue vs array :**

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)

- **Bitset** `bitset<size> name(init)`

  (note1: container member functions (note: element with index 0 is the RMB) (note2 : bistset can be initialized using string of bits "binary string"): `bitset<size> variable_name("BINARY_STRING");`

  Bit access

  operator[]
  Access bit (public member function)

  count
  Count bits set (public member function)

  size
  Return size (public member function)

  test
  Return bit value (public member function)

  any
  Test if any bit is set (public member function)

  none
  Test if no bit is set (public member function)

  all
  Test if all bits are set (public member function)

  Bit operations

  set
  Set bits (public member function)

  reset
  Reset bits (public member function)

  flip

Flip bits (public member function)

Bitset operations

to_string
Convert to string (public member function)

to_ulong
Convert to unsigned long integer (public member function)

to_ullong
Convert to unsigned long long (public member function)

- **Vector**

Iterators:

begin
Return iterator to beginning (public member function)

end
Return iterator to end (public member function)

rbegin
Return reverse iterator to reverse beginning (public member function)

rend
Return reverse iterator to reverse end (public member function)

Capacity:

size
Return size (public member function)

max_size
Return maximum no. of elements vector can ever hold (effected by machine and ram) (public member function)

resize
Change size (public member function)

capacity
Return size of allocated storage capacity (public member function)

empty
Test whether vector is empty (public member function)

reserve
Request a change in capacity (public member function)

shrink_to_fit
Shrink to fit (public member function)

Element access:

operator[]

Access element (public member function)

at
Access element (public member function)

front
Access first element (public member function)

back
Access last element (public member function)

data
Access data (public member function)

Modifiers:

assign
Assign vector content (public member function)

push_back
Add element at the end (public member function)

pop_back
Delete last element (public member function)

insert
Insert elements (public member function)

erase
Erase elements (public member function)

swap
Swap content (public member function)

clear
Clear content (public member function)

emplace
Construct and insert element (public member function)

emplace_back
Construct and insert element at the end (public member function)

- **Stack**

empty
Test whether container is empty (public member function)

size
Return size (public member function)

top
Access next element (public member function)

push
Insert element (public member function)

emplace
Construct and insert element (public member function)

pop
Remove top element (public member function)

swap
Swap contents (public member function)

Non-member function overloads

relational operators
Relational operators for stack (function)

swap (stack)
Exchange contents of stacks (public member function)

- **Queue**

empty
Test whether container is empty (public member function)

size
Return size (public member function)

front
Access next element (public member function)

back
Access last element (public member function)

push
Insert element (public member function)

emplace
Construct and insert element (public member function)

pop
Remove next element (public member function)

swap
Swap contents (public member function)

Non-member function overloads

relational operators
Relational operators for queue (function)

swap (queue)
Exchange contents of queues (public member function)

- **Deque**

  Iterators:

  begin
  Return iterator to beginning (public member function)

  end
  Return iterator to end (public member function)

  rbegin
  Return reverse iterator to reverse beginning (public member function)

  rend
  Return reverse iterator to reverse end (public member function)

  cbegin
  Return const_iterator to beginning (public member function)

  cend
  Return const_iterator to end (public member function)

  crbegin
  Return const_reverse_iterator to reverse beginning (public member function)

  crend
  Return const_reverse_iterator to reverse end (public member function)

  Capacity:

  size
  Return size (public member function)

  max_size
  Return maximum size (public member function)

  resize
  Change size (public member function)

  empty
  Test whether container is empty (public member function)

  shrink_to_fit
  Shrink to fit (public member function)

  Element access:

  operator[]
  Access element (public member function)

  at
  Access element (public member function)

  front
  Access first element (public member function)

back
Access last element (public member function)

Modifiers:

assign
Assign container content (public member function)

push_back
Add element at the end (public member function)

push_front
Insert element at beginning (public member function)

pop_back
Delete last element (public member function)

pop_front
Delete first element (public member function)

insert
Insert elements (public member function)

erase
Erase elements (public member function)

swap
Swap content (public member function)

clear
Clear content (public member function)

emplace
Construct and insert element (public member function)

emplace_front
Construct and insert element at beginning (public member function)

emplace_back
Construct and insert element at the end (public member function)

Non-member functions overloads

relational operators
Relational operators for deque (function)

swap
Exchanges the contents of two deque containers (function template)

- **Priority_Queue**

empty
Test whether container is empty (public member function)

size

Return size (public member function)

top
Access top element (public member function)

push
Insert element (public member function)

emplace
Construct and insert element (public member function)

pop
Remove top element (public member function)

swap
Swap contents (public member function)

Non-member function overloads

swap (queue)
Exchange contents of priority queues (public member function)

- **Map (multimap , unordered_map , unordered_multimap)ordered search : O(log n) unordered search: O(1)**

  Iterators:

  begin
  Return iterator to beginning (public member function)

  end
  Return iterator to end (public member function)

  rbegin
  Return reverse iterator to reverse beginning (public member function)

  rend
  Return reverse iterator to reverse end (public member function)

  cbegin
  Return const_iterator to beginning (public member function)

  cend
  Return const_iterator to end (public member function)

  crbegin
  Return const_reverse_iterator to reverse beginning (public member function)

  crend
  Return const_reverse_iterator to reverse end (public member function)

  Capacity:

  empty

Test whether container is empty (public member function)

size
Return container size (public member function)

max_size
Return maximum size (public member function)

Element access:

operator[]
Access element (public member function)

at
Access element (public member function)

Modifiers:

insert
Insert elements (public member function)

erase
Erase elements (public member function)

swap
Swap content (public member function)

clear
Clear content (public member function)

emplace
Construct and insert element (public member function)

emplace_hint
Construct and insert element with hint (public member function)

Operations:

find
Get iterator to element (public member function)

count
Count elements with a specific key (public member function)

lower_bound
Return iterator to lower bound (public member function)

upper_bound
Return iterator to upper bound (public member function)

equal_range
Get range of equal elements (public member function)

- **Set** **(multiset , unordered_set , unordered_multiset) ordered search : O(log n) unordered search: O(1)**

Iterators:

begin
Return iterator to beginning (public member function)

end
Return iterator to end (public member function)

rbegin
Return reverse iterator to reverse beginning (public member function)

rend
Return reverse iterator to reverse end (public member function)

cbegin
Return const_iterator to beginning (public member function)

cend
Return const_iterator to end (public member function)

crbegin
Return const_reverse_iterator to reverse beginning (public member function)

crend
Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty
Test whether container is empty (public member function)

size
Return container size (public member function)

max_size
Return maximum size (public member function)

Modifiers:

insert
Insert element (public member function)

erase
Erase elements (public member function)

swap
Swap content (public member function)

clear
Clear content (public member function)

emplace
Construct and insert element (public member function)

emplace_hint
Construct and insert element with hint (public member function)

Operations:

find
Get iterator to element (public member function)

count
Count elements with a specific value (public member function)

lower_bound
Return iterator to lower bound (public member function)

upper_bound
Return iterator to upper bound (public member function)

equal_range
Get range of equal elements (public member function)

Algorithm lib functions:

Non-modifying sequence operations:

find
Find value in range (function template)

count
Count appearances of value in range (function template)

Modifying sequence operations:

copy
Copy range of elements (function template)

swap
Exchange values of two objects (function template)

fill
Fill range with value (function template)

remove
Remove value from range (function template)

reverse
Reverse range (function template)

rotate
Rotate left the elements in range (function template)

shuffle
Randomly rearrange elements in range using generator (function template)

Sorting:

sort

Sort elements in range (function template)

is_sorted
Check whether range is sorted (function template)

lower_bound
Return iterator to lower bound (function template)

upper_bound
Return iterator to upper bound (function template)

includes
Test whether sorted range includes another sorted range (function template)

set_union
Union of two sorted ranges (function template)

set_intersection
Intersection of two sorted ranges (function template)

set_difference
Difference of two sorted ranges (function template)

Min/max:

min
Return the smallest (function template)

max
Return the largest (function template)

Other:

lexicographical_compare
Lexicographical less-than comparison (function template)

next_permutation
Transform range to next permutation (function template)

prev_permutation
Transform range to previous permutation (function template)

> ## *String functions and member functions:*

String mem. functions

Capacity:

size	Return length of string (public member function)
length	Return length of string (public member function)
max_size	Return maximum size of string (public member function)
resize	Resize string (public member function)
capacity	Return size of allocated storage (public member function)
reserve	Request a change in capacity (public member function)
clear	Clear string (public member function)
empty	Test if string is empty (public member function)
shrink_to_fit	Shrink to fit (public member function)

Element access:

operator[]	Get character of string (public member function)
at	Get character in string (public member function)
back	Access last character (public member function)
front	Access first character (public member function)

Modifiers:

operator+=	Append to string (public member function)
append	Append to string (public member function)
push_back	Append character to string (public member function)
assign	Assign content to string (public member function)
insert	Insert into string (public member function)
erase	Erase characters from string (public member function)
replace	Replace portion of string (public member function)
swap	Swap string values (public member function)
pop_back	Delete last character (public member function)

String operations:

c_str	Get C string equivalent (public member function)
data	Get string data (public member function)
get_allocator	Get allocator (public member function)
copy	Copy sequence of characters from string (public member function)
find	Find content in string (public member function)
rfind	Find last occurrence of content in string (public member function)
find_first_of	Find character in string (public member function)
find_last_of	Find character in string from the end (public member function)
find_first_not_of	Find absence of character in string (public member function)
find_last_not_of	Find non-matching character in string from the end (public member function)
substr	Generate substring (public member function)
compare	Compare strings (public member function)

Check type of characters

Table 3.3: cctype Functions	
`isalnum(c)`	`true` if `c` is a letter or a digit.
`isalpha(c)`	`true` if `c` is a letter.
`iscntrl(c)`	`true` if `c` is a control character.
`isdigit(c)`	`true` if `c` is a digit.
`isgraph(c)`	`true` if `c` is not a space but is printable.
`islower(c)`	`true` if `c` is a lowercase letter.
`isprint(c)`	`true` if `c` is a printable character (i.e., a space or a character that has a visible representation).
`ispunct(c)`	`true` if `c` is a punctuation character (i.e., a character that is not a control character, a digit, a letter, or a printable whitespace).
`isspace(c)`	`true` if `c` is whitespace (i.e., a space, tab, vertical tab, return, newline, or formfeed).
`isupper(c)`	`true` if `c` is an uppercase letter.
`isxdigit(c)`	`true` if `c` is a hexadecimal digit.
`tolower(c)`	If `c` is an uppercase letter, returns its lowercase equivalent; otherwise returns `c` unchanged.
`toupper(c)`	If `c` is a lowercase letter, returns its uppercase equivalent; otherwise returns `c` unchanged.

Seive code O(nLog(Log(n)))

```
int N = 1e6 + 5;
vector<bool> is_prime(N, true);

void sieve(ll sz){

    is_prime[0] = is_prime[1] = false; //not primes ( 1 is rarely  considered prime
)

    for (int i = 2; i * i <= n; i++) { // or 'i <= i/n' if 'i' could OF
        if (is_prime[i]) {
            for (int j = i * i; j <= n; j += i)
                is_prime[j] = false;
        }
    }
}
```

Check if specific number is prime (not seive)

```
bool isPrime(int n) { // O(sqrt(n))
    if (n == 2) return true;
    if (n < 2 || !(n & 1)) return false;
    for (int i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return false;
    }
```

```
    return true;
}
```

Bitmasking

```
//Even test
    if( (n%2) == 9)      // works for +ve values ONLY!
    if( (n&1) == 1)      // works for -ve an +ve values!

//print number in binary ( can also use it to count bits )
void printNumber(int n, int len)
{
    if(!len)
        return ;

    printNumber(n>>1, len-1);   // remove last bit
    cout<<(n&1);                                // print last bit
    cnt++;                                              //count bits
}

//print number in binary (method 2) -> you also can save it as binary in bitset

int val = 10;
cout << bitset<bits_no>(val);


// Get bit
int getBit(int num, int idx) {
  return ((num >> idx) & 1) == 1;               // 110100, idx = 4  -->  110 & 1 = 0
}

//Set bit
int setBit1(int num, int idx) {
    return num | (1<<idx);
}

//Reset bit
int setBit0(int num, int idx) {
    return num & ~(1<<idx);                                 // 110100, idx = 3  -->
110100 & ~000100 = 110100 & 111011
}


//Flip Bit
int flipBit(int num, int idx) {
    return num ^ (1<<idx);
}

//used in Gettig mask subsets + Counting 1 bits + Getting first 1 bit from  right
(LSB)
```

```
/*
X-1 is very important!

X       = 840   = 01101001000
X-1 = 839       = 01101000111

X & (X-1)       = 01101000000           first bit from right removed

X & ~(X-1)      = 01101001000 & 10010111000 = 00000001000       //get 1st 1 bit from
(lsb)
*/

int countNumBits2(int mask) {   //      O(bits Count)           // you also can use
'__builtin_popcount()'
    int ret = 0;
    while (mask)        {
        mask &= (mask-1);
        ++ret;  // Simply remove the last bit and so on
    }
    return ret;
}

//GET   subsets  of specific mask
//we know subsets of 101: 101, 100, 001, 000
void getAllSubMasks(int mask) {

    for(int subMask = mask ; subMask ; subMask = (subMask - 1) & mask)
        printNumber(subMask, 32);       // this code doesn't print 0

    // For reverse: ~subMask&mask = subMask^mask
}


//GET all subsets of length 'X'
// len = 3: 000, 001, 010, 011, 100, 101, 110, 111

void printAllSubsets(int len)   // Remember we did it recursively! This is much
SIMPLER!
{
    for (int i = 0; i < (1<<len); ++i)
        printNumber(i, len);

    // For reversed order. Either reverse each item or work from big to small
    //for (int i = (1<<len); i >= 0 ; --i)
    //  printNumber(i, len);
}
```

> ***nPr , nCr :***

- **in nPr (any re-order of same choosen elements increases all nPr count)**
- **nCr is alwyas less than nPr (diffrent arrangement of same choosen elements does not increase nCr counter more than once for same elements)**

Summarizing the Formulas for Counting Permutations and Combinations with and without Repetition

TABLE 1 Combinations and Permutations With and Without Repetition.

Type	*Repetition Allowed?*	*Formula*
r-permutations	No	$\frac{n!}{(n-r)!}$
r-combinations	No	$\frac{n!}{r!(n-r)!}$
r-permutations	Yes	n^r
r-combinations	Yes	$\frac{(n+r-1)!}{r!(n-1)!}$

---

next_permutation(start,end)

array must be sorted inc. to get all permutations count and arrangment (summary: to get all permutaion start from smallest arrangement in lexical order)

Inclusion Exclusion: $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$

Mod properties :

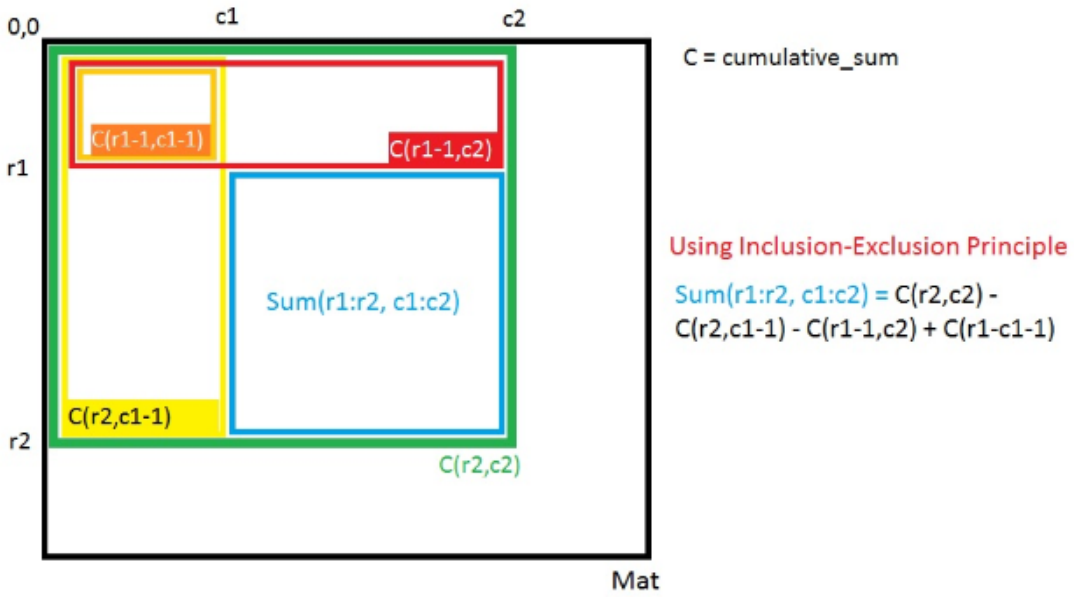
```
( a + b) % c = ( ( a % c ) + ( b % c ) ) % c
( a * b) % c = ( ( a % c ) * ( b % c ) ) % c
( a - b) % c = ( ( a % c ) - ( b % c ) ) % c
```

2D prefix sum code

```
int arr[N+1][M+1]={0}, prefixSum[N+1][M+1];
  int n,m;
  cin>>n>>m;
  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
      cin>>arr[i][j];

  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
      prefixSum[i][j]=arr[i][j] + prefixSum[i-1][j] + prefixSum[i][j-1] -
prefixSum[i-1][j-1];


  int r1,c1, r2,c2;
  cin>>r1>>c1>>r2>>c2; // one-based
  cout<<prefixSum[r2][c2] - prefixSum[r1-1][c2] - prefixSum[r2][c1-1] +
prefixSum[r1-1][c1-1]<<'\n';
```



Summation formulas

1. $$\sum_{i=1}^{n} c = cn$$

2. $$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

3. $$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

4. $$\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$$

Fast power code

```
// in iterative style
ll fastPow(ll base, ll power /*, ll m*/) { // O(log[power])
  ll ans = 1;
  while (power) {
    if (power & 1)
      ans = (ans * base); // if we have mod => ans = (ans * base) % m
    base = (base * base); // base = (base * base) % m
    power >>= 1;
  }
  return ans;
```

```
}
//in RECURSIVE style
int fast_pow(int x,int n) {
    if(n==0) return 1;
    //n is even
    else if(n%2 == 0)
        return fast_pow(x*x,n/2);
    //n is odd
    else
        return x * fast_pow(x*x,(n-1)/2);
}
```

---

sqrt(), sqrtl(), sqrtf(), log(), logf(), logl():

```
may have accuracy and approximation issues at very big or small numbers (hard to
predict when it will be off by +1 or -1   so you can handle its errors)
```

---

log for counting number of bits or digits :

```
int n = 124123123;
int no_bits = floor( log2(n) ) + 1; //  gets no of bits


int no_dig = floor( log10(n) ) + 1; // get no of digits
```

---

gcd and lcm

```
gcd <data_type> (x,y);
__gcd(x,y);
 int lcm = (x * y) / gcd(x,y);
```

Prime factorization (to get all prime factors of a number)

```
vector<int> v;
vector<int> prime_factors(int n){

    while ( (n & 1) == 0 ){//if even  note: '==' has higher precedence than '&'
        v.push_back(2);
        n >>= 1;
    }

    // n must be odd at this point. So we can skip
    // one element (Note i = i +2)
```

```
    for(int i = 3; i <= n / i; i += 2){
        // While i divides n, print i and divide n
        while (n % i == 0){
            v.push_back(i);
            n /= i;
        }
    }

    // This condition is to handle the case when n
    // is a prime number greater than 2
    if (n > 2) v.push_back(n);

    return v;
}
```

Factors of a number

```
void printDivisors(int n)
{
    // Note that this loop runs till square root
    for (int i=1; i<=sqrt(n); i++)
    {
        if (n%i == 0)
        {
            // If divisors are equal, print only one(case of n being complete
square)
            if (n/i == i)
                cout <<" "<< i;

            else // Otherwise print both
                cout << " "<< i << " " << n/i;
        }
    }
}
```

Quadratic function solver code

```
// x = (-b ± sqrt(b^2 - 4ac) ) / 2a;
float a, b, c, x1, x2, discriminant, realPart, imaginaryPart;
    cin >> a >> b >> c;
    discriminant = b*b - 4*a*c;

    if (discriminant > 0) {
        x1 = (-b + sqrt(discriminant)) / (2*a);
        x2 = (-b - sqrt(discriminant)) / (2*a);
    }
```

```
    else if (discriminant == 0) {
        x1 = -b/(2*a);
    }

    else {
        realPart = -b/(2*a);
        imaginaryPart =sqrt(-discriminant)/(2*a);
    }
```

Compare function example code

```
//lamda function (sort vector of pairs)
vector<pair<int, stack<int>>> arr(mp.begin() , mp.end());
sort(arr.begin(), arr.end(), [](const auto& a, const auto& b) { return a.first >
b.first;});

//compare function ( sort  desc. by first  if equal sort desc. by second)
bool comp (  pair<int, int> &a , pair<int,int> &b){
    if ( a.F != b.F ){
        return a.F > b.F;
    }else{
        return a.S > b.S;
    }
}
```

Binary-search on int code

polarity method : if low starts at highly possible case it ends at opposite polarity (first impossiple case) high the same ends at first possible case (high is answer!)

```


//only edit mid
int ans = 1;
        int l = 1;//re-assure from input ranges
        int h = arr[n - 1] - arr[0];

        while ( l < h ){
            int m = (l + h + 1) / 2;

            if ( h - l <= 4){
                for (int i = l; i <= h; i++)
                    if ( ok(arr , c , i) )
                        ans = i;
                break;
            }

            if ( !ok(arr , c , m) ) h = m - 1;
            else{
```

```
                ans = m;
                l = m + 1;
            }
        }
```

Binary-search on float code

```
double binarySearch(double st, double en){
   double L = 0 , R = en, mid;
   while(R - L > eps){ // eps: some very small value (dependent on the problem)
     mid = L + (R - L) / 2;
     if(valid(mid))
       L = mid;
     else
       R = mid;
   }
   return L + (R - L) / 2; // mid
}
```

Backtracing code rules

(maze and the general code template of backtracking) :

```


// Typical backtracking procedure ( base case - state - transition )
void recursion(state s)
{
   if( base(s) )
       return ;

   for each substate ss
       mark ss

       recursion (ss)

       unmark ss
}


//example : maze
char maze(MAX , MAX){
//.SX..
//..X.E
//....X
//X.XX.
}
bool vis[MAX][MAX];
bool findEnd2(int r, int c)     // Recursion State: r, c        and FULL visted
```

```
array
{
   if( !valid(r, c) || maze[r][c] == 'X' || vis[r][c] == 1)
       return false;            // invalid position or block position


   vis[r][c] = 1;       // we just visited it, don't allow any one back to it

   if( maze[r][c] =='E')
       return true;     // we found End

   // Try the 4 neighbor cells
    if(findEnd2(r, c-1)) return true;    // search up
   if(findEnd2(r, c+1)) return true;     // search down
   if(findEnd2(r-1, c)) return true;     // search left
   if(findEnd2(r+1, c)) return true;     // search right

   vis[r][c] = 0;       // undo marking, other paths allowed to use it now

   // Can't find a way for it!
   return false;
}
```

example of backtraking that i studied before: https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/

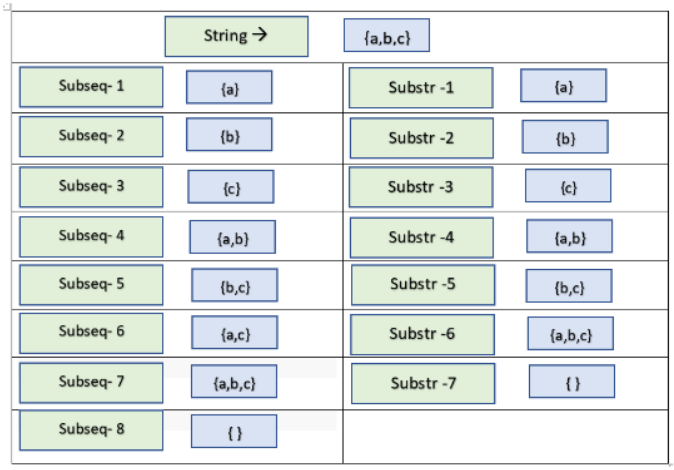
Substring vs subsequence vs subarray

Subsequence Vs Substring Vs Subarray

Factors	Subarray	Substring	Subsequence
Contiguous	Yes	Yes	No
Elements Ordered	Yes	Yes	Yes
Empty str/seq/arr	Yes	Yes	No

As we know, subarrays and substrings need to be made up of a contiguous sequence of elements of their parents, while subsequences do not have to be in a continuous sequence. In addition, all subarrays, substrings, and subsequences should preserve relative order, which means their elements should appear in the same order as they appear in their parents.

Let us look at an example below of Subsequence Vs Substring:



Diagonally navigation 2D array :

```
//traverse and print all zeroes in main diagonal
for (int step = 0 ; step < row; step++)
cout << arr[step][step];

//traverse and print all zeroes in anti diagonal
for (int step = 0 ; step < row; step++)
cout << arr[step][colm - 1 - step];
```

Dijkestra code (if i studied it)

Setprecision

`cout << fixed << setprecision(num_after_point) << float_val;`

Cout formating

Hex output `cout <<hex << number << dec;` **, Binary output** `cout << bitset<sz>number;`

printf() and scanf() specially with floating point fixed precision

```
scanf("%d", &testInteger);
printf("Number = %d", testInteger); //%f for float  %lf for double %c for char

//control floating point
//9dig before the point + the point then 6dig after the point) -> 123456789.123456
printf("%10.6lf",myDoubleNum);
```

First 1000 primes (converte to pdf to show the primes table)

	1	2	3	4	5	6	7	8	9	10
1 - 10	2	3	5	7	11	13	17	19	23	29
11 - 20	31	37	41	43	47	53	59	61	67	71
21 - 30	73	79	83	89	97	101	103	107	109	113
31 - 40	127	131	137	139	149	151	157	163	167	173
41 - 50	179	181	191	193	197	199	211	223	227	229
51 - 60	233	239	241	251	257	263	269	271	277	281
61 - 70	283	293	307	311	313	317	331	337	347	349
71 - 80	353	359	367	373	379	383	389	397	401	409
81 - 90	419	421	431	433	439	443	449	457	461	463
91 - 100	467	479	487	491	499	503	509	521	523	541
101 - 110	547	557	563	569	571	577	587	593	599	601
111 - 120	607	613	617	619	631	641	643	647	653	659
121 - 130	661	673	677	683	691	701	709	719	727	733
131 - 140	739	743	751	757	761	769	773	787	797	809

141 - 150	811	821	823	827	829	839	853	857	859	863
151 - 160	877	881	883	887	907	911	919	929	937	941
161 - 170	947	953	967	971	977	983	991	997	1009	1013
171 - 180	1019	1021	1031	1033	1039	1049	1051	1061	1063	1069
181 - 190	1087	1091	1093	1097	1103	1109	1117	1123	1129	1151
191 - 200	1153	1163	1171	1181	1187	1193	1201	1213	1217	1223
201 - 210	1229	1231	1237	1249	1259	1277	1279	1283	1289	1291
211 - 220	1297	1301	1303	1307	1319	1321	1327	1361	1367	1373
221 - 230	1381	1399	1409	1423	1427	1429	1433	1439	1447	1451
231 - 240	1453	1459	1471	1481	1483	1487	1489	1493	1499	1511
241 - 250	1523	1531	1543	1549	1553	1559	1567	1571	1579	1583
251 - 260	1597	1601	1607	1609	1613	1619	1621	1627	1637	1657
261 - 270	1663	1667	1669	1693	1697	1699	1709	1721	1723	1733
271 - 280	1741	1747	1753	1759	1777	1783	1787	1789	1801	1811
281 - 290	1823	1831	1847	1861	1867	1871	1873	1877	1879	1889
291 - 300	1901	1907	1913	1931	1933	1949	1951	1973	1979	1987
301 - 310	1993	1997	1999	2003	2011	2017	2027	2029	2039	2053
311 - 320	2063	2069	2081	2083	2087	2089	2099	2111	2113	2129
321 - 330	2131	2137	2141	2143	2153	2161	2179	2203	2207	2213
331 - 340	2221	2237	2239	2243	2251	2267	2269	2273	2281	2287
341 - 350	2293	2297	2309	2311	2333	2339	2341	2347	2351	2357
351 - 360	2371	2377	2381	2383	2389	2393	2399	2411	2417	2423
361 - 370	2437	2441	2447	2459	2467	2473	2477	2503	2521	2531
371 - 380	2539	2543	2549	2551	2557	2579	2591	2593	2609	2617
381 - 390	2621	2633	2647	2657	2659	2663	2671	2677	2683	2687
391 - 400	2689	2693	2699	2707	2711	2713	2719	2729	2731	2741
401 - 410	2749	2753	2767	2777	2789	2791	2797	2801	2803	2819
411 - 420	2833	2837	2843	2851	2857	2861	2879	2887	2897	2903
421 - 430	2909	2917	2927	2939	2953	2957	2963	2969	2971	2999
431 - 440	3001	3011	3019	3023	3037	3041	3049	3061	3067	3079

441 - 450	3083	3089	3109	3119	3121	3137	3163	3167	3169	3181
451 - 460	3187	3191	3203	3209	3217	3221	3229	3251	3253	3257
461 - 470	3259	3271	3299	3301	3307	3313	3319	3323	3329	3331
471 - 480	3343	3347	3359	3361	3371	3373	3389	3391	3407	3413
481 - 490	3433	3449	3457	3461	3463	3467	3469	3491	3499	3511
491 - 500	3517	3527	3529	3533	3539	3541	3547	3557	3559	3571
501 - 510	3581	3583	3593	3607	3613	3617	3623	3631	3637	3643
511 - 520	3659	3671	3673	3677	3691	3697	3701	3709	3719	3727
521 - 530	3733	3739	3761	3767	3769	3779	3793	3797	3803	3821
531 - 540	3823	3833	3847	3851	3853	3863	3877	3881	3889	3907
541 - 550	3911	3917	3919	3923	3929	3931	3943	3947	3967	3989
551 - 560	4001	4003	4007	4013	4019	4021	4027	4049	4051	4057
561 - 570	4073	4079	4091	4093	4099	4111	4127	4129	4133	4139
571 - 580	4153	4157	4159	4177	4201	4211	4217	4219	4229	4231
581 - 590	4241	4243	4253	4259	4261	4271	4273	4283	4289	4297
591 - 600	4327	4337	4339	4349	4357	4363	4373	4391	4397	4409
601 - 610	4421	4423	4441	4447	4451	4457	4463	4481	4483	4493
611 - 620	4507	4513	4517	4519	4523	4547	4549	4561	4567	4583
621 - 630	4591	4597	4603	4621	4637	4639	4643	4649	4651	4657
631 - 640	4663	4673	4679	4691	4703	4721	4723	4729	4733	4751
641 - 650	4759	4783	4787	4789	4793	4799	4801	4813	4817	4831
651 - 660	4861	4871	4877	4889	4903	4909	4919	4931	4933	4937
661 - 670	4943	4951	4957	4967	4969	4973	4987	4993	4999	5003
671 - 680	5009	5011	5021	5023	5039	5051	5059	5077	5081	5087
681 - 690	5099	5101	5107	5113	5119	5147	5153	5167	5171	5179
691 - 700	5189	5197	5209	5227	5231	5233	5237	5261	5273	5279
701 - 710	5281	5297	5303	5309	5323	5333	5347	5351	5381	5387
711 - 720	5393	5399	5407	5413	5417	5419	5431	5437	5441	5443
721 - 730	5449	5471	5477	5479	5483	5501	5503	5507	5519	5521
731 - 740	5527	5531	5557	5563	5569	5573	5581	5591	5623	5639

741 - 750	5641	5647	5651	5653	5657	5659	5669	5683	5689	5693
751 - 760	5701	5711	5717	5737	5741	5743	5749	5779	5783	5791
761 - 770	5801	5807	5813	5821	5827	5839	5843	5849	5851	5857
771 - 780	5861	5867	5869	5879	5881	5897	5903	5923	5927	5939
781 - 790	5953	5981	5987	6007	6011	6029	6037	6043	6047	6053
791 - 800	6067	6073	6079	6089	6091	6101	6113	6121	6131	6133
801 - 810	6143	6151	6163	6173	6197	6199	6203	6211	6217	6221
811 - 820	6229	6247	6257	6263	6269	6271	6277	6287	6299	6301
821 - 830	6311	6317	6323	6329	6337	6343	6353	6359	6361	6367
831 - 840	6373	6379	6389	6397	6421	6427	6449	6451	6469	6473
841 - 850	6481	6491	6521	6529	6547	6551	6553	6563	6569	6571
851 - 860	6577	6581	6599	6607	6619	6637	6653	6659	6661	6673
861 - 870	6679	6689	6691	6701	6703	6709	6719	6733	6737	6761
871 - 880	6763	6779	6781	6791	6793	6803	6823	6827	6829	6833
881 - 890	6841	6857	6863	6869	6871	6883	6899	6907	6911	6917
891 - 900	6947	6949	6959	6961	6967	6971	6977	6983	6991	6997
901 - 910	7001	7013	7019	7027	7039	7043	7057	7069	7079	7103
911 - 920	7109	7121	7127	7129	7151	7159	7177	7187	7193	7207
921 - 930	7211	7213	7219	7229	7237	7243	7247	7253	7283	7297
931 - 940	7307	7309	7321	7331	7333	7349	7351	7369	7393	7411
941 - 950	7417	7433	7451	7457	7459	7477	7481	7487	7489	7499
951 - 960	7507	7517	7523	7529	7537	7541	7547	7549	7559	7561
961 - 970	7573	7577	7583	7589	7591	7603	7607	7621	7639	7643
971 - 980	7649	7669	7673	7681	7687	7691	7699	7703	7717	7723
981 - 990	7727	7741	7753	7757	7759	7789	7793	7817	7823	7829
991 - 1000	7841	7853	7867	7873	7877	7879	7883	7901	7907	7919

ASCII code table:

Char	Number	Description

	0 - 31	Control characters (see below)
	32	space
!	33	exclamation mark
"	34	quotation mark
#	35	number sign
$	36	dollar sign
%	37	percent sign
&	38	ampersand
'	39	apostrophe
(	40	left parenthesis
)	41	right parenthesis
*	42	asterisk
+	43	plus sign
,	44	comma
-	45	hyphen
.	46	period
/	47	slash
0	48	digit 0
1	49	digit 1
2	50	digit 2
3	51	digit 3
4	52	digit 4
5	53	digit 5
6	54	digit 6
7	55	digit 7
8	56	digit 8
9	57	digit 9
:	58	colon
;	59	semicolon
<	60	less-than

=	61	equals-to
>	62	greater-than
?	63	question mark
@	64	at sign
A	65	uppercase A
B	66	uppercase B
C	67	uppercase C
D	68	uppercase D
E	69	uppercase E
F	70	uppercase F
G	71	uppercase G
H	72	uppercase H
I	73	uppercase I
J	74	uppercase J
K	75	uppercase K
L	76	uppercase L
M	77	uppercase M
N	78	uppercase N
O	79	uppercase O
P	80	uppercase P
Q	81	uppercase Q
R	82	uppercase R
S	83	uppercase S
T	84	uppercase T
U	85	uppercase U
V	86	uppercase V
W	87	uppercase W
X	88	uppercase X
Y	89	uppercase Y
Z	90	uppercase Z

[	91	left square bracket
\	92	backslash
]	93	right square bracket
^	94	caret
_	95	underscore
`	96	grave accent
a	97	lowercase a
b	98	lowercase b
c	99	lowercase c
d	100	lowercase d>
e	101	lowercase e>
f	102	lowercase f>
g	103	lowercase g>
h	104	lowercase h>
i	105	lowercase i>
j	106	lowercase j>
k	107	lowercase k>
l	108	lowercase l>
m	109	lowercase m>
n	110	lowercase n>
o	111	lowercase o>
p	112	lowercase p>
q	113	lowercase q>
r	114	lowercase r>
s	115	lowercase s>
t	116	lowercase t>
u	117	lowercase u>
v	118	lowercase v>
w	119	lowercase w>
x	120	lowercase x>

y	121	lowercase y>
z	122	lowercase z>
{	123	left curly brace>
|	124	vertical bar>
}	125	right curly brace>
~	126	tilde>

Binary numbers table (1-256):

No.	Binary Number
101	1100101
102	1100110
103	1100111
104	1101000
105	1101001
106	1101010
107	1101011
108	1101100
109	1101101
110	1101110
111	1101111
112	1110000
113	1110001
114	1110010
115	1110011
116	1110100
117	1110101
118	1110110
119	1110111
120	1111000

121	1111001
122	1111010
123	1111011
124	1111100
125	1111101
126	1111110
127	1111111
128	10000000
129	10000001
130	10000010
131	10000011
132	10000100
133	10000101
134	10000110
135	10000111
136	10001000
137	10001001
138	10001010
139	10001011
140	10001100
141	10001101
142	10001110
143	10001111
144	10010000
145	10010001
146	10010010
147	10010011
148	10010100
149	10010101
150	10010110

151	10010111
152	10011000
153	10011001
154	10011010
155	10011011
156	10011100
157	10011101
158	10011110
159	10011111
160	10100000
161	10100001
162	10100010
163	10100011
164	10100100
165	10100101
166	10100110
167	10100111
168	10101000
169	10101001
170	10101010
171	10101011
172	10101100
173	10101101
174	10101110
175	10101111
176	10110000
177	10110001
178	10110010
179	10110011
180	10110100

181	10110101
182	10110110
183	10110111
184	10111000
185	10111001
186	10111010
187	10111011
188	10111100
189	10111101
190	10111110
191	10111111
192	11000000
193	11000001
194	11000010
195	11000011
196	11000100
197	11000101
198	11000110
199	11000111
200	11001000
201	11001001
202	11001010
203	11001011
204	11001100
205	11001101
206	11001110
207	11001111
208	11010000
209	11010001
210	11010010

211	11010011
212	11010100
213	11010101
214	11010110
215	11010111
216	11011000
217	11011001
218	11011010
219	11011011
220	11011100
221	11011101
222	11011110
223	11011111
224	11100000
225	11100001
226	11100010
227	11100011
228	11100100
229	11100101
230	11100110
231	11100111
232	11101000
233	11101001
234	11101010
235	11101011
236	11101100
237	11101101
238	11101110
239	11101111
240	11110000

241	11110001
242	11110010
243	11110011
244	11110100
245	11110101
246	11110110
247	11110111
248	11111000
249	11111001
250	11111010
251	11111011
252	11111100
253	11111101
254	11111110
255	11111111
256	100000000

Ranges of int and double types :

```
(use `sizeof(var)` to know exact size in your machine in bytes)
```

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647 *[~10e9]*
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767

long int	8bytes	-9223372036854775808 to 9223372036854775807
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 18446744073709551615
long long int	8bytes	-(2^63) to (2^63)-1 *[~10e19]*
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	
long double	12bytes	
wchar_t	2 or 4 bytes	1 wide character
