Code Book - I

Rasel Hasan

raselhasan.cse11@gmail.com

Binary Search

==

LOWER BOUND of x : (first element >= x) - prothom je position e boshaile array ta sorted thake, existing element hoile prothom je position e ase, na hoile immediate boro tar position

UPPER BOUND of x : (first element > x) - shesh je position e boshaile array ta sorted thake

```
//code
ll arr[100005];

ll lowerbound(ll val, ll n)
{
   ll lo = 0, hi = n - 1, md, ans = n;
   while (lo <= hi) {
      md = lo + (hi - lo) / 2;
      if (val <= arr[md]) {
         ans = md;
         hi = md - 1;
      }
      else {
         lo = md + 1;
      }
   }
   return ans;
}

ll upperbound(ll val, ll n)
{
   ll lo = 0, hi = n - 1, md, ans = n;
   while (lo <= hi) {
      md = lo + (hi - lo) / 2;
      if (val < arr[md]) {
         ans = md;
         hi = md - 1;
      }
      else {
         lo = md + 1;
      }
   }
   return ans;
}
```

STL Map

==

Properties:

1. Associative container formed by pair (key, value).Each element in a map is uniquely identified by its key value.
2. Keys are always unique.
3. Sorted by key.
4. Can be accessed like array, such as mp[10], if key 10 presents in map returns value of key(10), else return 0

Constructor:

```
map<char, int> mp; //empty map creates in constant complexity
map<char, int> mp1(mp.begin(), mp.end()); //linear in size
map<char, int> mp2(mp1); //copy of mp1, linear in size
```

Assignment operator:

```
map<char, int> mp, mp1;
mp1 = mp; // linear in size of mp
```

Functions:

```
//Traversing elements of map ->
map<char, int> mp;
map<char, int>:: iterator it;
for(it = mp.begin(); it != mp.end(); it++){
    cout << it -> first << endl; // key
    cout << it -> second << endl; // value
}
mp.begin(); // complexity constant
mp.end(); // complexity constant
mp.size(); // returns number of elements in map, constant complexity
mp.clear(); //clears all the elements from map, linear complexity in size
mp.count(key); // returns true if key is present in map, else false, O(logn)
mp.empty(); //returns true if size 0, otherwise false, constant complexity
mp.erase(it); //erase single element pointed by iterator, constant complexity
mp.erase(key); //erase by key, logarithmic in size
mp.erase(it1, it2); //erase by range, linear in distance of it1 and it2
mp.find(key); //returns iterator if key is found, otherwise returns mp.end()
complexity logarithmic in size
mp.upper_bound(key); // returns upper bound iterator, complexity logn
mp.lower_bound(key); // returns lower bound iterator, complexity logn
```

STL Multimap

==

Properties:

1. Multiple elements in the container can have equivalent keys.
2. Sorted in ascending order.

Functions:
mmp.count(k); //The number of elements in the container contains that have a key equivalent to k. Complexity logarithmic in size plus number of matches.
mmp.equal_range(k); //Returns (pair of iterators, lower bound and upper bound) the bounds of a range that includes all the elements in the container which have a key equivalent to k. Complexity logarithmic in size.

STL Set

==
Properties:
1. All elements inside a set are **unique**.
2. Always remains in a **sorted** manner.
3. Any element in a set cannot be changed. It can only be inserted or deleted.
4. Implemented by BST (Binary Search Tree).

Initialization:
1. Defining the data types for the elements
2. Order of elements to be sorted

```
//empty set, increasing order (default)
set<int> s; //s = {}
//empty set, decreasing order
set<int, greater<int>> s; //s = {}
//set with values
set<int> s = {6, 3, 2, 5}; // s = {2, 3, 5, 6}
//set with from another set
set<int> s1(s);
//set from array
set<int> s(arr, arr+5); //using first index of array to fifth index
```

Iterator:
```
// Set with values
set<int, greater<int>> s1 = {6, 10, 5, 1};

// Iterator for the set
set<int> :: iterator it;

// Print the elements of the set
for(it=s1.begin(); it != s1.end();it++)
   cout<<*it<<" ";
cout<<endl;
```

Functions:
```
s.insert(x);//inserts value(x) to the set, complexity is O(logn)
s.size();//returns number of elements in the set, complexity is constant
s.erase(5);//erase value 5 from the set, complexity is O(logn)
s.erase(s.begin());//erase by iterator, complexity is almost constant
s.erase(it, s.end());//erase from iterator ‘it’ to before s.end(), here //complexity
is linear in size
s.find(val);//returns the iterator to the value if present, otherwise returns
//s.end(), complexity O(logn)
s.clear();//to clear a set, makes size zero, complexity linear in size
s.count(val);//returns 1 if val is present in set, otherwise returns 0, //complexity
O(logn)
s.empty();//returns true if the set size is zero, otherwise returns false
//constant complexity
s.lower_bound(val); //returns iterator to the first val in set, or s.end() if //all
elements are less than value
s.upper_bound(val);// returns first iterator of value greater than the actual
//value, complexity is logarithmic in size
```

STL Multiset

==

Properties:

3. Multiple elements in the container can have equivalent values.
4. Sorted in ascending order by default.

Functions:
```
ms.count(val); // Searches the container for elements equivalent to val and returns
the number of matches. Complexity logarithmic in size plus number of matches.
ms.equal_range(val); //Returns the bounds (lower bound and upper bound) of a range
that includes all the elements in the container that are equivalent to val.
Complexity logarithmic.
```

PBDS (Ordered Set)

==

Implementation Code :
https://github.com/Priyansh19077/YouTube-links/blob/main/PBDS.cpp
Tutorial :
https://www.youtube.com/watch?v=IWyIwLFucU4

Properties:

1. Provides all features of STL Set
2. Additional two features
    a. finding number of elements smaller than X in current set - order_of_key(x) in O(logn) time
    b. finding element present at X’th index in current set - find_by_order(x)

```
            in O(logn) time
```

Implementation:

```
//headers to include
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

//namespace to use
using namespace __gnu_pbds;

//typedef for type of set
//ascending ordered_set - unique elements
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set; // find_by_order, order_of_key

//ascending ordered_multiset - multiple elements
typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_multiset; // find_by_order, order_of_key

//change less to less_equal for non distinct pbds, but erase will bug

//descending ordered_set - unique elements
typedef tree<int, null_type, greater<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set; // find_by_order, order_of_key

//declaration in main function
      ordered_set st;

      //inserting elements
      st.insert(1);
      st.insert(10);
      st.insert(10); // by default, ordered set only contains unique values

      //finding kth element - zero indexed
      cout << "0th element: " << *st.find_by_order(0) << nl;
      cout << "1st element: " << *st.find_by_order(1) << nl;


      //finding number of elements smaller than X
      cout << "No. of elems smaller than 6: " << st.order_of_key(6) << endl;

      //lower bound -> Lower Bound of X = first element >= X in the set
      cout << "Lower Bound of 6: " << *st.lower_bound(6) << endl;

      //upper bound -> Upper Bound of X = first element > X in the set
      cout << "Upper Bound of 6: " << *st.upper_bound(6) << endl;
```

```
//remove elements
st.erase(1);
st.erase(11); // element that is not present is not affected
```

STL Vector

```
==========================================================
Initialization
vector<int> v; //empty vector of int data type
vector<int> v(4, 100); //vector of size 4, and val = 100(constant type)
vector<int> v1(v.begin(), v.end()); // iterating through previous vector v
vector<int> v2(v1); // initialized as a copy of v1
vector<int> a(n); //vector of size n, val 0, push_back will not work on it
for(int i = 0; i < n; i++) cin >> a[i];

vector<int> x, y;
x = y; // copy y to x, same type er howa lagbe

Functions
v.clear(); // removes all elements, makes size 0, complexity linear in size
v.erase(v.begin()+5); //erase 6th element
v.erase(v.begin(), v.begin()+3); //erase first 3 elements,
Complexity is number of elements erased + number of elements after the last element
deleted(moving).
v.push_back(val); // adds val to the end of the vector, O(1)
v.pop_back(); // removes last element, reduce size by 1, O(1)
v.back(); // returns last element on vector(reference), O(1)
v.front(); //returns first element on vector(reference), O(1)
v.empty(); //returns true if its empty, otherwise false, O(1)
v.size(); // returns size of the vector, O(1)

Special vector
vector<bool> mask;
mask.flip(); // flips values of the container, linear complexity
```

STL Deque

```
==========================================================
Properties
   1. Double ended queue, sequence container
   2. Insertion and deletion in both sides in O(1)

Constructor
deque<int> dq; // empty deque
deque<int> dq(10); // size 10, initial value zero on each index
deque<int> dq1(4, 100); // four ints with value 100
deque<int> dq2(dq1.begin(), dq1.end()); // iterating through dq1
```

```
deque<int> dq3(dq2); // copy of dq2
```

Functions

```
dq.size(); // returns number of elements in deque, O(1)
dq.empty(); // returns true if size is zero, otherwise returns false, O(1)
dq.erase(dq.begin()+5); // erase 6th element
dq.erase(dq.begin(), dq.begin()+3); //erase first three elements, O(number of
deleted elements)
dq.front(); // returns a reference to the first element of deque, O(1)
dq.back(); // returns a reference to the last element of deque, O(1)
dq[10]; // access element of index 10, O(1)
dq.push_back(val); //inserts val to back side of deque, O(1)
dq.push_front(val); // inserts val to front side of deque, O(1)
dq.pop_back(); // deletes last element from deque, O(1)
dq.pop_front(); //deletes first element from deque, O(1)
```

STL List

==

Properties:

1. push_front, push_back, pop_front, pop_back
2. insert element in constant time
3. Used doubly linked lists

Initialization

```
list<int> li; //empty list of int data type
list<int> li (5, 10); //list of size five with val 10 in each index
list<int> ls (li.begin(), li.end()); //iterating through another list
list<int> ls1 (ls); // copy of another list
```

Functions:

```
li.back(); //returns last element
li.front(); //returns first element
li.clear(); //removes all elements and makes size zero, complexity linear
li.empty(); // returns true if empty, otherwise false
li.erase(it); //removes val iterator points to
li.erase(it, it2); removes values in range, complexity number of elements erased.
li.push_back(val); //push val after last element
li.push_front(val);//push val before first element
li.pop_back(); //removes last element
li.pop_front();//removes first element
li.reverse(); //makes reverse ordered list, complexity linear in size
li.size();//returns size of list
li.swap(li1); // swaps elements of same type lists (li and li1), size may differ,
complexity constant

li.unique(); //an element is only removed from the list container if it compares
equal to the element immediately preceding it. Thus, this function is especially
useful for sorted lists.
```

```
li.sort(); //sorts elements in the list, default is ascending order
example li = {one, two, Three}
li.sort(); // output >> {Three, one, two}
```

String sorting:
For default strings, the comparison is a strict character code comparison, where all uppercase letters compare lower than all lowercase letters, putting all strings beginning by an uppercase letter before in the first sorting operation.

Using the function compare_nocase the comparison is made case insensitive.
li.sort(**compare_nocase**); // output>> {one, Three, two} sorts string in ascending order with case insensitive
Overall complexity nlogn

```
// comparison, not case sensitive
bool compare_nocase (const std::string& first, const std::string& second)
{
  unsigned int i=0;
  while ( (i<first.length()) && (i<second.length()) )
  {
    if (tolower(first[i])<tolower(second[i])) return true;
    else if (tolower(first[i])>tolower(second[i])) return false;
    ++i;
  }
  return ( first.length() < second.length() );
}
```

STL Stack

==
Properties:
1. Last in first out (LIFO)

Initialization
stack<int> stk;

Functions
```
stk.empty(); //return true if size is zero, otherwise false, O(1)
stk.size(); //returns number of elements in the stack, O(1)
stk.top(); // returns a reference to the top element of the stack, last inserted element, O(1)
stk.push(val); //inserts a new element at the top of the stack, O(1)
stk.pop(); // removes the element on top of the stack, O(1)
```

STL Queue

```
==========================================================
```

Properties:

1. First in first out (FIFO)

Initialization

```
queue <int> que;
```

Functions

```
que.empty(); //return true if size is zero, otherwise false, O(1)
que.size(); //returns number of elements in the queue, O(1)
que.front(); // returns a reference to the first inserted element in order, O(1)
que.back(); // returns a reference to the last element that was inserted in queue,
O(1)
que.push(val); //inserts a new element in the queue, O(1)
que.pop(); // removes the front element that was inserted first in order, O(1)
```

STL Priority Queue

```
==========================================================
```

Properties:

1. Creates max heap, or min heap
2. Creates a descending/ascending order of inserted elements

Initialization

```
priority_queue <int> prque; // max heap, top is maximum (1, 2, 5, 7), to is 7
priority_queue <int, vector<int>, greater<int>> prque; // min heap, top is minimum,
(5, 4, 2, 1), 1 is top element
```

priority_queue of pair

```
#define pll pair<ll, ll>
priority_queue <pll> pq; // max heap of pairs, top is maximum pair, sorted by first,
if first equal, then sorted by second
priority_queue<pll, vector<pll>, greater<pll> > pq; // min heap of pair, top is
minimum pair
```

Functions

```
prque.empty(); // O(1)
prque.pop(); // O(logn)
prque.push(val); // O(logn)
prque.size(); // O(1)
prque.top(); // O(1)
```

STL Library Functions

```
==========================================================
```

to_string(val);
converts val to string. data type of val can be all known data types like int, long, long long, float, double etc.

stoll(str);
converts str(string) to long long

STL Trigonometric functions

```
double PI = acos(-1); // 3.141592653589793238
double angle = 60.0; // angle value in degree unit and double datatype
cos_val = cos ( angle * PI / 180.0 );
sin_val = sin ( angle * PI / 180.0 );
tan_val = tan ( angle * PI / 180.0 );

cos_inverse_val = acos ( val ) * 180.0 / PI; //val in double, cos_inverse_val in degree unit, double data type
sin_inverse_val = asin ( val ) * 180.0 / PI;
tan_inverse_val = atan ( val ) * 180.0 / PI;
tan_inverse_y_by_x_val = atan2 (y,x) * 180 / PI;
```

STL Exp or logarithmic functions

```
exp(x); // returns e^x in double data type, x should be in double data type
log(x); // returns  value of natural logarithm (e_base) of x in double data type
log10(x); // returns value of base_10 logarithm of x in double data type
log2(x); // returns value of base_2 logarithm of x in double data type
```

STL Power functions

```
pow(x, y); // return data type double, returns val of (x ^ y), x and y should be in double data type
sqrt(x); // return data type double, returns val of square root of x, x should be in double data type
```

STL Algorithms

```
fill(); //complexity linear in distance between first and last
vector<ll> v(8);
fill (v.begin(), v.end(), 5);   // v : 5 5 5 5 5 5 5 5
fill (v.begin(), v.begin()+4, 5);   // v: 5 5 5 5 0 0 0 0

distance(); //returns number of elements between first and last iterator
distance(v.begin(), v.end()); // output: 8
```

```
lower_bound(); and upper_bound(); // sequence container must be in ascending sorted
manner
vector<ll>:: iterator low, up;
low = lower_bound(v.begin(), v.end(), val); // returns iterator pointing to the
lower_bound of val
up = upper_bound(v.begin(), v.end(), val); // returns iterator pointing to the
upper_bound of val
ll position_of_lower_bound = low - v.begin();
ll position_of_upper_bound = up - v.begin();
```

next_permutation(); //Rearranges the elements in the range [first,last) into the next lexicographically greater permutation.
If the function can determine the next higher permutation, it rearranges the elements as such and returns true. If that was not possible (because it is already at the largest possible permutation), it rearranges the elements according to the first permutation (sorted in ascending order) and returns false.
next_permutation(v.begin(), v.end());

reverse(); //reverses the order of the elements

```
reverse(v.begin(), v.end());
```

Vector Sorting

```
sort(a.begin(), a.end()); // sorting in ascending order
sort(a.rbegin(), a.rend()); // sorting in descending order
```

Array sorting

```
ll a[n + 5];
sort(a, a + n);          // sorting in ascending order
sort(a, a + n, greater<int>()); // sorting in descending order
```

Pair Sorting

```
vector< pair<ll, ll> > v;
sort(v.begin(), v.end()); //ascending order
```

Custom Comparator
The compare function checks if you want to return the first number before the second number. Return true (occurs nothing) if yes and false (swaps) if no.
Q.prothom ta ki thik position e ache??

```
//Returns true if x is smaller than y
bool cmp(pair<int, int> x, pair<int, int> y) {
    if (x.first != y.first)
        return x.first < y.first; //return the one with smaller first element
    else
        return x.second < y.second; //if first element is equal then return the one
with smaller second element
}
sort(v.begin(), v.end(), cmp);
```

Stable Sort (internally uses merge sort)
Some examples of stable sorting algorithms are Merge Sort, Insertion Sort, Bubble Sort, and Binary Tree Sort. While QuickSort, Heap Sort, and Selection sort are the

```
unstable sorting algorithm.
A sorting algorithm is stable if whenever there are two records R and S with the
same key and with R appearing before S in the original list, R will appear before S
in the sorted list.
```
```
stable_sort(arr, arr + n); // default ascending order
stable_sort(arr, arr + n, greater<int>()); // descending order
```

Unique (removes duplicates (side by side))

```
vector<ll> v; // such as v: 2 2 1 1 3 4 4 4 5
sort(v.begin(), v.end();         // v: 1 1 2 2 3 4 4 4 5
it = unique(v.begin(), v.end()); // v: 1 2 3 4 5 , , , ,
v.resize(distance(v.begin(), it)); // v: 1 2 3 4 5

// unique in string, for removing duplicates
string s;
sort(s.begin(), s.end());
s.erase(unique(s.begin(), s.end()), s.end());
// or
s.resize(distance(s.begin(), unique(s.begin(), s.end())));
```

STL Bitset BitMask

```
==========================================================
```
Memory taken by bitset :

```
approximate sizeof(bitset<N>) as:
If internal representation is 32bit (like unsigned on 32bit systems)
as 4 * ((N + 31) / 32) byte
If internal representation is 64bit (like unsigned long on 64bit systems)
as 8 * ((N + 63) / 64) byte
```

Errichto Blog
https://codeforces.com/blog/entry/73490
Constructor

```
bitset<10> bs; //0000000000
bitset<10> bs1(20); //0000010100
bitset<10> bs2(string("01010101")); //0001010101
```

Check nth bit

```
bitset<10> bs1(20); //0000010100
cout << bs1[2]; // 1         ^
```

Functions

```
bs.count(); // returns number of ones
bs.size(); // returns number of bits
bs.test(pos); //returns 1(true) if bit at pos is set, else returns 0(false)
bs.any(); //true if any of the bits is set (to one), and false otherwise
bs.none(); //true if none of the bits is set (to one), and false otherwise
bs.all(); //true if all of the bits are set (to one), and false otherwise
bs.set(); // sets 1 to all positions of bitset
bs.set(2, 0); // sets 0 at position 2
bs.set(3); //set 1 at pos 3
bs.reset(); //reset 0 to all positions of bitset
bs.reset(2); // reset 0 at position 2
bs.flip(); //flip bits
bs.flip(pos); //flip at pos
bs.to_string(); // returns string of bitset
bs.to_ullong(); // returns unsigned long long val of bitset
```

Operators

```
  bitset<4> foo (string("1001"));
  bitset<4> bar (string("0011"));
```

Bitwise and, or, xor

Formula:

- ★ $a + b = (a \mid b) + (a\&b)$
- ★ $a + b = (a \oplus b) + 2 \times (a\&b)$

```
  cout << (foo^=bar) << '\n';       // 1010 (XOR(if different bit then true,   else
false), assign)
  cout << (foo&=bar) << '\n';       // 0010 (AND, assign)
  cout << (foo|=bar) << '\n';       // 0011 (OR, assign)

  cout << (foo<<=2) << '\n';        // 1100 (SHL,assign)
  cout << (foo>>=1) << '\n';        // 0110 (SHR,assign)

  cout << (~bar) << '\n';           // 1100 (NOT, flip bit)
  cout << (bar<<1) << '\n';         // 0110 (SHL)
  cout << (bar>>1) << '\n';         // 0001 (SHR)

  cout << (foo==bar) << '\n';       // false (0110==0011)
  cout << (foo!=bar) << '\n';       // true  (0110!=0011)

  cout << (foo&bar) << '\n';        // 0010
  cout << (foo|bar) << '\n';        // 0111
  cout << (foo^bar) << '\n';        // 0101
```

Binary representation of x

```
ll x = 12345;
cout << bitset<60>(x) << endl;
/// or...
for(ll i = 59; i >= 0; i--)if(x &(1LL << i))cout<< 1; else cout << 0;
cout << endl;
/// or...
for(ll i = 59; i >= 0; i--) if((x >> i) & 1)cout << 1; else cout<< 0;
cout << endl;
```

__builtin_popcount(x); // returns popcount of a number — the number of ones in the binary representation of x. **__builtin_popcountll(x);** for long longs.

__builtin_clz(x); // returns number of leading zeroes in binary representation of x.
__builtin_ctz(x); // returns number of trailing zeroes in binary representation of x.
_builtin_parity(x); // returns true if x have odd parity, else returns false.

STL String

```
==========================================================
// print a string of length n of char ch
cout << string(5, ‘4’) << endl; // 44444
```

to_string(val);
converts val to string. data type of val can be all known data types like int, long, long long, float, double etc.

stoll(str); // if conversion not possible, throughs exception
converts str(string) to long long

sscanf(); //extract word/number/char from a c-string(character array)

```
char str[] = "10 20 hello world x";
char ch, word1[10], word2[10];
ll num1, num2;
sscanf(str, "%lld %lld %s %s %c", &num1, &num2, word1, word2, &ch);
cout << num1 << endl << num2 << endl << word1 << endl << word2 << endl << ch << endl;
```

stringstream; // extracts word/number/char from a string(STL string)

```
string s = “register 2005 300”;
stringstream ss;
ll id, interval;
while(getline(cin, s) and s.size() > 1) {
    ss.clear();
    ss << s;
    ss >> s >> id >> interval;
    cout << s << endl << id << endl << interval << endl;
}
```

Sieve of Eratosthenes

```
==========================================================
PDF // Generating primes by Jane Alam Jan
vector<ll> prime;
```

```
char sieve[100005];
void primegen(ll n){
    sieve[0] = sieve[1] = 1;
    for(int i = 4; i <= n; i += 2) sieve[i] = 1;
    ll sqrtn = sqrt(n);
    for(int i = 3; i <= sqrtn; i += 2){
        if(sieve[i] == 0){
            for(int j = i * i; j <= n; j += i + i) sieve[j] = 1;
        }
    }
    for(int i = 0; i <= n; i++) {
        if(sieve[i] == 0) prime.push_back(i);
    }
}
```

Segmented Sieve

```
==========================================================
ll segmentedsieve(ll a, ll b){
    if(a == 1) a++;
    ll sqrtn = sqrt(b);
    primegen(sqrtn);
    memset(sieve, 0, sizeof sieve);
    for(ll i = 0; i < prime.size() and prime[i] <= sqrtn; i++){
        ll p = prime[i];
        ll j = p * p;
        if(j < a) j = ((a + (p-1)) / p) * p;
        for( ; j <= b; j += p) sieve[j-a] = 1;
    }
    ll res = 0;
    for(int i = a; i <= b; i++){
        if(sieve[i-a] == 0) res++;
    }
    return res;
}
```

Bitwise Sieve 64bit

```
==========================================================
#define ll long long
#define mx 100000000
#define one (1LL)
vector<ll> prime;
ll status[(mx / 64) + 2];

bool checkprime(ll n) {
    if(status[n / 64] & (one << (n % 64))) return false;
    return true;
}

void setbit(ll n) {
    status[n / 64] = status[n / 64] | (one << (n % 64));
}

void mkprime(ll n) {
    setbit(0);
    setbit(1);

    prime.push_back(2);
    ll sqrtn = sqrt(n);
    for(ll i = 3; i <= sqrtn; i += 2) {
        if(checkprime(i)) {
            for(ll j = i * i; j <= n; j += i + i) {
                setbit(j);
            }
        }
    }

    for(ll i = 3; i <= n; i += 2) {
        if(checkprime(i)) prime.push_back(i);
    }
}
```

Prime factorization

```
===========================================================

Number of primes upto 10^n :  How many primes are there?

vector<ll> factors;
void factorize(ll n){
    int sqrtn = sqrt(n);
    for(int i = 0; i < prime.size() and prime[i] <= sqrtn; i++){
        if(n % prime[i] == 0){
            while(n % prime[i] == 0){
                n /= prime[i];
                factors.push_back(prime[i]);
            }
            sqrtn = sqrt(n);
        }
    }
    if(n != 1) factors.push_back(n);
}
```

Number of Divisors O(sqrt(n))

```
===========================================================
ll nod(ll n){
    ll sqrtn = sqrt(n), cnt = 0;
    for(int i = 1; i < sqrtn; i++){
        if(n % i == 0) cnt += 2; /// found a divisor pair {a, b}
    }

    /// need to check if sqrtn divides n
    if(n % sqrtn == 0){
        if(sqrtn * sqrtn == n) cnt++; /// if n is a perfect square
        else cnt += 2; /// not perfect square
    }
    return cnt;
}
```

Number of Divisors using prime factors x^a * y^b = (a+1) * (b+1)

```
==========================================================
ll nod(ll n){
    ll sqrtn = sqrt(n), cnt = 1;
    for(int i = 0; i < prime.size() and prime[i] <= sqrtn; i++){
        if(n % prime[i] == 0){
            ll p = 0; /// counter for power of prime
            while(n % prime[i] == 0){
                p++;
                n /= prime[i];
            }
            sqrtn = sqrt(n);
            p++; /// increasing p by one because of p^0
            cnt *= p; /// multiply with cnt
        }
    }
    if(n != 1){
        cnt *= 2; /// Remaining prime has power of p^1, so cnt need to multiply  //
by 2
    }
    return cnt;
}
```

High Composite Number (HCN)

```
==========================================================
Property :
1. The prime factorization of HCN will contain the first K consecutive primes.
2. Power of prime factors of HCN will form a non-increasing sequence.

Highly Composite Numbers are product of Primorials (Primorials are same as Factorials,
```

but instead of natural number we multiply primes.
P3 = 2×3×5 and p5 = 2×3×5×7×11).

Problem: Given an integer N, find the largest HCN which is smaller than or equal to N.

```
ll n, resnum, resdiv;
ll prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
void recur(ll pos, ll limit, ll num, ll div){
    if(div > resdiv){ // get the number with highest nod
        resnum = num;
        resdiv = div;
    }
    else if(div == resdiv and num < resnum){ // in case of tie, take smaller number
        resnum = num;
    }
    if(pos == 9) return; // end of prime list
    ll p = prime[pos];
    for(int i = 1; i <= limit; i++){
        if(num * p > n) break;
        recur(pos + 1, i, num * p, div * (i + 1));
        p *= prime[pos];
    }
}
int main(){
    n = 50;
    resnum = 0;
    resdiv = 0;
    recur(0, 30, 1, 1);
    printf("%lld %lld\n", resnum, resdiv);
}
```

Upper bound of Number of Divisors :
For programming contest, we could memorize values of HCN that comes frequently. Mainly 1344 for N ≤ 10^9 and 103,680 for N ≤ 10^18.

Sum of NOD

==
https://forthright48.com/divisor-summatory-function/
Problem : Find the snod of n = 100000 ?
nod(1) + nod(2) + nod(3) + … + nod(n) = ??

```
//code of O(n) solution
int snod( int n ) {
    int res = 0;
    for ( int i = 1; i <= n; i++ ) {
        res += n / i;
    }
    return res;
}

//code of O(sqrt(n)) solution
int snod( int n ) {
    int res = 0;
    int u = sqrt(n);
    for ( int i = 1; i <= u; i++ ) {
        res += ( n / i ) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
    return res;
}
```

Sum of Divisors (SOD)

===

Problem : Find the sod of n?

sod(12) = 1 + 2 + 3 + 4 + 6 + 12 = 28;

if $N = p_1^{a_1} \times p_2^{a_2} \times \ldots p_k^{a_k}$, then

$$SOD(N) = (p_1^0 + p_1^1 + p_1^2 \ldots p_1^{a_1}) \times (p_2^0 + p_2^1 + p_2^2 \ldots p_2^{a_2}) \times \ldots (p_k^0 + p_k^1 + p_k^2 \ldots p_k^{a_k})$$

```
//code of O(sqrt(n)) solution
int sod(int n){
    int sqrtn = sqrt(n);
    int res = 0;
    for( int i = 1; i <= sqrtn; i++){
        if(n % i == 0){
            res += i; //"i" is a divisor
            res += n / i; //"n/i" is also a divisor
        }
    }
    if(sqrtn * sqrtn == n) res -= sqrtn; //same number counted two times on first loop
    return res;
}
//////////////////////////////////////////////////////

//code of sod using prime factorization
int sod(int n){
    int res = 1;
    int sqrtn = sqrt(n);
    for(int i = 0; i < prime.size() && prime[i] <= sqrtn; i++){
        if(n % prime[i] == 0){
            int tempSum = 1; // Contains value of (p^0+p^1+...p^a)
            int p = 1;
            while(n % prime[i] == 0){
                n /= prime[i];
                p *= prime[i];
                tempSum += p;
            }
            sqrtn = sqrt(n);
            res *= tempSum;
        }
    }
    if(n != 1){
        res *= (n + 1); // Need to multiply (p^0+p^1)
    }
    return res;
}
```

LCM SUM Formula

===

$$\therefore SUM = \frac{n}{2}\left(\sum_{d|n}(\phi(d) \times d) + 1\right)$$

SUM = lcm(1, n) + lcm(2, n) + lcm(3, n) + ... + lcm(n, n)

pseudocode for lcmsum(n) >>

```
int lcm_sum_function(int n):
    sum = 0
    for d in divisor_of[n]:
        sum += phi[d] * d

    return (n * (sum + 1)) / 2;
```

lcm sum function for sum[] array >>

```
void lcm_sum() {
    for (int i = 1; i < mx; i++) {
        for (int j = i; j < mx; j += i) { //just like finding all divisors
            sum[j] += i * phi[i]; //d * phi[d] for a divisor d in divisor_of[n]
         }
        sum[i]++; //+1 in the formula
        sum[i] /= 2;//(n / 2) in the formula
        sum[i] *= i;//divide first to avoid overflow
    }
}
```

Combinatorics

===

Formula :

```
   01.   nPr(n,r) = n! / (n - r)!
   02.   nCr(n,r) = n! / (r! * (n-r)!)
   03.   nCr(n,r) = nCr(n-1, r-1) + nCr(n-1, r)

//nCr for all n and r : recursive way using memorization - O(n^2)
ll ncrdp[5005][5005]; // for n== 0 or r == 0, handle on your code yourself
ll ncr(ll n, ll r) {
      if(n == r) return ncrdp[n][r] = 1;
      else if(r == 1) return ncrdp[n][r] = n;
      else if(ncrdp[n][r]) return ncrdp[n][r];
      else return ncrdp[n][r] = ((ncr(n-1, r-1) % mod) + (ncr(n-1, r) % mod)) % mod;
}

//nCr for only one n and r - bigmod method using modular inverse
//idea - precalculate factorials (p! % mod), for all p from 1 to n, then use bigmod
function to calculate ncr
nCr=n!(r!(n−r)!)^(−1)
k^(p−1)≡1 (mod p) if gcd(k,p)=1 and p is prime (Fermat's little theorem)
k^(p−2)≡k^(−1) (mod p)
nCr=(n!(r!(n−r)!)^(p−2)) (mod p)
Time complexity =O(n+logp) // n for precalculation and logp for bigmod
Watch out when p≤n , the answer might be zero in case.
```

Inclusion Exclusion Principle

```
==========================================================
n(a) union n(b) = n(a) + n(b) - n(a intersection b) //for 2 elements in a set
Problem: How many numbers are divisible by {2, 3, 5, 6, 7, 11, 13, 15, 17} from 1 to
1000. // here m = 9 elements, n = 1000
ll m, n, arr[maxx];
//array for storing m elements
ll inexfun() {
    cin >> m;
    for(ll i = 0; i < m; i++) cin >> arr[i];
    cin >> n;
    ll res = 0;
    for(ll i = 1; i < (1 << m); i++) {
        ll p = 1;
        if(__builtin_popcount(i) >= 10) continue;
        // product of first 10 primes > 1e9
        for(ll j = 0; j < m; j++) {
            if(i & (1 << j)) p *= arr[j];
            //if elements are not prime in array, p = lcm(p, arr[j]);
        }
        if(__builtin_popcount(i) & 1) res += n / p;
        else res -= n / p;
    }
    return res;
}
```

Mathematics (Formulas)

```
==========================================================
```
01. Sum of consecutive numbers from a to b

```
Ex : 3 4 5 6 7
sum = (a+b) * ((b-a+1)/2) + ((b-a+1)%2?(a+b)/2:0);
```

02. Arithmetic Series (5 + 10 + 15 + 20 + 25)

```
if first number is a, last number is p, number of elements is n,
sum = (n * ( a + p)) / 2; //be careful of overflow
or sum = (n * (2 * a + (n - 1) * d)) / 2;
n’th element = a + (n - 1) * d; // common difference is d
```

CPP Shortcuts

```
==========================================================
```

if else condition ::

```
Ex : cout << (x > y ? “YES” : “NO”) << endl;
Meaning : if(x > y) cout << “YES\n”;
            else cout << “NO\n”;
```

memset and stl fill ::

```
ll arr[n]; memset(arr, 0, sizeof arr); //1d array
ll arr[n][m]; memset(arr, 0, sizeof arr); // 2d array
ll arr[n][m]; memset(arr, 0, sizeof(arr[0][0]) * m * n);
// where n = height, m = width

// memset has limitations, better use stl fill
ll arr[n][m];
for(ll i = 0; i < n; i++) fill(arr[i], arr[i] + m, 0);
// puts 0 value at every arr[i][j];
```

Floating Point Numbers

```
==========================================================
‘long double’ takes 80bits or 10bytes of memory space.
use ‘long double’ for all floating number variable declaration.
‘double’ takes 64bits or 8 bytes.

long double x;
long double a, b;
long double eps = 1e-9; // a very small number as comparison helper
if( a + eps < b ); // check (a < b)
if( a > b + eps ); // check (a > b)
if( a > eps ); // check (a > 0)
if( a < -eps ); // check (a < 0)
```

ASCII Table

```
===========================================================
Digit Character       ( 0 -------- 9 )
Ascii decimal value   ( 48 ------ 57 )

Upper Case letter     ( A -------- Z )
Ascii decimal value   ( 65 ------ 90 )

Lower Case letter     ( a -------- z )
Ascii decimal value   ( 97 ----- 122 )
```

Digits of Factorial

```
===========================================================
Number of digits of factorial in any base -
```

$$Any\ base\ conversion\ formula:\ log_b x\ = \frac{log_c x}{log_c b}$$

```
ll digitfac(ll n, ll base){
    double x = 0;
    for (ll i = 1; i <= n; i++){
        x += log10(i) / log10(base); // Base Conversion
    }
    ll res = x + 1 + eps; // eps means a very small number such as 10^(-9)
    return res;
}
```

Trailing zeros of a number/factorial

```
===========================================================
Number of trailing zeroes of an integer number in any base -
```

Link : Trailing Number of Zeros

```
// Returns highest power of p in n factorial.

int highestPowInFact(int n, int p)   {
    int cnt = 0;
    while(n /= p, n) cnt += n;
    return cnt;
}
```

GCD | Euclidean Algorithm | Greatest Common Divisor

```
==========================================================
It finds the value of GCD(a,b).
```

Recursive version:

```
int gcd ( int a, int b ) {
    if ( b == 0 ) return a;
    return gcd ( b, a % b );
}
```

Iterative version:

```
int gcd ( int a, int b ) {
    while ( b ) {
        a = a % b;
        swap ( a, b );
    }
    return a;
}
```

BuiltIn CPP version:

```
__gcd(a, b); // to find gcd of a and b
```

Coding Pitfall:

01. Return abs() value for handling gcd of negative value.
02. Gcd of (0, 0) should be infinite, it can be the reason for RTE.

Extended Euclidean Algorithm

```
==========================================================
```

1. It finds the value of GCD(a,b).
2. It finds two integers x and y such that, ax+by=gcd(a,b).
3. It's always possible to find such x and y that makes a representation where ax+by = gcd(a, b);

```
//code of extended euclidean algo...
int ext_gcd ( int A, int B, int *X, int *Y ){
    int x2, y2, x1, y1, x, y, r2, r1, q, r;
    x2 = 1; y2 = 0;
    x1 = 0; y1 = 1;
    for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 = x1, y2 = y1, x1 = x, y1 = y ){
        q = r2 / r1;
        r = r2 % r1;
        x = x2 - (q * x1);
        y = y2 - (q * y1);
    }
    *X = x2; *Y = y2;
    return r2;
}
```

Linear Diophantine Equation

```
==========================================================
Ax + By = C, it finds integer values of x and y for this eqn.
// Code
bool lineardiophantine(int A, int B, int C, int *x, int *y){
    int g = gcd(A, B);
    if (C % g != 0) return false; //No Solution

    int a = A / g, b = B / g, c = C / g;
    ext_gcd(a, b, x, y); //Solve ax + by = 1

    if (g < 0){ //Make Sure gcd(a,b) = 1
        a *= -1; b *= -1; c *= -1;
    }

    *x *= c; *y *= c; //ax + by = c
    return true; //Solution Exists
}
//for running main function
int x, y, A = 2, B = 3, C = 5;
    bool res = linearDiophantine(A, B, C, &x, &y);

    if (res == false) printf("No Solution\n");
    else printf ("One Possible Solution (%d %d)\n", x, y);
```

Euler Totient Function (phi of n)

```
==========================================================
```

$$\begin{aligned}
\varphi(n) &= \varphi(p_1^{k_1})\,\varphi(p_2^{k_2})\cdots\varphi(p_r^{k_r}) \\
&= p_1^{k_1-1}(p_1-1)\,p_2^{k_2-1}(p_2-1)\cdots p_r^{k_r-1}(p_r-1) \\
&= p_1^{k_1}\left(1-\tfrac{1}{p_1}\right)p_2^{k_2}\left(1-\tfrac{1}{p_2}\right)\cdots p_r^{k_r}\left(1-\tfrac{1}{p_r}\right) \\
&= p_1^{k_1}p_2^{k_2}\cdots p_r^{k_r}\left(1-\tfrac{1}{p_1}\right)\left(1-\tfrac{1}{p_2}\right)\cdots\left(1-\tfrac{1}{p_r}\right) \\
&= n\left(1-\tfrac{1}{p_1}\right)\left(1-\tfrac{1}{p_2}\right)\cdots\left(1-\tfrac{1}{p_r}\right).
\end{aligned}$$

```
Number of coprime numbers less than or equal to n
Complexity same as prime factorization

ll eulerphi(ll n){
    if(!chk[n]) return n - 1;
    ll res = n;
    ll sqrtn = sqrt(n);
    for(ll i = 0; i < prime.size() and prime[i] <= sqrtn; i++){
        if(n % prime[i] == 0){
            while(n % prime[i] == 0) n /= prime[i];
            sqrtn = sqrt(n);
            res /= prime[i];
            res *= prime[i] - 1;
        }
    }
    if(n != 1) res /= n, res *= n - 1;
    return res;
}
```

Calculates phi till mx just like Sieve of Eratosthenes

```
int phi[mx];
bitset <mx> mark;
void sieve_phi() {
    for (int i = 1; i < mx; i++) phi[i] = i;
    mark[1] = 1;
    for (int i = 2; i < mx; i++) {
        if (mark[i]) continue;
        for (int j = i; j < mx; j += i) {
            mark[j] = 1;
            phi[j] = phi[j] / i * (i - 1);
        }
    }
}
```

BigMod

==

Divide and Conquer Method(Recursive):

```
ll bigmod( ll b, ll p, ll m ){
    // Base Case
    if(p == 0) return 1 % m;
    // p is even
    if(p % 2 == 0){
        ll y = bigmod(b, p / 2, m);
        return (y * y) % m; // b^p = y * y
    }
    // b^p = b * b^(p-1)
    else {
        return (b * bigmod(b, p - 1, m)) % m;
    }
}
```

Repeated Squaring Method (RSM):

```
ll bigmod(ll b, ll p, ll m){
    ll res = 1 % m, x = b % m;
    while(p){
        if(p & 1) res = (res * x) % m;
        x = (x * x) % m;
        p >>= 1;
    }
    return res;
}
```

Sparse Table Build and range minimum query

```
==========================================================
int st[20][mx];
int arr[mx];

void compute_st(ll n) {
    for(ll i = 0; i < n; i++) st[0][i] = i;
    for(ll k = 1; (1 << k) < n; k++) {
        for(ll i = 0; i + (1 << k) <= n; i++) {
            int x = st[k-1][i];
            int y = st[k-1][i + (1 << (k-1))];
            st[k][i] = arr[x] <= arr[y] ? x : y;
        }
    }
}

ll rmq(ll l, ll r) {
    ll k = log2(r - l);
    ll x = st[k][l];
    ll y = st[k][r - (1 << k) + 1];

    return arr[x] <= arr[y] ? x : y;
}
```

TopSort

```
==========================================================
int st[20][mx];int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}
```

```
void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}
```

DSU Disjoint Set Union (Find, Connect)

```
===============================================================
vector<int> parent(100005);

int findparent(int u) {
    if(u == parent[u]) return u;
    return parent[u] = findparent(parent[u]);
}

void connect(int u, int v) {
    u = findparent(u);
    v = findparent(v);

    if(u != v) parent[u] = v;
}
```

Maximum Subarray Sum (Kadane's Algo)

```
===============================================================
//General version >>
vector<int> a(n);
int ans = a[0], sum = 0;
for (int r = 0; r < n; ++r) {
    sum += a[r];
    ans = max(ans, sum);
    sum = max(sum, 0);
}
cout << sum << nl;

//With index of the subarray of max-sum(left-right)
vector<int> a(n);
int ans = a[0], ans_l = 0, ans_r = 0;
int sum = 0, minus_pos = -1;

for (int r = 0; r < n; ++r) {
    sum += a[r];
    if (sum > ans) {
        ans = sum;
        ans_l = minus_pos + 1;
        ans_r = r;
    }
    if (sum < 0) {
        sum = 0;
        minus_pos = r;
    }
}
```