Binary search

```
# binary search
def bs(arr, x):
       l, h = 0, len(arr) - 1
        while l <= h:
                    mid = l +(h - l) // 2
                    if arr[mid] < x:
                              l = mid + 1
                    elif arr[mid] > x:
                              h = mid -1
                     else:
                                 return True
          return False
```

Complex numbers

```
typedef long long C;
typedef comple x<C> p;
#define X real()
#define Y imag()
P p = {4,2};
cout << p.X << " " << p.Y << " \n"; // 4 2
//The following code defines vectors v = (3, 1) and u = (2, 2), and after that
//calc ulates the sum s = v + u .
P v = {3,1};
P u = {2,2};
P s = v+u;
cout << s.X << " " << s.Y << " \n"; // 5 3
//The following code calculates the distance between points (4, 2) and (3, - 1):
//abs() == root(x^2 + y2)
P a = {4,2};
P b = {3,-1};
//The following code calculates the angle of the vector (4, 2), rotates it 1/2
//radians counte rcl ock wise, and then calculates the angle again:
//arg(v) used to calculate angle of a vector with respect to x
//polar(s, a) constructs a vector whose length is s and that points to an angle a.
P v = {4,2};
cout << arg(v) << " \n"; // 0.463648
v *= polar( 1.0 ,0.5);
cout << arg(v) << " \n"; // 0.963648
cout << abs(b-a) << " \n"; // 3.16228
```

Points and lines

```
P a = {4,2};
```

Points and lines (cont)

> P b = {1,2};
C p = (conj(a)*b).Y; // 6
The above code works, because the function conj negates the y coordinate of
a vector, and when the vectors (x 1 , − y 1) and (x 2 , y 2) are multiplied together, the y
coordinate of the result is x 1 y 2 − x 2 y 1 .

The cross product a × b of vectors a = (x 1 , y 1) and b = (x 2 , y 2) is calculated using
the formula x 1 y 2 − x 2 y 1 . The cross product tells us whether b turns left (positive
value), does not turn (zero) or turns right (negative value) when it is placed
directly after a .

Coin problem

```
bool ready[N];
int value[N];
int solve(int x) {
if (x < 0) return INF;
if (x == 0) return 0;
if (ready[x]) return value[x];
int best = INF;
for (auto c : coins) {
best = min(best, solve( x-c )+1);
// Note that we can also iterat ively construct the array value using a loop that
//simply calculates all the values of solve for parameters 0 . . . n :
value[0] = 0;
for (int x = 1; x <= n; x++) {
value[x] = INF;
for (auto c : coins) {
if (x-c >= 0) {
value[x] = min(va lue[x], value[ x-c ]+1);
}
}
}
}
value[x] = best;
ready[x] = true;
return best;
}
//cons tru cting the solution
int first[N];
value[0] = 0;
for (int x = 1; x <= n; x++) {
```

Coin problem (cont)

```
> value[x] = INF;
for (auto c : coins) {
if (x-c >= 0 && value[x-c]+1 < value[x]) {
value[x] = value[x-c]+1;
first[x] = c;
}
}
}
//to print
while (n > 0) {
cout << first[n] << "\n";
n -= first[n];
}
//Counting the number of solutions
//The following code constructs an array count such that count [ x ] equals the
//value of solve ( x ) for 0 ≤ x ≤ n :
count[0] = 1;
for (int x = 1; x <= n; x++) {
for (auto c : coins) {
if (x-c >= 0) {
count[x] %= m;
count[x] += count[x-c];
}
}
}
```

where ready [x] indicates whether the value of solve (x) has been calculated,
and if it is, value [x] contains this value. The constant N has been chosen so that
all required values fit in the arrays.

set implimentation using bit manuplation

```
int x = 0;
x |= (1< <1);
x |= (1< <3);
x |= (1< <4);
x |= (1< <8);
cout << __buil tin _po pco unt(x) << " \n"; // 4
//Then, the following code prints all elements that belong to the set:
for (int i = 0; i < 32; i++) {
if (x&(1 <<i)) cout << i << " ";
}
```

set implimentation using bit manuplation (cont)

```
> // output: 1 3 4 8
//For example, the following code first constructs the sets x = {1, 3, 4, 8} and
//y = {3, 6, 8, 9}, and then constructs the set z = x ∪ y = {1, 3, 4, 6, 8, 9}:
int x =
int y =
int z =
cout <<
(1<<1)|(1<<3)|(1<<4)|(1<<8);
(1<<3)|(1<<6)|(1<<8)|(1<<9);
x|y;
__builtin_popcount(z) << "\n"; // 6
//The following code goes through the subsets of {0, 1, . . . , n − 1}:
for (int b = 0; b < (1<<n); b++) {
// process subset b
}
//The following code goes through the subsets of a set x :
int b = 0;
do {
// process subset b
} while (b=(b-x)&x);
//Hamming distance
int hamming(int a, int b) {
return __builtin_popcount(a^b);
}
```

maximum sub array

```
# Kadane's Algorithm: O(n)
def kadane s(n ums):
        maxSum = nums[0]
        curSum = 0
       for n in nums:
                curSum = max(cu rSum, 0)
                curSum += n
                maxSum = max(ma xSum, curSum)
        return maxSum
# Return the left and right index of the max subarray sum,
# assuming there's exactly one result (no ties).
# Sliding window variation of Kadane's: O(n)
def slidin gWi ndo w(n ums):
        maxSum = nums[0]
        curSum = 0
        maxL, maxR = 0, 0
       L = 0
       for R in range( len (nu ms)):
               if curSum < 0:
                        curSum = 0
                       L = R
                curSum += nums[R]
               if curSum > maxSum:
                        maxSum = curSum
                        maxL, maxR = L, R
        return [maxL, maxR]
```

Generating subsets

```
//method 1
void search(int k) {
if (k == n) {
// process subset
} else {
search (k+1);
subset.pu sh_ bac k(k);
search (k+1);
subset.po p_b ack();
}
}
```

Generating subsets (cont)

```
> //method 2
for (int b = 0; b < (1<<n); b++) {
vector subset;
for (int i = 0; i < n; i++) {
if (b&(1<<i)) subset.push_back(i);
}
}
```

method 1 - search generates the subsets of the set {0, 1, . . . , n − 1}. The
function maintains a vector subset that will contain the elements of each subset.
The search begins when the function is called with parameter 0.

Method -2 shows how we can find the elements of a subset that
corresponds to a bit sequence. When processing each subset, the code builds a
vector that contains the elements in the subset

Policy-based data structures

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_ pbds;
typedef tree<i nt, nul l_t ype ,le ss, rb_ tre e_tag,
tree_o rde r_s tat ist ics _no de_ upd ate> indexe d_set;
indexe d_set s;
s.inse rt(2);
s.inse rt(3);
s.inse rt(7);
s.inse rt(9);
auto x = s.find _by _or der(2);
cout << *x << " \n"; // 7
cout << s.orde r_o f_k ey(7) << " \n"; // 2
//If the element does not appear in the set, we get the position that the element would have in the //set:
cout << s.orde r_o f_k ey(6) << " \n"; // 2
cout << s.orde r_o f_k ey(8) << " \n"; // 3
```

Comparison functions

```
//the following comparison function comp sorts
//strings primarily by length and second arily by alphab etical order:
bool comp(s tring a, string b) {
if (a.size() != b.size()) return a.size() < b.size();
return a < b;
}
//Now a vector of strings can be sorted as follows:
sort(v.be gin(), v.end(), comp);
```

Comparison operators

```
vector<pair<int,int>> v;
v.push _ba ck( {1,5});
v.push _ba ck( {2,3});
v.push _ba ck( {1,2});
sort(v.be gin(), v.end());
vector <tu ple <in t,i nt, int >> v;
v.push _ba ck( {2, 1,4});
v.push _ba ck( {1, 5,3});
v.push _ba ck( {2, 1,3});
sort(v.be gin(), v.end());
```

Pairs (pair) and tuples are sorted primarily according to their first elements (first). However, if the first elements of two pairs are equal, they are sorted according to
their second elements (second):

Longest increasing subsequence

```
for (int k = 0; k < n; k++) {
length[k] = 1;
for (int i = 0; i < k; i++) {
if (array[i] < array[k]) {
length[k] = max(le ngt h[k ],l eng th[ i]+1);
}
}
}
```

To calculate a value of length (k), we should find a position i < k for which array [i] < array [k] and length (i) is as large as possible. Then we know that length (k) = length (i) + 1, because this is an optimal way to add array [k] to a subsequence. However, if there is no such position i , then length (k) = 1, which means that the subsequence only contains array [k].

Bitset

```
bitset<10> s;
s[1] = 1;
s[3] = 1;
s[4] = 1;
s[7] = 1;
cout << s[4] << " \n"; // 1
cout << s[5] << " \n"; // 0
bitset <10> s(stri ng( " 001 001 101 0")); // from right to left
cout << s[4] << " \n"; // 1
cout << s[5] << " \n"; // 0
//count returns the number of one in the bit seet
```

Bitset (cont)

```
> bitset<10> s(string("0010011010"));
cout << s.count() << "\n"; // 4
// using bit operation
bitset<10> a(string("0010110110"));
bitset<10> b(string("1011011000"));
cout << (a&b) << "\n"; // 0010010000
cout << (a|b) << "\n"; // 1011111110
cout << (a^b) << "\n"; // 1001101110
```

Generating permutations of a set

```
//ex {0, 1, 2} are (0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)
//Method 1 using recursion
void search() {
if (permu tat ion.size() == n) {
// process permut ation
} else {
for (int i = 0; i < n; i++) {
if (chose n[i]) continue;
chosen[i] = true;
permut ati on.p us h_b ack(i);
search();
chosen[i] = false;
permut ati on.p op _ba ck();
}
}
}
//method 2 using the c++ stl next_p erm utation
vector permut ation;
for (int i = 0; i < n; i++) {
permut ati on.p us h_b ack(i);
}
do {
// process permut ation
} while (next_ per mut ati on( per mut ati on.b eg in( ),p erm uta tio n.e nd()));
```

Method - 1 - search goes through the permutations of the set {0, 1, . . . , n − 1}. The function builds a vector permutation that contains the permutation, and the search begins when the function is called without parameters.

Method - 2 - Another method for generating permutations is to begin with the permutation {0, 1, . . . , n − 1} and repeatedly use a function that constructs the next permutation in increasing order.

prime Factorization

```
//The function divides n by its prime factors, and adds them to the vector.
vector <in t> factor s(int n) {
vector <in t> f;
for (int x = 2; x*x <= n; x++) {
while (n%x == 0) {
f.push _ba ck(x);
n /= x;
}
}
if (n > 1) f.push _ba ck(n);
return f;
}
```

Sieve of Eratosthenes

```
for (int x = 2; x <= n; x++) {
if (sieve[x]) continue;
for (int u = 2*x; u <= n; u += x) {
sieve[u] = x;
}
}
```

Modular exponentiation

```
//The following function calculates the value of x n mod m :
int modpow(int x, int n, int m) {
if (n == 0) return 1%m;
long long u = modpow (x, n/2,m);
u = (u*u)%m;
if (n%2 == 1) u = (u*x)%m;
return u;
}
```

Diophantine equations

ax + b y = gcd(a, b)
A Diophantine equation can be solved if c is divisible by gcd(a, b), and otherwise it cannot be solved.
39 x + 15 y = 12
gcd(39, 15) = gcd(15, 9) = gcd(9, 6) = gcd(6, 3) = gcd(3, 0) = 3
A solution to a Diophantine equation is not unique, because we can form an infinite number of solutions if we know one solution. If a pair (x, y) is a solution, then also all pairs

Diophantine equations (cont)

(x + kb/gcd(a,b), y - ka/gcd(a,b)) are solutions, where k is any integer.

Point distance from a line

(a − c) × (b − c) / 2

shortest distance between p, s1, s2 which are the vertex of a triangle is

d = ((s1 - p) * (s2 - p)) / |s2 - s1|

Area of polygon

```
// C++ program to evaluate area of a polygon using
// shoelace formula
#include <bi ts/ std c++.h>
using namespace std;

// (X[i], Y[i]) are coordi nates of i'th point.
double polygo nAr ea( double X[], double Y[], int n)
{
       // Initialize area
        double area = 0.0;

       // Calculate value of shoelace formula
       int j = n - 1;
       for (int i = 0; i < n; i++)
       {
               area += (X[j] + X[i]) * (Y[j] - Y[i]);
               j = i; // j is previous vertex to i
       }

       // Return absolute value
        return abs(area / 2.0);
}

// Driver program to test above function
int main()
{
        double X[] = {0, 2, 4};
        double Y[] = {1, 3, 7};

       int n = sizeof (X) /si zeo f(X [0]);

       cout << polygo nAr ea(X, Y, n);
}
```

Finding the smallest solution

```
int x = -1;
for (int b = z; b >= 1; b /= 2) {
while (!ok(x+b)) x += b;
}
int k = x+1
```

An important use for binary search is to find the position where the value of a function changes. Suppose that we wish to find the smallest value k that is a valid solution for a problem. We are given a function ok (x) that returns true if x is a valid solution and false otherwise. In addition, we know that ok (x) is false when x < k and true when x ≥ k .

vector

```
vector v;
v.push _ba ck(3); // [3]
v.push _ba ck(2); // [3,2]
v.push _ba ck(5); // [3,2,5]
cout << v[0] << " \n"; // 3
cout << v[1] << " \n"; // 2
cout << v[2] << " \n"; // 5
for (int i = 0; i < v.size(); i++) {
cout << v[i] << " \n";
}
for (auto x : v) {
cout << x << " \n";
}
sort(v.be gin(), v.end());
revers e(v.be gin(), v.end());
random _sh uff le( v.b egin(), v.end());
//can also be used in the ordinary array like below
sort(a, a+n);
reverse(a, a+n);
random _sh uff le(a, a+n);
```

Priority queue

```
priority_queue q;
q.push(3);
q.push(5);
q.push(7);
q.push(2);
cout << q.top() << " \n"; // 7
```

Priority queue (cont)

```
> q.pop();
cout << q.top() << "\n"; // 5
q.pop();
q.push(6);
cout << q.top() << "\n"; // 6
q.pop();
```

If we want to create a priority queue that supports finding and removing the smallest element, we can do it as follows:

priority_queue<int,vector,greater> q;

Stack

```
stack s;
s.push(3);
s.push(2);
s.push(5);
cout << s.top(); // 5
s.pop();
cout << s.top(); // 2
```

User-defined structs

```
struct P {
int x, y;
bool operat or< (const P &p) {
if (x != p.x) return x < p.x;
else return y < p.y;
}
};
```

For example, the following struct P contains the x and y coordinates of a point.
The comparison operator is defined so that the points are sorted primarily by the x coordinate and secondarily by the y coordinate.

BackTracking

```
void search(int y) {
if (y == n) {
count++;
return;
}
for (int x = 0; x < n; x++) {
if (column[x] || diag1[x+y] || diag2[ x-y +n-1]) continue;
column[x] = diag1[x+y] = diag2[ x-y +n-1] = 1;
search (y+1);
column[x] = diag1[x+y] = diag2[ x-y +n-1] = 0;
```

BackTracking (cont)

```
>}
}
```

Two pointers method (copy)

Subarray sum problem
s1-The initial subarray contains 3 elements check sum if not contnue to
s2- The left pointer moves one step to the right. The right pointer does not move check sum if not
s3- Again, the left pointer moves one step to the right, and this time the right pointer moves three steps to the right check sum if not loop back to s1
2Sum problem
s1-The initial positions of the pointers are the left pointer at index 0 and the right pointer is at the last index
s2- Then the left pointer moves one step to the right. The right pointer moves three steps to the left
s3- After this, the left pointer moves one step to the right again. The right pointer does not move
s4- loopback to s1

Two pointers method

Subarray sum problem
s1-The initial subarray contains 3 elements check sum if not contnue to
s2- The left pointer moves one step to the right. The right pointer does not move check sum if not
s3- Again, the left pointer moves one step to the right, and this time the right pointer moves three steps to the right check sum if not loop back to s1
2Sum problem
s1-The initial positions of the pointers are the left pointer at index 0 and the right pointer is at the last index
s2- Then the left pointer moves one step to the right. The right pointer moves three steps to the left
s3- After this, the left pointer moves one step to the right again. The right pointer does not move
s4- loopback to s1

Finding the maximum value

```
int x = -1;
for (int b = z; b >= 1; b /= 2) {
while (f(x+b) < f(x+b+1)) x += b;
}
int k = x+1;
```

The idea is to use binary search for finding the largest value of x for which
f (x) < f (x + 1). This implies that k = x + 1 because f (x + 1) > f (x + 2).

multiset

```
multiset s;
s.inse rt(5);
s.inse rt(5);
s.inse rt(5);
cout << s.count(5) << " \n"; // 3
```

multiset (cont)

```
> s.erase(5);
cout << s.count(5) << "\n"; // 0
s.erase(s.find(5));
cout << s.count(5) << "\n"; // 2
```

Bitset

```
bitset<10> s;
s[1] = 1;
s[3] = 1;
s[4] = 1;
s[7] = 1;
cout << s[4] << " \n"; // 1
cout << s[5] << " \n"; // 0
bitset <10> s(stri ng( " 001 001 101 0")); // from right to left
cout << s[4] << " \n"; // 1
cout << s[5] << " \n"; // 0
bitset <10> s(stri ng( " 001 001 101 0"));
cout << s.count() << " \n"; // 4 num of 1
```

queue

```
queue q;
q.push(3);
q.push(2);
q.push(5);
cout << q.front(); // 3
q.pop();
cout << q.front(); // 2
```

Deque

```
deque d;
d.push _ba ck(5); // [5]
d.push _ba ck(2); // [5,2]
d.push _fr ont(3); // [3,5,2]
d.pop_ back(); // [3,5]
d.pop_ fro nt(); // [5]
```

Knapsack problems

```
possible ( x, k ) = possible ( x − w k , k − 1) V possible ( x, k − 1)
possib le[0] = true;
for (int k = 1; k <= n; k++) {
for (int x = W; x >= 0; x--) {
if (possi ble[x]) possib le[ x+w[k]] = true;
}
}
```

In this section, we focus on the following problem: Given a list of weights
[w 1 , w 2 , . . . , w n], determine all sums that can be constructed using the weights.
For example, if the weights are [1, 3, 3, 5], the following sums are possible: all sum between 0 to 12 except 2 and 10

Paths in a grid

```
int sum[N][N];
for (int y = 1; y <= n; y++) {
for (int x = 1; x <= n; x++) {
sum[y][x] = max(su m[y ][x -1] ,su m[y -1] [x] )+v alu e[y ][x];
}
}
```

find a path from the upper-left corner to the lower-right
corner of an n × n grid, such that we only move down and right. Each square
contains a positive integer, and the path should be constructed so that the sum of
the values along the path is as large as possible.

Set iterators

```
//points to the smallest element in the set
auto it = s.begin();
cout << *it << " \n";
//print in increasing order
for (auto it = s.begin(); it != s.end(); it++) {
cout << *it << " \n";
}
//print the largest element in the set
auto it = s.end(); it--;
cout << *it << " \n";
// find the element x in the set
auto it = s.find(x);
if (it == s.end()) {
// x is not found
}
//find element nearest to the element x
```

Set iterators (cont)

```
> auto it = s.lower_bound(x);
if (it == s.begin()) {
cout << *it << "\n";
} else if (it == s.end()) {
it--;
cout << *it << "\n";
} else {
int a = *it; it--;
int b = *it;
if (x-b < a-x) cout << b << "\n";
else cout << a << "\n";
}
```

Bitmanuplation

```
int x =
cout <<
cout <<
cout <<
cout <<
5328; // 000000 000 000 000 000 010 100 110 10000
__buil tin _clz(x) << " \n"; // 19
__buil tin _ctz(x) << " \n"; // 4
__buil tin _po pco unt(x) << " \n"; // 5
__buil tin _pa rity(x) << " \n"; // 1
```