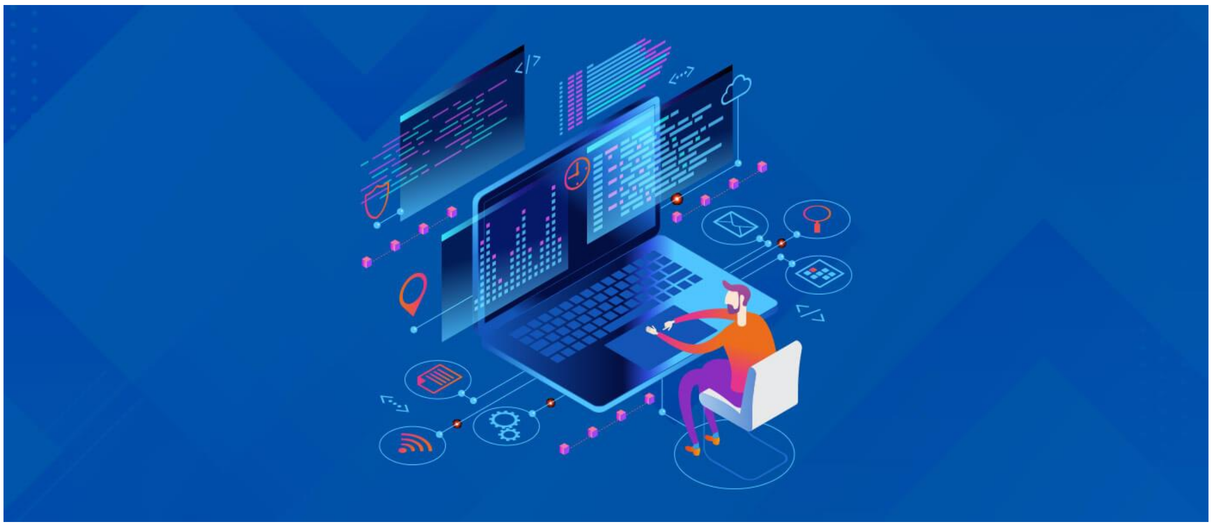
Competitive Programming



CODECHEF
An unacademy Educational Initiative

CODEFORCES

HackerRank

Programming Techniques

Template

It is a set a pre-written code to initialize and fasten the coding process. As CP is all about accuracy and speed. So, template plays an important role in submitting. Given below is a typical template for CP.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-14 17:32:56
*/
#include <bits/stdc++.h>

using namespace std;
#define FAST                      \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);                   \
    cout.tie(0); //To fasten the process
int main()
{
    FAST
    //Solution comes here
    return 0;
}
```

Shortening Code

Codes can be shortened in competitive coding using two ways:

- Type names: By using the `typedef` keyword. For example:

  ```
  typedef vector<int> vi;
  typedef pair<int, int> pii;
  ```

  Later on, `vi` can be used for vector of integers and `pii` can be used for a pair of integers.

- Macros: By using the `#define` keyword. For example:

  ```
  #define FOR(i, a, b) for (int i = a; i <= b; i++)
  #define ROF(i, a, b) for (int i = a; i >= b; i--)
  ```

Programming Basics

Input and Output

In C++, the standard streams `cin` is used for input and `cout` is used for output. It works as follows:

```
int a, b;
string str;
cin >> a >> str >> b;
cout << a << " " << str << " " << b << endl;
```

NOTE: newline " `\n` " works faster than "`endl`"

We can also use the C inbuilt functions `scanf` and `printf` for input and output.

To take string inputs, we either use `cin` or `getline`.

```
string str;
cin >> str;
getline(cin, str);
```

Well, taking input using getline is quite cool but this also invites a problem. Sometimes when you have to take string inputs the problem may have a large number of test cases, then you might see some mismatching in strings, mismatching in the sense, suppose your question is having 4 test cases, and in each one of them, you will have to take 1 input of string, in such cases most online compilers will take 3 string inputs. To avoid such problems, you should follow the following code.

```
int t;

cin >> t;
cin.ignore();
while (t--)
{
    string str;
    getline(cin, str);
}
```

When we want to print output in scientific notation, we will use `fixed` keyword.

```
c = 1e24;
cout << fixed << c << endl;
```

When number of inputs is unknown, the following loop is used:

```
while (cin >> x)
{
    // code
}
```

When we want to add values at the end of a string, one way is `str = str + ch;`

and another way is `str.push_back(ch);`

Working with numbers

```
cout << 7/2 << endl;
```

The expected output for this code is 3.5 but the actual output is 3.

```
cout << 7/2.0 << endl;
```

The expected output for this code is 3.5 but the actual output is 3.5.

because once a higher datatype is found, the whole calculation is done in that higher datatype. The precedence order of the datatypes is:

```
double -> float -> long long int -> long int -> int -> char
```

Data Types	*Range*
`char`	`-127 to 127 or 0 to 255`
`unsigned char`	`0 to 255`
`signed char`	`-127 to 127`
`int`	`-2147483648 to 2147483647`
`unsigned int`	`0 to 4294967295`
`signed int`	`-2147483648 to 2147483647`
`short int`	`-32768 to 32767`
`unsigned short int`	`0 to 65,535`
`signed short int`	`-32768 to 32767`
`long int`	`-2,147,483,648 to 2,147,483,647`
`signed long int`	`same as long int`
`unsigned long int`	`0 to 4,294,967,295`
`long long int`	-(2^{63}) to (2^{63})-1
`unsigned long long int`	`0 to 18,446,744,073,709,551,615`
`wchar_t`	`1 wide character`

NOTE: The g++ compiler provides a 128-bit data type `__int128_t` that can store values in the range -2^{127} to $2^{127}+1$.

NOTE: It is risky to compare floating point numbers as there may be some rounding errors. In such cases we should assume them to be equal if the absolute difference between them is equal to some small number ε. ε can be as small as 10^{-9}.

Modular Arithmetic

Sometimes the answer of the problem becomes so large, that it becomes difficult to store such large numbers, in such cases we are asked to output the numbers modulo m, where m is some prime. In most of the coding contests we will see that the value of m is selected as 10e9+7. Actually, it stores the remainder obtained by dividing the larger number with m. There are a few properties of modulo or remainders.

- `(a + b) mod m = ((a mod m) + (b mod m)) mod m`
- `(a - b) mod m = ((a mod m) - (b mod m)) mod m`
- `(a x b) mod m = ((a mod m) x (b mod m)) mod m`

One application of the modular arithmetic could be found in the code for finding the factorial of a number.

```
long long x = 1;
for (int i = 1; i <= n; i++)
{
    x = (x * i) % m
}
cout << x << "\n";
```

Modular (or Binary) Exponentiation

Another application of the modular arithmetic could be found in the code for finding the exponentiation of any number, i.e, x^y. Now, obviously, one can solve this problem by the naïve method, that is the repeated multiplication method.

```
long long ans = 1;
for (int i = 1; i <= y; i++)
{
    ans = (ans * x) % m
}
cout << ans << "\n";
```

but, this will take magnificently much more time if say, y ≥ 10.

So, we will use an optimized algorithm, namely repeated square paradigm to deal with this.

But for that, we will make use of recursion, which we will see next. Read the content written below, iff you are confident enough with Recursions.

What Repeated Square Paradigm does is,

```
if y == 0, return 1
else if y % 2 == 0,  return ((x^(y/2) % m) * (x^(y/2) % m)) % m
else, return ((x^⌊y/2⌋ % m) * (x^⌊y/2⌋ % m) * x) % m
```

Again, to get rid of overflows, we will take modular multiplication of two numbers at a time.

So,

`return ((x^⌊y/2⌋ % m) * (x^⌊y/2⌋ % m) * x) % m` can be rephrased as

`return (((x^⌊y/2⌋ % m) * (x^⌊y/2⌋ % m)) % m) * (x % m)) % m`

Let's see the C++ Code for this

```
long long int power(long long int x, long long int y, long long int m)
{
    if (y == 0)
    {
        return 1;
    }
    else if (y % 2 == 0)
    {
        return (power(x, (y / 2), m) * power(x, (y / 2), m)) % m;
    }
    else
    {
        return (((power(x, floor(y / 2), m) * power(x, floor(y / 2), m)) % m) * (x % m))
% m;
    }
}
```

Well, there is also an iterative method for chopping this Modular Exponentiation, also known as Binary Exponentiation. The Algorithm is very simple.

Suppose, we have x^y, we can convert y into binary. Let's say, the binary of y is $y_{b_k}y_{b_{k-1}}y_{b_{k-2}}y_{b_{k-3}}y_{b_{k-4}} \cdots y_{b_1}$.

So, $x^y = x^{\left(y_{b_k}y_{b_{k-1}}y_{b_{k-2}}y_{b_{k-3}}y_{b_{k-4}} \cdots y_{b_1}\right)_2}$

What we can do is, set a variable to 1, say `ans = 1`.

We will keep on right shifting the binary of y, and multiplying x to itself.

We will stop when the power, i.e., binary of y becomes 0.

If the rightmost bit of the binary digit formed after each right shift is set, I mean, is 1, we will multiply current value of base, i.e., x, to the variable `ans`

Let's see the C++ Code for this

```
long long int bin_expo(long long int x, long long int y, long long int m)

{

    long long int ans = 1;

    while (y != 0) // Checking whether binary of y became 0 or not?

    {

        if (y & 1 == 1) // Checking whether the rightmost digit of the binary of y is
set or not?

        {

            ans = ((ans % m) * (x % m)) % m;

        }

        y = y >> 1;

        x = ((x % m) * (x % m)) % m;

    }

    return ans;

}
```

NOTE: Amongst the Recursive and Iterative Method, the Iterative Method works better, because it's time complexity is in logarithmic terms.

Now, again the code mentioned above isn't fool proof. Our Code can work well for larger values of 'x', but will potentially fail for higher values of 'y' and 'm'

Firstly, we will try to optimize for higher values of m, and we can do so by breaking multiplication operatives inside the binary exponentiation function into repetitive addition operatives. As we know, the basic arithmetic operations are Addition and Subtraction, working upon which, we can develop various other advanced arithmetic functions like multiplication, division, which can further be hybridised to form operations like exponentiations. You might have noticed in the previous code fragment, `bin_expo`, we were simply multiplying fragments. Now, when the m value becomes higher than permissible range of integers, the task for which modular arithmetic is indulged will be compromised, due to overflow during multiplication. To get rid of that, we will indulge another function, `bin_mult`, that will break the multiplication into addition and apply notion of modular arithmetic in each steps. Well!. Let's see the C++ code

```
long long int bin_mult(long long int x, long long int y, long long int m)
```

```
{
    long long int ans = 0;
    while (y != 0)
    {
        if (y & 1 == 1)
        {
            ans = ((ans % m) + (x % m)) % m;
        }
        y = y >> 1;
        x = ((x % m) + (x % m)) % m;
    }
    return ans;
}
long long int bin_expo(long long int x, long long int y, long long int m)
{
    long long int ans = 1;
    while (y != 0)
    {
        if (y & 1 == 1)
        {
            ans = bin_mult((ans % m), (x % m), m);
        }
        y = y >> 1;
        x = bin_mult((x % m), (x % m), m);
    }
    return ans;
}
```

There is an important Mathematical Theorem, namely, Fermat's Little Theorem, which says, if m is prime, $x^{(m - 1)}$ % m = 1

Now, there is a question in your mind...Where to use this weirdo ?

Well, let us see an Example.

Q. Simplify $x^{y^z} \% \, m$

Solution:

We know, Dividend = Divisor × Quotient + Remainder

Compiling this,

$$y^z = (m-1) \times \left\lfloor \frac{y^z}{m-1} \right\rfloor + y^z \bmod (m-1)$$

So, $x^{y^z} = x^{(m-1)\times\left\lfloor \frac{y^z}{m-1}\right\rfloor + y^z \bmod (m-1)} = \left(x^{(m-1)\times\left\lfloor \frac{y^z}{m-1}\right\rfloor}\right) \times \left(x^{y^z \bmod (m-1)}\right)$

$$x^{y^z} \bmod m = \left(x^{(m-1)\times\left\lfloor \frac{y^z}{m-1}\right\rfloor} \bmod m\right) \times \left(x^{y^z \bmod (m-1)} \bmod m\right)$$
$$= \left(x^{y^z \bmod (m-1)} \bmod m\right)$$

Now, we will be learning about the generalizations (isn't limited to primes only) of the FLT – Euler's Theorem, which says,

$$x^{\varphi(m)} \bmod m = 1$$

where,

$\varphi(m)$ is the Euler's Totient Function, which counts the positive integers up to a given integer m that are relatively prime to m.

Euler's Theorem can be rephrased as

$$x^{m\prod_{p|m}\left(1-\frac{1}{p}\right)} \bmod m = 1$$

where the product $\prod_{p|m}\left(1-\frac{1}{p}\right)$ is over the distinct prime numbers dividing m

Moreover, according to Euler's Theorem,

$$x^y \bmod m = x^{y \bmod \varphi(m)} = x^{y \bmod \left(m\times\left(\prod_{p|m}\left(1-\frac{1}{p}\right)\right)\right)} \bmod m$$

Some of the question based on these concepts are:

- Shopping | CodeChef

Solution: 66258108 | CodeChef

- Modular Equation | CodeChef
- Super Pow - LeetCode

```
class Solution

{

public:

    int bin_expo(int x, int y, int m)

    {
        int ans = 1;
        while (y)
        {
            if (y & 1)
            {
                ans = ((ans % m) * (x % m)) % m;
            }
            y >>= 1;
            x = ((x % m) * (x % m)) % m;
        }
        return ans;
    }
    int superPow(int a, vector<int>& b)
    {
        /*
        1337 = 7 * 191
        Phi(1337) = 1337 * (1 - 1 / 7) * (1 - 1 / 191) = 1140

        By Euler's Theorem,
        (a ^ b) mod m = (a ^ (b mod Phi(m))) mod m
        */

        int b_modular = 0;
        for(int i = 0; i < b.size(); i++)
        {
            b_modular += ((b[i] % 1140) * bin_expo(10, b.size() - 1 - i, 1140)) % 1140;
        }
        return bin_expo(a, b_modular, 1337);
    }
};
```

Recursion

It works on the principle of Divide and Conquer to achieve its goal. A function is declared and it is called multiple times within it.

```
type fun(parameters)
{
    /*CODE*/
    fun(parameters);
}
```

Some of the questions based on these applications are:

- Add-Ons

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 01:01:37
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
int fun(int n)
{
    if (n == 1)
    {
        return 1;
    }
    else
    {
        return n + fun(n - 1);
    }
}
int main()
{
    int n;
    cin >> n;
    cout << fun(n) << endl;
    getch();
    return 0;
}
```

- pow (n, p)

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 01:07:09
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
int fun(int n, int p)
{
    if (p == 0)
    {
        return 1;
    }
    else
    {
        return n * fun(n, p - 1);
    }
}
int main()
{
    int n, p;
    cin >> n >> p;
    cout << fun(n, p);
    getch();
    return 0;
}
```

- Fact-o-real

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 01:11:40
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
int fun(int n)
{
    if (n == 0 || n == 1)
    {
        return 1;
    }
    else
    {
        return n * fun(n - 1);
    }
}
int main()
{
    int n;
    cin >> n;
    cout << fun(n);
    getch();
    return 0;
}
```

- n'th son of Fibonacci

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 01:34:06
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
int fun(int n)
```

```
{
    if (n == 0 || n == 1 || n == 2)
    {
        return n;
    }
    else
    {
        return fun(n - 1) + fun(n - 2);
    }
}
int main()
{
    int n;
    cin >> n;
    cout << fun(n);
    getch();
    return 0;
}
```

- Is it Sorted?

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 01:41:18
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
bool fun(vector<int> v, int n)
{
    if (n == 1)
    {
        return true;
    }
    if (v[0] < v[1])
    {
        v.erase(v.begin());
        if (fun(v, n - 1) == true)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
int main()
{
    int n;
    cin >> n;
    vector<int> v(n);
    for (auto &i : v)
    {
        cin >> i;
    }
    if (fun(v, n) ) == true)
    {
        cout << "YES" << endl;
    }
    else
    {
        cout << "NO" << endl;
    }
    getch();
    return 0;
}
```

- Reverse the string

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 04:00:50
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
void fun(string str)
{
    if (str.size() == 0)
    {
        return;
    }
    else
    {
        char ch = str[0];
        fun(str.substr(1));
        cout << ch << " ";
    }
}
int main()
{
    string str;
    cin >> str;
    fun(str);
    getch();
    return 0;
}
```

- Replacements

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 04:09:17
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
void fun(string str)
{
    if (str.size() == 0)
    {
        return;
    }
    else
    {
        if (str[0] == 'p' && str[1] == 'i')
        {
            cout << 3.14;
            fun(str.substr(2));
        }
        else
        {
            cout << str[0];
            fun(str.substr(1));
        }
    }
}
int main()
{
    string str;
    getline(cin, str);
    fun(str);
    getch();
    return 0;
}
```

- Transfer - ☹

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 04:11:50
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
void fun(string str, char ch, int count)
{
    if (str.size() == 0)
    {
        for (int i = 0; i < count; i++)
        {
            cout << ch;
        }
        return;
    }
    else
    {
        if (str[0] == ch)
        {
            count++;
            fun(str.substr(1), ch, count);
        }
        else
        {
            cout << str[0];
            fun(str.substr(1), ch, count);
        }
    }
}
int main()
{
    string str;
    getline(cin, str);
    char ch;
    cin >> ch;
    fun(str, ch, 0);
    getch();
    return 0;
}
```

- Subsets

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 01:41:18
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
void fun(string str, string ans)
{
    char ch = str[0];
    if (str.size() == 0)
    {
        if (ans == "")
        {
            cout << "\"\"" << endl;
        }
        else
        {
            cout << ans << endl;
        }
        return;
    }
    string res = str.substr(1);
    fun(res, ans);
    fun(res, ans + ch);
}
int main()
{
    string s;
    cin >> s;
    fun(s, "");
    getch();
    return 0;
}
```

- Permutations

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 04:47:41
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<int> v(n);
    for (auto &i : v)
    {
        cin >> i;
    }
    do
    {
        for (auto i : v)
        {
            cout << i << " ";
        }
        cout << endl;
    } while (next_permutation(v.begin(), v.end()));
    getch();
    return 0;
}
```

- Reaching n'th Stair with at most k jumps

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 04:52:05
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
int fun(int start, int end, int jumps)
{
    if (start == end)
    {
        return 1;
    }
    if (start >= end)
    {
        return 0;
    }
    else
    {
        int count = 0;
        for (int i = 1; i <= jumps; i++)
        {
            count += fun(start + i, end, jumps);
        }
        return count;
    }
}
int main()
{
    int n, k;
    cin >> n >> k;
    cout << fun(0, n, k) << endl;
    getch();
    return 0;
}
```

Backtracking

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

Some of the questions based on these applications are:

- Poor Rat

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 05:22:17
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
bool isxySafe(int **arr, int x, int y, int n)
{
    if ((x < n && y < n) && (arr[x][y] != 0))
    {
        return true;
    }
    return false;
```

```
}
bool fun(int **arr, int x, int y, int n, int **path)
{
    if ((x == n - 1) && (y == n - 1))
    {
        path[x][y] = 1;
        return true;
    }
    if (isxySafe(arr, x, y, n))
    {
        path[x][y] = 1;
        if (fun(arr, x + 1, y, n, path))
        {
            return true;
        }
        if (fun(arr, x, y + 1, n, path))
        {
            return true;
        }
        path[x][y] = 0;
    }
    return false;
}
int main()
{
    int n;
    cin >> n;
    int **arr = new int *[n];
    for (int i = 0; i < n; i++)
    {
        arr[i] = new int[n];
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> arr[i][j];
        }
    }
    int **path = new int *[n];
    for (int i = 0; i < n; i++)
    {
        path[i] = new int[n];
        for (int j = 0; j < n; j++)
        {
            path[i][j] = 0;
        }
    }
    if (fun(arr, 0, 0, n, path))
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                cout << path[i][j] << " ";
            }
            cout << endl;
        }
    }
    getch();
    return 0;
}
```

- Green-eyed Queens

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-29 05:37:59
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
bool isxySafe(int **arr, int x, int y, int n)
{
    for (int i = 0; i < x; i++)
    {
        if (arr[i][y] == 1)
        {
            return false;
        }
    }
    int i = x, j = y;
    while (i >= 0 && j >= 0)
    {
        if (arr[i][j] != 0)
        {
            return false;
        }
        i--;
        j--;
    }
    i = x;
    j = y;
    while (i >= 0 && j < n)
    {
        if (arr[i][j] != 0)
        {
            return false;
        }
        i--;
        j++;
    }
    return true;
}
bool fun(int **arr, int x, int y, int n)
{
    if (x >= n)
    {
        return true;
    }
    for (int j = 0; j < n; j++)
    {
        if (isxySafe(arr, x, j, n))
        {
            arr[x][j] = 1;
            if (fun(arr, x + 1, j, n))
            {
                return true;
            }
        }
        arr[x][j] = 0;
    }
    return false;
}
int main()
{
    int n;
    cin >> n;
    int **arr = new int *[n];
    for (int i = 0; i < n; i++)
    {
        arr[i] = new int[n];
        for (int j = 0; j < n; j++)
        {
            arr[i][j] = 0;
        }
    }
    if (fun(arr, 0, 0, n))
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                cout << arr[i][j] << " ";
            }
            cout << endl;
        }
    }
    getch();
    return 0;
}
/*
* * * * * j(Columns)
*
*
*
*
i(Rows)
*/
```

Bit Manipulation

A binary number is the one made by combinations of zeroes and ones only. Unlike humans, computers don't understand decimals, they are familiar with the black and white world of zeroes and ones 😜.
To convert a bit representation $b_k...b_2b_1b_0$ to decimal, we will use the formula:

$$Decimal = b_k2^k + b_{k-1}2^{k-1} + ... + b_22^2 + b_12^1 + b_02^0$$

A signed variable of n- bits can contain any integers in the range -2^{n-1} and $2^{n-1} - 1$.
An unsigned variable of n- bits can contain any integers in the range 0 and 2^{n-1}.

NOTE: To find the additive inverse of any binary number, we first invert all the zeroes to ones and vice versa and finally add 1 to the answer.

Bitwise Operations

- AND Operation (&): Let X and Y be two operands. The Binary 'AND' operation between them can be demonstrated using the truth table as follows:

X	Y	X & Y
0	0	0
0	1	0
1	0	0
1	1	1

- OR Operation (|): Let X and Y be two operands. The Binary 'OR' operation between them can be demonstrated using the truth table as follows:

X	Y	X & Y
0	0	0
0	1	1
1	0	1
1	1	1

- NOT Operation (~): Let X be an operand. The Binary 'NOT' operation can be demonstrated using the truth table as follows:

X	~X
0	1
1	0

NOTE: ~x = -x - 1

- XOR Operation (^): Let X and Y be two operands. The Binary ‘XOR’ operation between them can be demonstrated using the truth table as follows:

X	Y	X ^ Y
0	0	0
0	1	1
1	0	1
1	1	0

NOTE: XOR operation has some astonishing properties. Some of them are:

1. X ^ Y = Y ^ X
2. X ^ X = 0
3. X ^ 0 = X
4. X ^ (Y ^ Z) = (X ^ Y) ^ Z

Q. Check if a number is Even or Odd using Bitwise Operation.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 15:45:08
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    if (n & 1 == 1)
    {
        cout << "ODD" << endl;
    }
    else
    {
        cout << "EVEN" << endl;
    }
    getch();
    return 0;
}
```

Q. Check if a number is any power of 2.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 15:49:25
*/
#include <bits/stdc++.h>
```

```
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    if ((n & (n - 1)) == 0 && n != 0)
    {
        cout << "YES" << endl;
    }
    else
    {
        cout << "NO" << endl;
    }
    getch();
    return 0;
}
```

Q. Swap two numbers using XOR.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-02 11:31:41
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int num1, num2;
    cin >> num1;
    cin >> num2;
    num1 = num1 ^ num2;
    num2 = num2 ^ num1;
    num1 = num2 ^ num1;
    cout << num1 << " " << num2 << endl;
    getch();
    return 0;
}
```

Q. There are some elements in an array. All elements have even count except one. Find that one.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-02 11:45:30
*/
#include <bits/stdc++.h>
```

```
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<int> v(n);
    for (auto &i : v)
    {
        cin >> i;
    }
    int ans = 0;
    for (auto i : v)
    {
        ans ^= i;
    }
    cout << ans << endl;
    getch();
    return 0;
}
```

Bit Shifting

There are two types of Bits Shifting,

Left Shifting (<<): Appends zeroes at the end.

NOTE: `x << k` appends k zeroes at the end of the binary representation of the number x.

E.g.: 14 << 2 = 56

NOTE: `x << k` corresponds to multiplication by 2^k.

Right Shifting (>>): Removes from the end.

NOTE: `x >> k` removes k bits from the end of the binary representation of the number x

E.g.: 49 >> 3 = 6

NOTE: x >> k corresponds to division by 2^k.

Bit Masking

It is the process of representing numbers as bits (set/unset). For example:
A and B are two friends. A goes to school on Day 1, 3, 5, 6. B goes to school on Day 1, 2, 3, 4. They want to come in common days. What should they do? They should come on days 1 and 3. Let's apply Bit Masking.

A = {1, 3, 5, 6} = 0 1 1 0 1 0 1 {Representing the 7 days in terms of bits}
B = {1, 2, 3, 4} = 0 0 0 1 1 1 1 {Representing the 7 days in terms of bits}

A & B = 0 0 0 0 1 0 1 {1 appears in 1st and 3rd position}

Q. The maximum value that the integer data type can hold is 2147483647 = INT_MAX. Represent it in terms of bit masking.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 15:37:56
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int k = 31;
    string bin = bitset<32>(1 << k - 1).to_string(); //Decimal->Binary
    cout << bin << endl;
    cout << (1 << k) - 1 << endl;
    getch();
    return 0;
}
```

Q. There are N≤5000 workers. Each worker is available during some days of this month (which has 30 days). For each worker, you are given a set of numbers, each from interval [1, 30], representing his/her availability. You need to assign an important project to two workers but they will be able to work on the project only when they are both available. Find two workers that are best for the job — maximize the number of days when both these workers are available.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-02 15:29:22
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
void printbinary(int num)
{
    for (int i = 30; i > 0; --i)
    {
        cout << ((num >> i) & 1) << " ";
    }
}
int main()
{
    int num_workers;
    cin >> num_workers;
    vector<int> masks(num_workers, 0);
    for (int i = 0; i < num_workers; i++)
    {
        int days_count, mask = 0;
        cin >> days_count;
        for (int j = 0; j < days_count; j++)
        {
            int day;
            cin >> day;
            mask = mask | (1 << day);
        }
        masks[i] = mask;
    }
    /*
    for(int i=0;i<masks.size();i++)
    {
        printbinary(masks[i]);
        cout << endl;
    }
    */
    int max = 0, a = 0, b = 0;
    for (int i = 0; i < num_workers; i++)
    {
        for (int j = i + 1; j < num_workers; j++)
        {
            if (max < __builtin_popcount(masks[i] & masks[j]))
            {
                max = __builtin_popcount(masks[i] & masks[j]);
                a = i;
                b = j;
            }
        }
    }
    cout << "Person " << a + 1 << " & Person " << b + 1 << ": " << max << endl;
    getch();
    return 0;
}
```

Bit Masking is having a variety of applications. Some of them are:

- <u>Fetching information about the k'th digit of a number</u>.

If the k'th bit of the number is 1 then `(n & (1 << k))` is not equal to zero.

If the k'th bit of the number is 0 then `(n & (1 << k))` is equal to zero.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 16:06:40
*/
```

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

using namespace std;

int main()
{
    int n;
    cin >> n;
    cout << string(bitset<32>(n).to_string()) << endl;
    int k;
    cin >> k;
    if ((n & (1 << k)) != 0)
    {
        cout << "1" << endl;
    }
    else
    {
        cout << "0" << endl;
    }
    getch();
    return 0;
}
```

- <u>Setting the k’th bit to One</u>.

The formula `(n | (1 << k))` sets the k’th bit on fire😜. i.e., sets it to `1`.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 16:09:54
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    cout << string(bitset<32>(n).to_string()) << endl;
    int k;
    cin >> k;
    n = (n | (1 << k));
    cout << string(bitset<32>(n).to_string()) << endl;
    getch();
    return 0;
}
```

- <u>Setting the k’th bit to Zero</u>.

The formula `(n & ~ (1 << k))` sets the k’th bit on ice😜. i.e., sets it to `0`.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 16:17:38
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    cout << string(bitset<32>(n).to_string()) << endl;
    int k;
    cin >> k;
    n = (n & ~ (1 << k));
    cout << string(bitset<32>(n).to_string()) << endl;
    getch();
    return 0;
}
```

- Inverting the k'th bit.

The formula `(n ^ (1 << k))` sets the k'th bit on ice😜 if it was previously on fire😜 and sets it to fire😜 if it was previously on ice😜.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 16:17:38
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    cout << string(bitset<32>(n).to_string()) << endl;
    int k;
    cin >> k;
    n = (n & ~ (1 << k));
    cout << string(bitset<32>(n).to_string()) << endl;
    getch();
    return 0;
}
```

- Setting last bit to Zero.

The formula `(n & (n - 1))` sets the last bit on ice😜.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-08-31 16:29:14
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    cout << string(bitset<32>(n).to_string()) << endl;
    n = (n & (n - 1));
    cout << string(bitset<32>(n).to_string()) << endl;
    getch();
    return 0;
}
```

NOTE: `1 << k` is an integer bit mask. To create a `long long` bit mask, we can write `1LL << k`.

NOTE: Some additional functions are also there to help us work with bits. Some of them are:

1. `__builtin_clz(n)` counts the number of zeroes at the beginning of the bit representation
2. `__builtin_ctz(n)` counts the number of zeroes at the end of the bit representation
3. `__builtin_popcount(n)` counts the number of ones in the binary representation of the number.
4. `__builtin_parity(n)` counts the parity of the number of ones.

- Converting lowercase to uppercase and vice versa.

For lowercase letters, the fifth bit is set while for uppercase letters its unset. So, to convert an uppercase to lowercase, we will have to set the fifth bit and to convert a lowercase to uppercase, we will have to unset the fifth bit.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-01 22:26:48
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    char ch;
    cin >> ch;
    cout << (char)(ch | (1 << 5)) << endl; //UP -> LOW
    ch = ch | (1 << 5);
    cout << (char)(ch & ~(1 << 5)) << endl; //LOW -> UP
    getch();
    return 0;
}
```

NOTE: `char (1 << 5)` can be replaced by the character ' ' (blank space).

So, `char (ch | ' ')` would also convert character ch from uppercase to lowercase.

NOTE: `char ~(1 << 5)` can be replaced by the character '_' (underscore).

So, `char (ch & '_')` would also convert character ch from lowercase to uppercase.

Q. Print all the subsets of an array of numbers.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-02 15:47:44
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<int> v(n);
    for (auto &i : v)
    {
        cin >> i;
    }
    for (int i = 0; i < (1 << n); i++) // 1 << n = 2^n
    {
        for (int j = 0; j < n; j++)
        {
            if ((i & (1 << j)) != 0)
            {
                cout << v[j] << " ";
            }
        }
        cout << endl;
    }
    getch();
    return 0;
}
```

Bitsets

C++ STL have introduced a new type of data structure known as bitsets. It is nothing but an array that can store values 1 and 0. The Syntax for initializing a bitset of a given size is:

```
bitset<size> name;
```

bitsets also an inbuilt function called flip which is called as `bitset_name.flip(parameters);`

It toggles the bitset. If a parameter is passed the value at that position gets toggled and if it is not given any parameters, the whole bitset is toggled. Same applies for another function reset, which is called as

```
bitset_name.reset(parameters);
```

Q. Convert decimal to binary using bitsets.

Solution:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-02 23:46:20
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int dec;
    cin >> dec;
    cout << bitset<32>(dec) << endl;
    getch();
    return 0;
}
```

Efficiency

Time Complexity

It estimates how much time the algorithm will take a certain input. A time complexity is denoted by `O(...)` where the three dots represent some functions.

Calculation Rules

- Rule 1: If the code consists of single commands, then time complexity is `O (1)`. For example:

  ```
  a++;
  b--;
  c = a + b;
  ```

- Rule 2: If the code is executed `n` times, then time complexity is `O (n)`. For example:

  ```
  for (int i = 0; i < n; i++)
  {
        //Code
  }
  ```

- Rule 3: If the code has k nested loops, then time complexity is O (n^k). For example:

```
for (int i = 0; i < n; i++)
{
      for (int j = 0; j < n; j++)
      {
            for (int k = 0; k < n; k++)
            {
                  for (int l = 0; l < n; l++)
                  {
                        ...
                        //Code
                  }
            }
      }
}
```

A Case Analysis: Maximum Subarray Sum

Suppose we are given an array of 8 elements -1, 2, 4, -3, 5, 2, -5, 2 and we need to find the largest possible sum of a sequence of consecutive values in the array. We can do it in 3 ways.

- O (n^3) Time Complexity:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-04 11:35:45
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int size;
    cout << "Enter the size of the array : ";
    cin >> size;
    cout << endl;
    int arr[size];
    for (int i = 0; i < size; i++)
    {
        cout << "\nEnter element " << i + 1 << " : ";
        cin >> arr[i];
    }
    int x = (size * (size + 1)) / 2;
    int store[x];
    int l = 0;
    cout << "\nThe maximum subarray sum : ";
    for (int i = 0; i < size; i++)
    {
        for (int j = i; j < size; j++)
        {
            int count = 0;
            for (int k = i; k <= j; k++)
            {
                count += arr[k];
            }
            store[l] = count;
            l++;
        }
    }
    int max = store[0];
    for (int a = 0; a < x; a++)
    {
        if (store[a] > max)
        {
            max = store[a];
        }
    }
    cout << max;
    getch();
    return 0;
}
```

- O (n^2) Time Complexity:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-04 11:40:47
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int size;
    cout << "Enter the size of the array : ";
    cin >> size;
    cout << endl;
    int arr[size];
    for (int i = 0; i < size; i++)
    {
        cout << "\nEnter element " << i + 1 << " : ";
        cin >> arr[i];
    }
    int best = 0;
    cout << "\nThe maximum subarray sum : ";
    for (int i = 0; i < size; i++)
    {
        int sum = 0;
        for (int j = i; j < size; j++)
        {
            sum += arr[j];
            best = max(sum, best);
        }
    }
    cout << best << endl;
    getch();
    return 0;
}
```

- O (n^1) Time Complexity:

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-04 11:41:24
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        cout << "\nEnter element " << i + 1 << ": ";
        cin >> arr[i];
    }
    int sum = 0, best = 0;
    cout << "\nThe maximum subarray sum : ";
    for (int i = 0; i < n; i++)
    {
        sum = max(arr[i], arr[i] + sum);
        best = max(sum, best);
    }
    cout << best << endl;
    getch();
    return 0;
}
```

Sorting and Searching

Definition

Arranging the array elements in ascending order is termed as sorting. There are multiple algorithms for sorting, some works in O (n^2), whereas, some works in O $(n \times log_2 n)$ and some even in O (n).

Some common sorting techniques are:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Count Sort

Bubble Sort

The Algorithm for Bubble Sort is quite easy. We keep on choosing two consecutive values and check if they are in the correct order, if yes, then we continue else we swap them. After each iteration, the max valued elements amongst the elements of the modified array is sent at the last.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-04 21:35:27
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;

int main()
```

```
{
    int size;
    cout << "Enter the Size : ";
    cin >> size;
    int arr[size];
    cout << endl;
    for (int i = 0; i < size; i++)
    {
        cout << "\nEnter the element " << i + 1 << " : ";
        cin >> arr[i];
    }
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size - 1 - i ; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    cout << "The Sorted Array : ";
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Selection Sort

The Algorithm for Selection Sort is quite easy. At first, we declare an integer that stores the minimum index and initialize it with the value 0. Then we search for smallest element in the array and after finding that, we just swap the content of the array in minimum index and the min valued element. After that we increment the position of the minimum index by one and continue this process until the array gets sorted.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-04 21:40:12
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int size;
    cout << "Enter the Size : ";
    cin >> size;
    int arr[size];
    cout << endl;
    for (int i = 0; i < size; i++)
    {
        cout << "\nEnter the element " << i + 1 << " : ";
        cin >> arr[i];
    }
    for (int i = 0; i < size - 1; i++)
    {
        int min_index = i;
        for (int j = i + 1; j < size; j++)
        {
            if (arr[min_index] > arr[j])
            {
                min_index = j;
            }
        }
        swap(&arr[min_index], &arr[i]);
    }
    cout << "The Sorted Array : ";
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Insertion Sort

The Algorithm for Insertion Sort is quite easy. Firstly, if the array element is the first element, then it is already sorted. Pick next element, and compare it with all the elements in the sorted subarray. Shift all the elements in the sorted sub-list that is greater than the value to be sorted and insert that value. Repeat this process until the whole array gets sorted.

```
/*
    AUTHOR: BlueKnight
    CREATED: 2021-09-04 21:44:56
*/
#include <bits/stdc++.h>
#include <conio.h>

using namespace std;
```

```
int main()
{
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    for (int i = 1; i < n; i++)
    {
        int pivot = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > pivot)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = pivot;
    }
    for (auto i : arr)
    {
        cout << i << " ";
    }
    getch();
    return 0;
}
```