> To check if a number is prime

// This is checked so that we can skip

// middle five numbers in below loop

if (n % 2 == 0 || n % 3 == 0)

return false;

for (int i = 5; i \* i <= n; i = i + 6)

if (n % i == 0 || n % (i + 2) == 0)

return false;

> To check if a number is palindrome.. get the reverse of that number and compare with the original.

> To find number of bits set:

while(n) {

n &= (n-1);

++count;

}

Complexity of above method: O(nlogn)

Another method in O(1)

int num\_to\_bits[16] = { 0, 1, 1, 2, 1, 2, 2, 3,

1, 2, 2, 3, 2, 3, 3, 4 };

// Get nibble of a given number and map them in the array.

unsigned int countSetBitsRec(unsigned int num) {

int nibble = 0;

if (0 == num)

return num\_to\_bits[0];

// Find last nibble

nibble = num & 0xf;

return num\_to\_bits[nibble] + countSetBitsRec(num >> 4);

}

> To find next divisible number by 2, find if the number has only 1 set bit by using (n !(n&n-1)) , if it is then that is the number, else right shift one-by-one and get the count. Then 1<<count is the number.

> **To find intersection of 2 line segments, we need to find the slopes of 2 lines to find their orientation which can be - clockwise, anti-clockwise or collinear.**

Slope of line segment (p1, p2): σ = (y2 - y1)/(x2 - x1)

Slope of line segment (p2, p3): τ = (y3 - y2)/(x3 - x2)

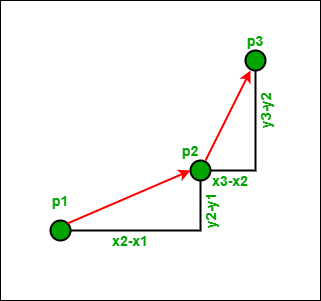
If σ > τ, the orientation is clockwise (right turn)

Using above values of σ and τ, we can conclude that,

the orientation depends on sign of below expression:

(y2 - y1)\*(x3 - x2) - (y3 - y2)\*(x2 - x1)

Above expression is negative when σ < τ, i.e., counterclockwise



// To find orientation of ordered triplet (p1, p2, p3).

// 0 --> Collinear, 1 --> Clockwise, 2 --> Counterclockwise

int orientation(Point p1, Point p2, Point p3) {

// See 10th slides from following link for derivation

// of the formula

int val = (p2.y - p1.y) \* (p3.x - p2.x) - (p2.x - p1.x) \* (p3.y - p2.y);

if (val == 0) return 0; // colinear

return (val > 0)? 1: 2; // clock or counterclock wise

}

> Use above method to check if 2 line segments intersect

> To copy set bits in range [l, r] from y to x.

// get the length of the mask

int maskLength = (1<<(r-l+1)) - 1;

int mask = ((maskLength)<<(l-1)) & y ;

x = x | mask;

> To multiply 2 numbers using bitwise operator

// While second number doesn't become 1

while (b > 0) {

// If second number becomes odd, add the first number to result

if (b & 1)

res = res + a;

// Double the first number and halve the second number

a = a << 1;

b = b >> 1;

}

How does this work?

The value of a\*b is same as (a\*2)\*(b/2) if b is even, otherwise the value is same as ((a\*2)\*(b/2) + a)

> To check if a binary stream is divisible by n or not:

if(number == 1)

rem = (rem\*2 + 1)%n;

else

rem = rem\*2;

// if remainder is 0 , print Yes

> To find count of set bits in O(1)

// Generate the table algorithmically

BitsSetTable256[0] = 0;

for (int i = 0; i < 256; i++)

BitsSetTable256[i] = (i & 1) + BitsSetTable256[i / 2];

int countSetBits(int n) {

return (BitsSetTable256[n & 0xff] +

BitsSetTable256[(n >> 8) & 0xff] +

BitsSetTable256[(n >> 16) & 0xff] +

BitsSetTable256[n >> 24]);

}

> To find 2 missing numbers in an array. Find X^Y, where X is XOR of all elements in given array and Y being XOR of numbers from 1 to n. Now, X^Y will be XOR of those 2 missing numbers. Find the right set bit in that as:

int t = XOR & ~(XOR-1)

Now, traverse through array and create 2 sets(denoted by 2 variables) in which first set contains all elements with bits set and another set containing elements whose bits are not set. Now traverse over array and XOR X and Y based on whether bit is set or not.

int x = 0, y = 0; // Initialize missing numbers

**for (int i = 0; i < n-2; i++) {**

**if (arr[i] & set\_bit\_no)**

**x = x ^ arr[i]; /\*XOR of first set in arr[] \*/**

**else**

**y = y ^ arr[i]; /\*XOR of second set in arr[] \*/**

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

**if (i & set\_bit\_no)**

**x = x ^ i; /\* XOR of first set in arr[] and {1, 2, ...n }\*/**

**else**

**y = y ^ i; /\* XOR of second set in arr[] and {1, 2, ...n } \*/**

**}**

> Fibonacci in logn() time

apart from DP and recursion, another approach to find fibonacci number is:

To find nth fibonacci number, multiply the below matrix itself 'n' number of times

[ 1 1 ]

[ 1 0 ]

**void power(int F[2][2], int n) {**

**if(n == 0 || n == 1) return;**

**int M[2][2] = {{1, 1}, {1, 0}};**

**power(F, n / 2);**

**multiply(F, F);**

**if (n % 2 != 0)**

**multiply(F, M);**

**}**

At last F[0][0] will be the resultant fibonacci number.

Another approach(Using formula) :

In this method we directly implement the formula for nth term in the fibonacci series.

Fn = {[(√5 + 1)/2] ^ n} / √5

> Max area of a quadrilaterals is:

K=sqrt {(s-a)(s-b)(s-c)(s-d)} where s is (a + b + c + d) / 2

> Add 2 numbers using bitwise operator

int bitwiseadd(int x, int y) {

while (y != 0) {

int carry = x & y; // handles 1 + 1 case in which 1 shifts to left position

x = x ^ y; // handles 1 + 0 and 0 + 1 cases

y = carry << 1;

}

return x;

}

// Bitwise operator based function to check divisibility by 9

**bool** isDivBy9(**int** n)

{

// Base cases

**if** (n == 0 || n == 9)

**return** **true**;

**if** (n < 9)

**return** **false**;

// If n is greater than 9, then recur for [floor(n/9) - n%8]

**return** isDivBy9((**int**)(n>>3) - (**int**)(n&7));

}

Inverse a bit string n^(~0)

> To find the position of rightmost bit do 2'complement then & it with original number that will be the answer or (n&(!(n-1)));

> Detect if two integers have opposite signs XOR the 2 numbers if they have opposite signs the result will have lftmost bit as 1 hence <0

>> Find leftmost set bit

public static int highestOneBit(int i) {

i |= (i >> 1);

i |= (i >> 2);

i |= (i >> 4);

i |= (i >> 8);

i |= (i >> 16);

return i - (i >>> 1);

}

> Swap 2 numbers

x = x^y;

y = x^y;

x = x^y;

> Find prime factors of a number

while (n % 2 == 0) {

cout << 2 << " ";

        n = n/2;

}

// n must be odd at this point. So we can skip

// one element (Note i = i +2)

for (int i = 3; i <= sqrt(n); i = i + 2)

{

    // While i divides n, print i and divide n

    while (n % i == 0)

    {

        cout << i << " ";

        n = n/i;

    }

}

To check the divisibility by x

for 2 : check if rightmost bit is 1 or not

for 3 : Total odd number of bits – even number of bits should be divisible by 3. Do it recursively

for 4 : Total number of set bit should be 1 and total 0's from left should be even

for 5 : ?

for 6 : Check if divisible by 2 and 3

for 7 : recursively check (n <<3) – n

for 8 : check if last 3 digits are 0

> Find gcd/HCF of 2 numbers

A simple solution is to find all prime factors of both numbers, then find intersection of all factors present in both numbers. Finally return product of elements in the intersection.

if (x % minimum == 0 && y % minimum == 0)

return minimum;

// Highest no. between 2 and minimum/2 which can divide both the numbers

// is the required HCF

for (int i = minimum / 2; i >= 2; i--)

if (x % i == 0 && y % i == 0)

return i;

int gcd(int a, int b) {

    if (b == 0)

        return a;

    return gcd(b, a % b);

}

gcd of array would be picking each element 1-by-1 and finding gcd of that.

> Given a number ‘n’, we need to find count of binary strings of length ‘n’ which don’t have consecutive 1’s.

int countStrings(int n) {

    int a[n], b[n];

    a[0] = b[0] = 1;

    for (int i = 1; i < n; i++)

    {

        a[i] = a[i-1] + b[i-1];

        b[i] = a[i-1];

    }

    return a[n-1] + b[n-1];

}

We can observe that the count is actually (n+2)’th Fibonacci number for n >= 1. which are 0, 1, 1, 2, 3, 5, 8, 13,...

> Convert a number ‘n’ in base 10 to base x

int ans(int n){

// If the number is less than x return the number as it is.

if(n < x)

return n;

else

return n%2 + 10\*(ans(n/2));

}