**this** is a constant pointer.

insertion sort mini swaps & graph nodes wala method

sorting sometimes reduce time complexity/ or it some solves

in array try to make a mark.

Sometime use two hashMaps.

Inversion-merge sort

String try to reach upto dp

Hcf\*lcm=product of two nos

Hcf using while faster way

a,b,temp while(any greater than 0) temp =other other =one%other one=temp

**power** calculation faster

while (y > 0) { do it by odd even

if ((y & 1)==1)

res =(res \* x);

y = y >> 1;

x =(x\*x) % p;

for **multiply** large nos

static int multiply(int x, int res[], int res\_size)

{

int carry = 0; // Initialize carry

// One by one multiply n with individual

// digits of res[]

for (int i = 0; i < res\_size; i++)

{

int prod = res[i] \* x + carry;

res[i] = prod % 10; // Store last digit of

// 'prod' in res[]

carry = prod/10; // Put rest in carry

}

// Put carry in res and increase result size

while (carry!=0)

{

res[res\_size] = carry % 10;

carry = carry / 10;

res\_size++;

}

return res\_size;

}

}

int foo = Integer.parseInt(string, base);

To get 1st set bit  **a&(a-1) by** and in xor 1st “1” indicate place where bits are diff in two no.

For 2 power a & a-1==0

Use regular expression to extract data

Always take care of -ve number

For repeated

|  |
| --- |
| bitset=0 |
|  | if((bitset & (1<<digit)) >= 1) |
|  | return digit; |
|  | else //if not present then set the corresponding bit |
|  | bitset |= (1<<digit); |

For msb 1 take log…

Log return msb 1

Modulus fails for -ve number.

For product try left and right array.

Array.binarySearch()- not found return -insertion index-1

Starting with 0 means octal -07, 08(gives error)

instanceOf- data type

check for integer overflow, go for divide

not adjacent use inclusive=ex+curr, ex=max(prv inc,ex)

for median use two heaps

//for gcd

static void sieve()

{

    int[] isPrime=new int[MAX+1];

    for (int i = 2; i<= MAX; i++)

    {

        // if prime[i] is not marked before

        if (isPrime[i] == 0)

        {

            // fill vector for every newly

            // encountered prime

            p.add(i);

            // run this loop till square root of MAX,

            // mark the index i \* j as not prime

            for (int j = 2; i \* j<= MAX; j++)

                isPrime[i \* j]= 1;

        }

    }

}

// function to find totient of n

static int phi(int n)

{

    int res = n;

    // this loop runs sqrt(n / ln(n)) times

    for (int i=0; p.get(i)\*p.get(i) <= n; i++)

    {

        if (n % p.get(i)== 0)

        {

            // subtract multiples of p[i] from r

            res -= (res / p.get(i));

            // Remove all occurrences of p[i] in n

            while (n % p.get(i)== 0)

            n /= p.get(i);

        }

    }

    // when n has prime factor greater

    // than sqrt(n)

    if (n > 1)

    res -= (res / n);

    return res;

}