REDEFINING
LIMITATIONS

SOFTWARE INCUBATOR

Presents

SINTAX

A Competitive Programming Workshop

STRING in C++

String is basically the array of characters

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string str;
    cin >> str;
    cout << str;
    return 0;
}
```

INPUT

Software Incubator

OUTPUT

Software

getline()

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    string str;
    getline(cin, str);
    cout << str;
    return 0;
}
```

INPUT

Software Incubator

Output

Software Incubator

getline()

```
#include <bits/stdc++.h>
using namespace std;
int main()  {
    int n;
    cin >> n;
    string str;
    getline(cin, str);
    cout << n << "\n";
    cout << str;
    return 0;
}
```

INPUT

SPACE

```
5 </--------------------/>
Software Incubator
```

Output

```
5
</--------------------/>
```

SPACE

cin.ignore()

```
#include <bits/stdc++.h>
using namespace std;
int main()  {
    int n;
    string str;
    cin >> n;
    cin.ignore(256, '\n');
    getline(cin, str);
    cout << n << "\n";
    cout << str;
    return 0;
}
```

INPUT

SPACE

```
5 </----------------/>
Software Incubator
```

Output

```
5
Software Incubator
```

Problem Statement

Write a program to find the absolute difference of the sum of elements of two different arrays.

(Arrays may have different number of elements)

FUNCTIONS

Why do we need functions?

1. Reduce redundancy.
2. Easy debugging
3. Write once use many times.

Syntax of functions

1. Function definition

```
return_type  name (type1 parameter1, type2 parameter2, ...){
        Code…
        return statement
}
```

2. Calling

```
name(parameter1, parameter2, …)
```

```
#include<bits/stdc++.h> // program to find sum of two numbers
using namespace std;

int sum(int x, int y)  {       // function definition
   int c;
   c=x+y;
   return c;
}

int main() {
     int a,b;
     cin >> a >> b;
     int m = sum(a, b);      // function calling
     cout<<m;
     return 0;
}
```

INPUT

2 3

OUTPUT

5

Tip: Passing array as function parameter

```
void print(int arr[],int n){          //Function Definition
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";
    return;
}

print(arr, n);                        // Function calling
```

Solution

```
#include <iostream>
using namespace std;

long long sum(int arr[], int n){
        long long ans=0;
        for(int i=0;i<n;i++)
                ans+=arr[i];
        return ans;
}
```

```
int main() {
        int n,m;
        cin>>n>>m;
        int arr1[n],arr2[m];
        for(int i=0; i<n; i++)
                cin>>arr1[i];
        for(int i=0; i<m; i++)
                cin>>arr2[i];

        long long sum1=sum(arr1, n);
        long long sum2=sum(arr2, m);
        cout<<abs(sum1-sum2)<<"\n";
        return 0;
}
```

Built-in functions

1. max(a,b)
2. min(a,b)
3. swap(a,b)
4. abs(a)
5. pow(a,b)
6. floor(a)
7. ceil(a)
8. round(a)
9. sqrt(a)

TIME COMPLEXITY

Problem Statement

Write a program to find a pair (a,b) such that their sum is equal to a given integer n. (a, b and n are positive integers.)

Approach 1

```
void solve(int n){
    for(int a=1;a<=n-1;a++)
        for(int b=1;b<=n-1;b++)
            if(a+b==n)
                cout<<a<<" "<<b<<"\n";
    return ;
}
```

Approach 2

```
void solve(int n){

    for(int a=1;a<=n-1;a++)
        cout<<a<<" "<<n-a<<"\n";
    return ;
}
```

Approach 1

```
void solve(int n){
    for(int a=1;a<=n-1;a++)
        for(int
b=1;b<=n-1;b++)
            if(a+b==n)
                cout<<a<<"
"<<b<<"\n";
    return ;
}
```

Approach 2

```
void solve(int n){

for(int a=1;a<=n-1;a++)
        cout<<a<<"    "<<n-a<<"\n";
    return ;
}
```

How can we compare two algorithms?

1. No. of lines? No
2. Execution time? No
3. No. of operations? Yes

EXAMPLES

1. cin>>x; cout<<x; --------- c unit of time.=> constant
2. for(int i=1;i<=10;i++) --- 10c units of time.=> constant
3. for(int i=1;i<=n;i++) --- n units of time. => not constant
4. for (int i = 1; i <= n; i++) {
   for (int j = 1; j <= n; j++) {
   cout<<"SI ";
   }
   }

When i=1, it will run n times.
When i=2, it will run n times.
When i=3, it will run n times and so on.
The total number of times "SI" will run is n+n+n+...+n = n*n. So, the time complexity will be n^2 units of time.

4.
```
for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            cout<<"SI ";
        }
    }
```

Let us see how many times “SI” will be printed.

When i=1, it will run 1 time.
When i=2, it will run 2 times.
When i=3, it will run 3 times and so on.
The total number of times “SI” will run is
$1+2+...+(n-1)+n=n*(n+1)/2=n^2/2+ n/2$.

The time complexity is the computational complexity that describes the amount of time it takes to run an algorithm.

Since there are many ways to solve a problem but we always try to find the most **efficient solution** i.e the one which takes **minimum amount of time**.

1.
```
for (int i = 1; i <= n; i+=c) {
        cout<<"SI ";
}
```
Time complexity - O(n)

2.
```
for (int i = n; i >= 1; i-=c) {
        cout<<"SI ";
}
```
Time complexity - O(n)

3.
```
for (int i = 1; i <= n; i*=2) {
        cout<<"SI ";
}
```
Time complexity - O(logn)

4.
```
for (int i = n; i >= 1; i/=2) {
        cout<<"SI ";
}
```

Time complexity - O(log n)

Find the time complexity for the given code snippet

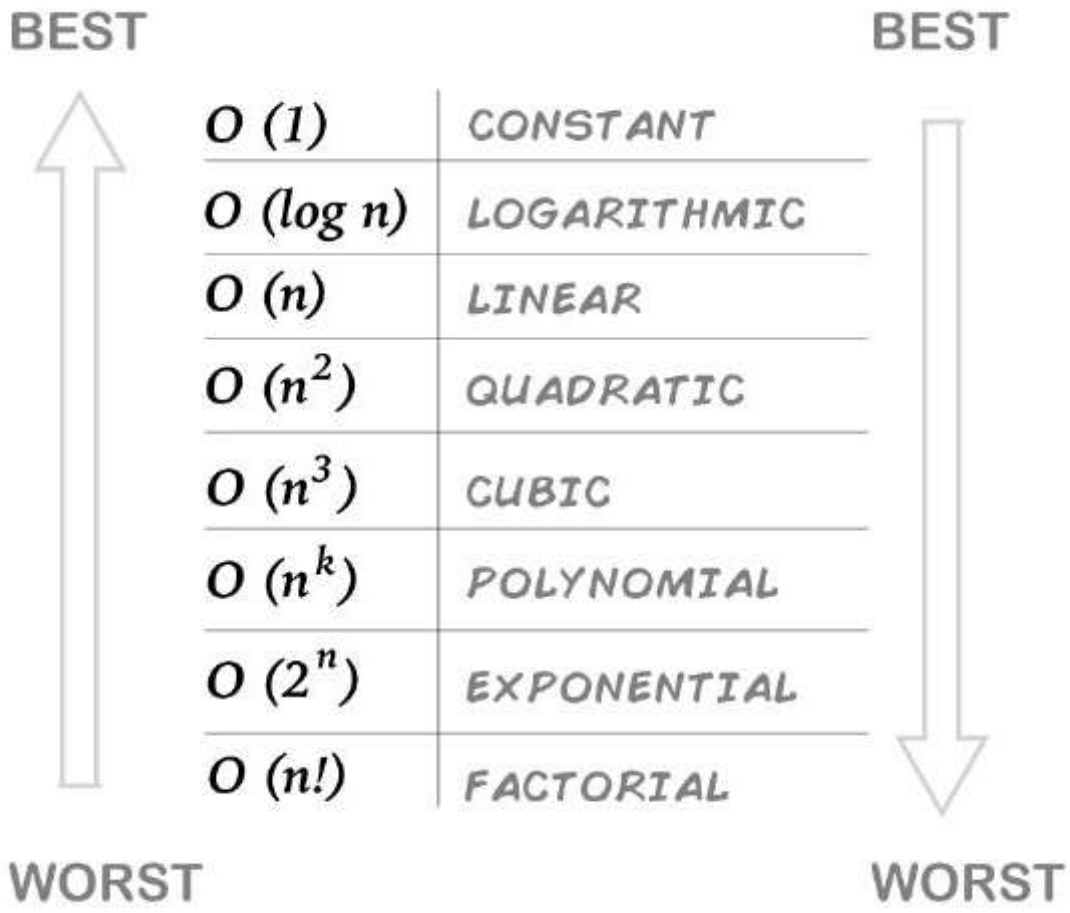
```
int n;
cin >> n;

for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++){
                cout << "SI"<< " ";
        }
}

for(int i=1; i<=n; i++){
        cout << "SI"<< " ";
}

cout << "SI"<< " ";
```

—----------> c

----------> $n*n = n^2$

—----------> n

—----------> c


BEST
BEST
O (1)
CONSTANT
O (log n)
LOGARITHMIC
O (n)
LINEAR
O (n²)
QUADRATIC
O (n³)
CUBIC
O (nᵏ)
POLYNOMIAL
O (2ⁿ)
EXPONENTIAL
O (n!)
FACTORIAL
WORST
WORST

Q.) How many computations a computer can do in 1 second?

Ans. 10^8 approx. computations.

SPACE COMPLEXITY

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main()
{
    ll a;
    cin >> a;
    cout <<a;
    return 0;
}
```

PRE-PROCESSORS

Pre-Processors

Pre-Processors are **directives** which give instruction to the compiler to preprocess information before actual compilation starts.

Eg- #include
#define

Pre-Processors

#include<filename>

The #include preprocessor directive is used to **include header files** in our code.
If included file is not found, compiler renders error.

Eg- #include<bits/stdc++.h>
#include<iostream.h>

Pre-Processors

#define ll long long

It is a macro.
Here we have assigned the value of **long long** to **ll** .
So, whenever the name(ll) is encountered the compiler replaces the name with actual code.
Macros are defined using **#define** directive.

Eg-

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main()
{
    ll a;
    cin >> a;
    cout <<a;
    return 0;
}
```

INPUT

12345678912345678 9

OUTPUT

12345678912345678 9

CODE ORGANISATION

This is my template which I use while coding

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int T;
    cin >> T;
    while(T--) {
    }
       return 0;
}
```

Verdict

AC - Accepted

TLE - Time Limit exceeded

WA - Wrong Answer

RE - Runtime error

Compilation Error

Analysis on the basis of constraint

PREFIX SUM

Q. You are given an array of

N integer. Print the sum of integers in array from given index L to R for some queries Q.

Analysis on the basis of constraint

We must follow up on our approach by analysing the constraint of the problem.

For eg -
Constraint -
$1 <= t <= 10^5$
$1 <= n <= 10^5$

For this it is clear that we cannot run a for loop from 1 to n i.e. our algorithm must have time complexity less than O(n).

Given Array:

0	1	2	3	4
2	5	1	4	6
1	2	3	4	5

Q: L = 1 , R = 4

Ans: 16

Here we are considering that number of queries, Q=1

What if we have multiple range queries on the same array?

Given Array:

0	1	2	3	4
2	5	1	4	6

Q1 : L = 1 , R = 4

Q2 : L = 0 , R = 2

Q3 : L = 2 , R = 4


Yaar aise game me cheating hogi
to mai nahi khelunga

Given Array:

0	1	2	3	4
2	5	1	4	6

Prefix Array:

0	1	2	3	4
2	7	8	12	18

Q1 : L = 1 , R = 4 Ans.1 A[4] - A[0] = 18-2 =16

Q2 : L = 0 , R = 2 Ans.2 A[2] = 8

Q3 : L = 2 , R = 4 Ans.3 A[4] - A[1] = 18-7 =11


Han apun ne cheating kiya
jo ukhadna hai ukhad le

COUNT FACTORS OF A NO.

Factors of 30 = 1 , 2 , 3 , 5 , 6 , 10 , 15 , 30

Approach 1 (Brute Force)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
int main() {
    ll t;  cin >> t;
    while(t--) {
        ll n, i;   cin >> n;
        int count = 0;
        for(i = 1; i <= n; ++i)  {
                if((n % i)== 0)
                        count++;
         }
        cout << count<<'\n';
    }
    return 0;
}
```

Time : 1 sec
$n < 10^8$

Constraints

Subtasks : $1 <= N <= 10^7$ (30 Marks)

Subtasks : $1 <= N <= 10^{12}$ (100 Marks)

Approach 2

If we run loop upto (n/2) terms.

In constraint of $\mathbf{10^{12}}$ we have to run loop = 10^{12} / 2

= $\mathbf{5 * 10^{11}}$ times

Approach 2 (Using n/2)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
int main() {
    ll t;  cin >> t;
    while(t--) {
        ll n, i;   cin >> n;
        int count = 2;
        for(i = 2; i <= n/2; ++i)  {
                if((n % i)== 0)
                        count++;
         }
        cout << count;
    }
    return 0;
}
```

Approach 3

If we run loop upto sqrt(n) terms.

In constraint of $\mathbf{10^{12}}$ we have to run loop = sqrt(10^{12})

= $\mathbf{10^{6}}$ times

We know that factors exist in pairs.

For n=30,

1 * 30

2 * 15

3 * 10

5 * 6

6 * 5

10 * 3

15 * 2

30 * 1

1 * 30

2 * 15

3 * 10

5 * 6

30 * 1

15 * 2

10 * 3

6 * 5

We notice that the factors start repeating after the point sqrt(n).

Thus, we run our loop from 1 to sqrt(n), count the factors and return the count multiplied by 2.

https://ideone.com/l2z1r8

Another way to visualize sqrt(n)

Factors always exist in pairs.

Let's assume that we have two factors x and y of a number n.

So, $n = x * y$

Also, $n = \sqrt{n} * \sqrt{n}$

$x * y = \sqrt{n} * \sqrt{n}$

CASE 1:

If **x < √n** then **y > √n**

CASE 2:

If **x > √n** then **y < √n**

CASE 3:

If **x == √n** then **y== √n**

This implies that there will always be a factor (either x or y) which will be less than or equal to √n

Prime Numbers

Prime Numbers

The numbers having only two factors 1 and the number itself are known as prime numbers and the numbers which are not prime are known as composite numbers.

Eg - 2, 3, 5, 7, 11, etc.

1 is neither prime nor composite

Sieve of Eratosthenes


PROPS CO., LTD.
Aye kya tutul putul

Sieve Of Eratosthenes

The sieve of Eratosthenes is one of the most efficient ways to find all primes smaller than n when n is smaller than 1000000.

Time Complexity = O(n * log(log(n)))

Refer the link to learn more about sieve of eratosthenes:-

https://www.geeksforgeeks.org/sieve-of-eratosthenes/

Sieve Of Eratosthenes

Let n=29
Step 1: Create an integer array of size 30 i.e sieve[30].
Indicies are from 0 to 29.

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29

Sieve Of Eratosthenes

Step 2: Initialize the complete array with 1. 0 and 1 are neither prime nor composite so store 0 at these indices.

0 0	0 1	1 2	1 3	1 4	1 5
1 6	1 7	1 8	1 9	1 10	1 11
1 12	1 13	1 14	1 15	1 16	1 17
1 18	1 19	1 20	1 21	1 22	1 23
1 24	1 25	1 26	1 27	1 28	1 29

Sieve Of Eratosthenes

Step 3: All multiples of two 2 will be non-prime.

0 0	0 1	1 2	1 3	0 4	1 5
0 6	1 7	0 8	1 9	0 10	1 11
0 12	1 13	0 14	1 15	0 16	1 17
0 18	1 19	0 20	1 21	0 22	1 23
0 24	1 25	0 26	1 27	0 28	1 29

Sieve Of Eratosthenes

Step 4: All multiples of two 3 will be non-prime.

0 0	0 1	1 2	1 3	0 4	1 5
0 6	1 7	0 8	0 9	0 10	1 11
0 12	1 13	0 14	0 15	0 16	1 17
0 18	1 19	0 20	0 21	0 22	1 23
0 24	1 25	0 26	0 27	0 28	1 29

Sieve Of Eratosthenes

Step 5:Since 4 is itself non-prime so we need not to do anything.

0 0	0 1	1 2	1 3	0 4	1 5
0 6	1 7	0 8	0 9	0 10	1 11
0 12	1 13	0 14	0 15	0 16	1 17
0 18	1 19	0 20	0 21	0 22	1 23
0 24	1 25	0 26	0 27	0 28	1 29

Sieve Of Eratosthenes

Step 6: All multiples of 5 will be non-prime.

0 0	0 1	1 2	1 3	0 4	1 5
0 6	1 7	0 8	0 9	0 10	1 11
0 12	1 13	0 14	0 15	0 16	1 17
0 18	1 19	0 20	0 21	0 22	1 23
0 24	0 25	0 26	0 27	0 28	1 29

```
int main() {
        int n;
        cin>>n;
        int sieve[n+1];
        memset(sieve, 1, sizeof(sieve));
        sieve[0]=sieve[1]=0;
        for(int i=2;i<=sqrt(n);i++) {
            if(sieve[i]==1) {
                for(int j=i*i;j<=n;j+=i)
                    sieve[j]=0;
            }
         }
        return 0;
}
```

Now all prime numbers will have 1 in the respective index of the array.

PRIME COUNT

Q. Given a number N, find number of primes in the range 1 to N.

Constraint : 1 <= N <= 10^6

Modular Arithmetic

When we divide two integers we will have an equation that looks like the following:

A/B = Q remainder R

A is the dividend B is the divisor

Q is the quotient R is the remainder

Sometimes, we are only interested in what the **remainder** is when we divide A by B

Remainder Theorem

Given, A/B:

A -> Dividend
B -> Divisor
Q -> Quotient
R -> Remainder

Dividend = (Divisor × Quotient) + Remainder

$A = (B \times Q) + R$

SUM OF DIGITS

Q.) Find the sum of digits of a number.

https://ideone.com/zPiBSm

a%b can be written as **a-(a/b)*b .**

Eg -> a = 7, b = 3, then

a%b = 7 - (7/3)*3

= 7 -(2) * 3

= 7 - 6 = 1

QUESTION

Q.) Find the Closest multiple of number.

https://ideone.com/ufLJqJ

Cyclic Property of modulo

0%5=0 5%5=0

1%5=1 6%5=1

2%5=2 7%5=2

3%5=3 8%5=3

4%5=4 9%5=4

a%m=(a+m) % m = (a%m + m%m)

= a%m + 0 = a%m


Isme ek chakkar aur hai

Negative Modulo

-1%5=4

-2%5=3

-3%5=2

-4%5=1

-5%5=0

-6%5=4

-7%5=3

-8%5=2

-9%5=1

-10%5=0

a%m = (a%m + m) % m

Properties of modulo

(a + b) % m = (a%m + b%m)%m

(a - b) % m = (a%m - b%m + m)%m

(a * b) % m = (a%m * b%m)%m

(a / b)%m = ????


+ , - , *
/
Humey apne me mat milayiye hum alag hai

Properties of modulo

(a + b) % m = (a%m + b%m)%m

(a - b) % m = (a%m - b%m + m)%m

(a * b) % m = (a%m * b%m)%m

(a / b)%m = (a%m / b%m)%m

BIG FACTORIAL

Q.) Find the Factorial of a number .

Constraint : 0 <= N <= 10^5