Big MOD:

// Calculate (a^p)%m

long long  bigmod ( long long a, int p, int m )

{

   if ( p == 0 ) return 1; // If power is 0, then a ^ 0 = 1 for any value of a, And 1 Mod m=1 for any value of m, So return 1

   if ( p % 2 ) // If power is odd, Split it : a ^ 5 =( a )\* (a ^ 4) --> left and right child respectively.

   {

       return ( ( a % m ) \* ( bigmod ( a, p - 1, m ) ) ) % m;

   }

   else //If power is even then split it equally and return the result...

   {

       long long c = bigmod ( a, p / 2, m ); /// Both part will have the same result

       return ( (c%m) \* (c%m) ) % m;

   }

}

Prime Factorization:

vector<int>prime; // Contains all the prime numbers from 1 to n

vector<int>factors; /// After factorize() it’ll contain the prime factors of n

void factorize( int n ) {

   int sqrtn = sqrt ( n );

   for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {

       if ( n % prime[i] == 0 ) { // Found a prime that divides n

           while ( n % prime[i] == 0 ) { // Check how many times it divides n

               n /= prime[i];

               factors.push\_back(prime[i]);

           }

           sqrtn = sqrt ( n );

       }

   }

   if ( n != 1 ) {

       factors.push\_back(n); // The only prime factor > sqrt(n)

   }

}

Number of Divisor:

int NOD ( int n ) {

    int sqrtn = sqrt ( n );

    int res = 1;

    for ( int i = 0; i < prime.size() && prime[i] <= sqrtn; i++ ) {

        if ( n % prime[i] == 0 ) {

            int p = 0; /\*Counter for power of prime\*/

            while ( n % prime[i] == 0 ) {

                n /= prime[i];

                p++;

            }

            sqrtn = sqrt ( n );

            p++;/\*Increase it by one at end\*/

            res \*= p; /\*Multiply with answer\*/

        }

    }

    if ( n != 1 ) {

        res \*= 2; /\*Remaining prime has power p^1. So multiply with 2\*/

    }

    return res;

}

Number of Leading Zeros:

const double eps = 1e-9;

/// Find the first K digits of N!

int leadingDigitFact ( int n, int k ) {

    double fact = 0;

    ///Find log(N!)

    for ( int i = 1; i <= n; i++ ) {

        fact += log10 ( i );

    }

    ///Find the value of q

    double q = fact - floor ( fact+eps );

     double B = pow ( 10, q );

    ///Shift decimal point k-1 times

    for ( int i = 0; i < k - 1; i++ ) {

        B \*= 10;

    }

    ///Don't forget to floor it

    return floor(B+eps);

}

Number of Digits in factorial:

int factorialDigit ( int n ) {

    double x = 0;

    for ( int i = 1; i <= n; i++ ) {

        x += log10 ( i );

    }

    int res = ( (int) x ) + 1;

    return res;

}

GCD:

int gcd ( int a, int b ) {

    while ( b ) {

        a = a % b;

        swap ( a, b );

    }

    return a;

}

Prime Sieve:

vector<int> prime; /\*Stores generated primes\*/

char sieve[SIZE]; /\*0 means prime\*/

void primeSieve ( int n ) {

    sieve[0] = sieve[1] = 1; /\*0 and 1 are not prime\*/

    prime.push\_back(2); /\*Only Even Prime\*/

    for ( int i = 4; i <= n; i += 2 ) sieve[i] = 1; /\*Remove multiples of 2\*/

    int sqrtn = sqrt ( n );

    for ( int i = 3; i <= sqrtn; i += 2 ) {

        if ( sieve[i] == 0 ) {

            for ( int j = i \* i; j <= n; j += 2 \* i ) sieve[j] = 1;

        }

    }

    for ( int i = 3; i <= n; i += 2 ) if ( sieve[i] == 0 ) prime.push\_back(i);

}

Taking input in string and transforming it into numbers:

stringstream strm;

string str;

int num, idx = 0;

int arr[100];

strm << str;

while(strm >> num)

{

arr[idx++] = num;

}

Euler’s Totient:

#include <bits/stdc++.h>

using namespace std;

int mark[7000006];

void sieve(int n)

{

memset(mark, 7000000, 0);

int idx = 0;

int limit = sqrt(n);

mark[2] = 0;

mark[1] = 1;

for(int i = 4; i<= n; i+= 2)

{

mark[i] = 1;

}

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

{

if(!mark[i])

{

if(i <= limit)

{

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

{

mark[j] = 1;

}

}

}

}

}

int phi[7000006];

void sievephi(int n)

{

int i, j;

for(i = 1; i <= n; i++) phi[i] = i;

phi[1] = 1;

for(i = 2; i<= n; i++)

{

if(!mark[i])

{

for(int j = i ; j<= n; j += i)

{

mark[j] = 1;

phi[j] = (phi[j] / i) \*(i-1);

}

}

}

}

int main()

{

int n;

cin>>n;

sieve(6000006);

sievephi(n);

for(int i = 1; i<=n; i++) cout<<phi[i]<<" ";

cout<<endl;

}

Kadane’s Algorithm:

int MaxSubArray(int a[], int size)

{

int cur\_max = a[0];

int global\_max = a[0];

for(int i = 1; i< size; i++)

{

cur\_max = max(a[i], a[i] + cur\_max);

global\_max = max(global\_max, cur\_max);

}

return global\_max;

}

String to Upper:

transform(su.begin(), su.end(), su.begin(), ::toupper);

String to Lower:

transform(sl.begin(), sl.end(), sl.begin(), ::tolower);

Maximum Rectangle Area Using Histogram Formula:

int maxRectangleArea(int a[], int size)

{

int top, area = 0, maxArea = -1, i;

stack <int > stck;

for(i = 0; i< size;)

{

if(stck.empty() || a[stck.top()] < a[i])

{

stck.push(i++);

}

else

{

top = stck.top();

stck.pop();

if(stck.empty())

{

area = a[top] \* i;

}

else

{

area = a[top] \* (i - stck.top() - 1);

}

if(area > maxArea)

{

maxArea = area;

}

}

}

while(!stck.empty())

{

top = stck.top();

stck.pop();

if(stck.empty())

{

area = a[top] \* i;

}

else

{

area = a[top] \* (i - stck.top() - 1);

}

if(area > maxArea)

{

maxArea = area;

}

}

return maxArea;

}

Using upperbound lowerbound on arrays:

int prime[100];

int idx = 100;

int L = lower\_bound(prime, prime + (idx), l) - prime;

int U = lower\_bound(prime, prime + (idx), u) - prime;

memset(array, 0/-1, sizeof(array));

file input output

freopen ( "00\_input.txt", "r", stdin );

freopen ( "00\_output.txt", "w", stdout );

pairing two numbers from groups.

long long int ans = 0, tek = 0;

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

{

ans += tek \* arr[i];

tek += arr[i];

}

cout<<ans<<endl;

Use of next\_permutation:

int a[10];

int recur(int total, int pos)

{

if(pos >= 5)

{

if(total == 23)

return 1;

else

return 0;

}

else

{

return recur(total + a[pos], pos+1) ||

recur(total - a[pos], pos+1) ||

recur(total \* a[pos], pos+1);

}

}

int main()

{

while(1)

{

scanf("%d %d %d %d %d", &a[0], &a[1], &a[2], &a[3], &a[4]);

if(a[0] == 0)

break;

sort(a, a + 5);

int ans = recur(a[0], 1);

if(ans)

cout<<"Possible"<<endl;

else

{

if(!ans)

{

while(next\_permutation(a, a + 5))

{

ans = recur(a[0], 1);

if(ans) break;

}

if(ans)

{

cout<<"Possible"<<endl;

}

else cout<<"Impossible"<<endl;

}

}

}

return 0;

}

Dijkstra:

priority\_queue < pair < int , int >, vector < pair <int, int> >, greater < pair <int, int> > > pq;

getline (cin, string);

Formula for Digital Root:

S(X) = (X-1) mod9 + 1;

And for the kth term

(k-1) \* 9 + X

Histogram technique:

int main()

{

int n;

scanf("%d", &n);

int a[n+3];

/\* In a sequence of number, find the immediate larger number on the right from any index\*/

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

{

scanf("%lld", &a[i]);

}

// Histogram technique

/\* 6

10 3 7 4 12 2

ans : 3 + 0 + 1 + 0 + 1 + 0 -> 5

\*/

stack < int > stk;

stk.push(n);

long long int ans = 0;

for(int i = n-1; i >= 0 ; i--)

{

while(!stk.empty() && stk.top() != n && a[i] > a[stk.top()]) stk.pop();

ans += (stk.top() - i - 1);

stk.push(i);

}

printf("%lld\n", ans);

return 0;

}