To find no. of ‘trailing’ zeros in N!

Sol: we need to count no. of (2\*5) in prime factorization of N!

No of 2s > no. of 5s

Therefore, we only need to count no. of 5s in prime factorization of N!

Formula:

No. of trailing 0s in N! = Count of 5s in prime factors of n!

**= floor(n/5) + floor(n/25) + floor(n/125) + ....**

Q). Find LCM of nos. in an array.

Long long LCM = arr[0];

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

**LCM = (LCM \* arr[i]) / gcd(LCM, arr[i]);**

**return ans;**

**MODULO ARITHMETICS:**

For modulo in addition, substraction & multiplication we have:

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

(a – b) % m = (a%m - b%m) %m

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

Above does NOT apply for division.

**Temperature Conversion Formula:**

F = C\*9/5 + 32;

**To find no. of digits in a no.**

Digits = floor( log10(N) + 1 );

**Finding no. of digits in a factorial:**

Digits = floor( log10( N! ) ) + 1;

Now since, log(a\*b) = log(a) + log(b)

**Double digits = 0;** // float maybe too small

**for(int i=2 ; i<=N ; i++)**

**digits += log10(i);** // var digits has to be **double** or **long double**

// **int wont work** since will round every log10(i)

Int result = floor(digits) + 1;

-------------------------------------------------------------------------------

//For much larger values, Kamenetsky’s formula

digits = **N \* log10(( N/ M\_E)) + log10(2\*M\_PI\*N)/2** ;

int result = floor(digits + 1);

**Rotating Arrays:**

It is same as cyclic shift in array.

Counter-clockwise shift = Left Cyclic Shift

Clockwise shift = Right cyclic shift

**Prefix Sum Arrays:**

**Ans:** Its an array of same size as arr[], in which every element at ith position, is the sum of elements arr[0] to arr[i].

i.e.

prefixSum[i] = arr[0] + arr[1] + arr[2] + … + arr[i];

for all 0 <= i <= N.

**Maximum Continuous Subarray Sum Problem:**

To find the max. sum of continuous subarray of a given array:

<https://practice.geeksforgeeks.org/problems/kadanes-algorithm-1587115620/0/?track=dsa-workshop-1-arrays&batchId=308>

Sol: Kadane’s Algo: O(n)

Amazing algorithm, we need to maintain max\_sum and current\_sum,

1. Whenever current\_sum < 0,

current\_sum = 0

Mean we will discard the previous elements sum.

1. Whenever current\_sum > max\_sum ,

max\_sum <- current\_sum.