Class-2 Logic Building And Mathematical Foundation

**RECAP OF LAST CLASS**

1. Array vs Dynamic Array
2. Matrix vs List of List
3. Time and Space Complexity - At Function Level
4. Every problem we solve will be around function

# Time Complexity of A Function

Number of iterations a function is performing.

If No loop -1 Iteration.

# Space Complexity of A Function

Max number of variables, required by a function at any time, (excluding input/output) If we use an array of size N in a function - Count it as N variables.

**[Time Limit]** Maximum Number of Iterations allowed: 108

**[Memory Limit]** Maximum Number of variables allowed: 106

# How to calculate Time and Space Complexity

1. A function without any loops. Time Complexity : 1
2. One single loop.

**[PROBLEM]** Given N, find count of multiples of 3, 7 in range [1, N]

1 <= N <= 109

N = 10

ANS= 4

Explanation 3, 6, 7, 9

int countMultiples(int n) {

int c = 0;

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

if (i % 3 == 0 || i % 7 == 0)

c++;

return c;

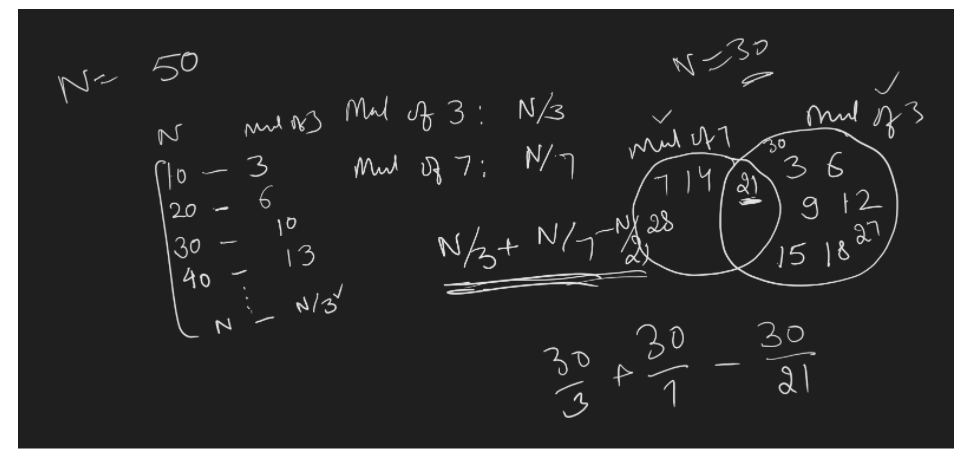
}

21, 42

Time Complexity: n

Space Complexity: 1 (c is output variable so can be left out)

TLE



n(A U B) = n(A) + n(B) - n(A ∩ B)

// Use Integral Division

int countMultiplesOf3And7(int n) {

return n/3 + n/7 - n/21;

}

*Time* Complexity : 1

# Time Complexity of Multiple Sequential Loops (Adds Up)

**Count Worst Case number of steps (that is why we have 1 extra in some cases)**

Time Complexity: n + (n-1) + n/2 + (1 + log2n) + (1 + log3n) + (1 + log2n) + sqrt(n)

Ignore Constants: n + n + n + **logn + logn + logn** + **sqrt(n)**

Ignore Less Significant Values : 3n

Ignore Constants: n

Big-Oh: **O(n)**

int beingZero(int n) {

int s = 0;

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

s = s + i;

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

s = s + i;

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

s = s + i;

for (int i = 1; i <= n; i = i \* 2)

s = s + i;

for (int i = 1; i <= n; i = i \* 3)

s = s + i;

for (int i = n; i >= 1; i = i / 2)

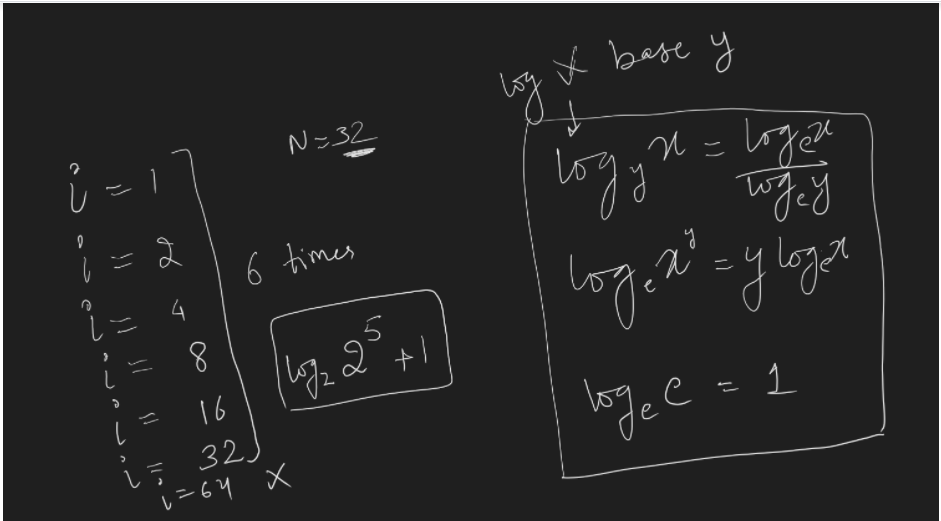
s = s + i;

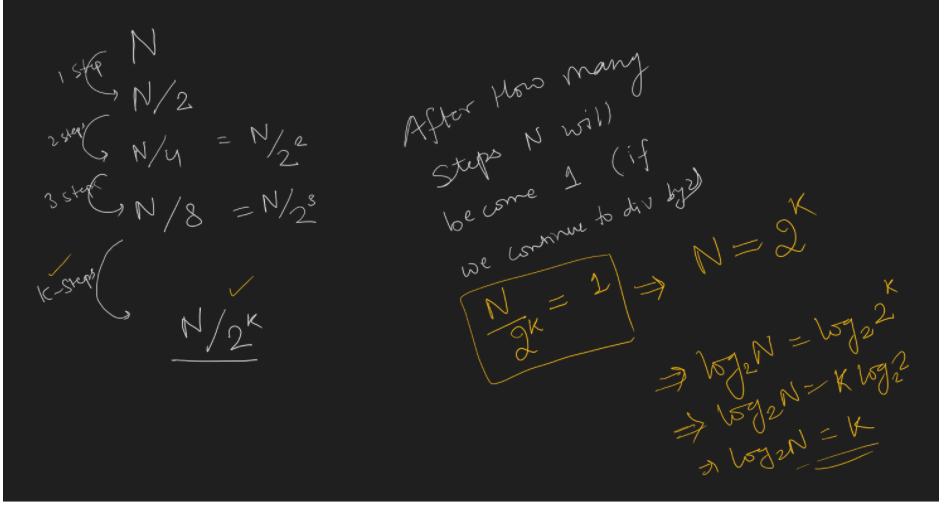
for (int i = 1; i \* i <= n; i++)

s = s + i;

return s;

}





## Nested Loops

1. Independent
   1. Time Complexity: multiplying iterations of outer and inner
2. Dependent

**[PROBLEM]** Given an array of size N, fill every index of array A[i] with the product of all other  
values in the array except i.

**CONSTRAINTS**

1 <= A[i] <= 100  
1 <= N <= 106

INPUT

A = {3,2, 6, 9, 4, 1}

OUTPUT

A = {432, 648, 216, 144, 324, 1296}

**BRUTE FORCE:** Least optimal way of solving the problem.

index 0: a[1] \* a[2] \* a[3] \* a[4] \* a[5]

index 1 : a[0] \* a[2] \* a[3] \* a[4] \* a[5]

index 2: a[0] \* a[1] \* a[3] \* a[4] \* a[5]

…

index i: **(a[0]\*...\*a[i-1]) \* (a[i+1]\* .... \*a[n-1])**

*vector*<int> fillAndReturnNewArray(*vector*<int> &a) {

}

// Pseudo Code

// Time Complexity: n (if declaration of list of size n takes n time) + n \* n

*list*<int> fillAndReturnNewList(*list*<int> a)

{

*list*<int> b(a.*size*()); // n

for (int i = 0; i < a.*size*(); i++) { //filling index i

int p = 1;

for (int j = 0; j < a.*size*(); j++) {

if (i != j)

P = P \* a[j];

}

b[i] = p;

}

return b;

}

*list*<int> fillAndReturnNewList(*list*<int> a)

{

*list*<int> b(a.*size*());

for (int i = 0; ¡ < a.*size*(); i++) { // filling index i

int left = 1 ;

for (int j = i - 1; j >= 0; j - ) // i

left = left \* a[j];

int right = 1;

for (int j = i + 1; j < a.*size*(); j++) // n - i - 1 or n - (i + 1)

right = right \* a[j];

b[i] = left \* right;

}

return b;

}

Time Complexity: n\*(i + n- i-1) = **n \* (n -1) = O(n2)**

## *Time* Complexity of Dependent Loop

int beingZero(int n)

{

int k = 1;

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

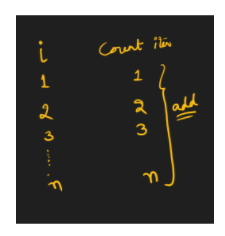
for (int j = 1; j <= i; j++)

*print*(k++);

}

Time Complexity: n\*(n+1)/2 = O(n2)

When we have nested dependent loops, count iterations for every outer loop index and add all of them to get time complexity.



int beingZero(int n) {

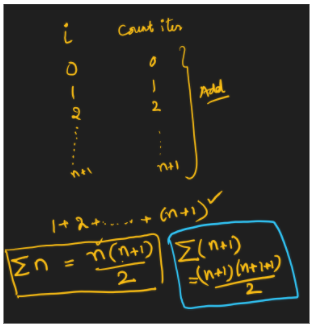
int k = 1;

for (int i = 0; i <= n + 1; i++)

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

*print*(k++);

}



Time Complexity: **(n+1)(n+2)/2 = O(n\*n) = O(n2)**

**Big-Oh Notation**

1. Ignore any constants if they are in Mul, Div or Addition
2. Ignore less significant terms if they are in ADDITION or SUB.

Purpose of Big-Oh notation is to **CATEGORIZE ALGORITHMS.**

When we say optimize, we look for jumping from one category to another.

We will always do two types of analysis

1. Practical (Detailed Analysis)
2. Asymptotic (Big-Oh)

Iterative Code Time/Space Complexity Analysis is DONE.

EXERCISE

Arrange these complexities in BEST to WORST ORDER (Consider values of N >= 1000)

N2

NlogN

2N

N/2

N3

N

logN

sqrt(N)

1

N2logN

N4

N!

NN

loglogN

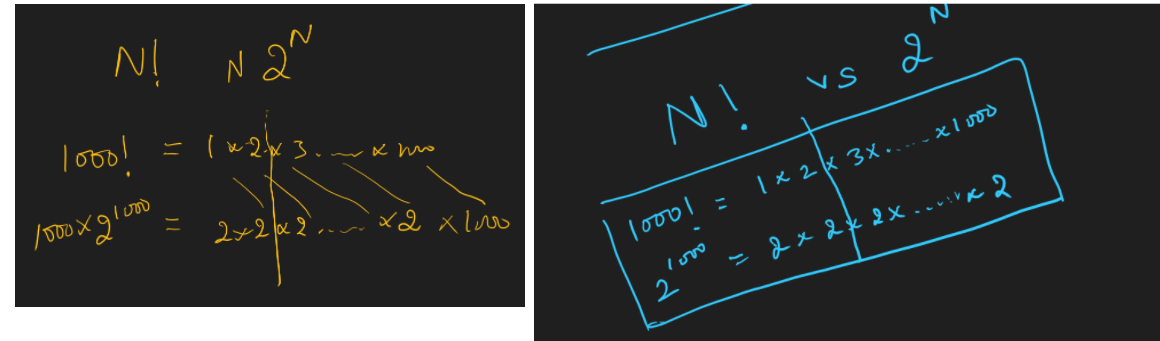
N\*2N

N\*N!

2N

Answer:

1, loglogN, logN, sqrt(N), N/2, N, 2N, NlogN, N2, N2logN, N3, N4, 2N, N2N, N!, N\*N!, NN



2D MATRIX RxC

[PROBLEMS] Print every element of the 2D matrix of RxC.

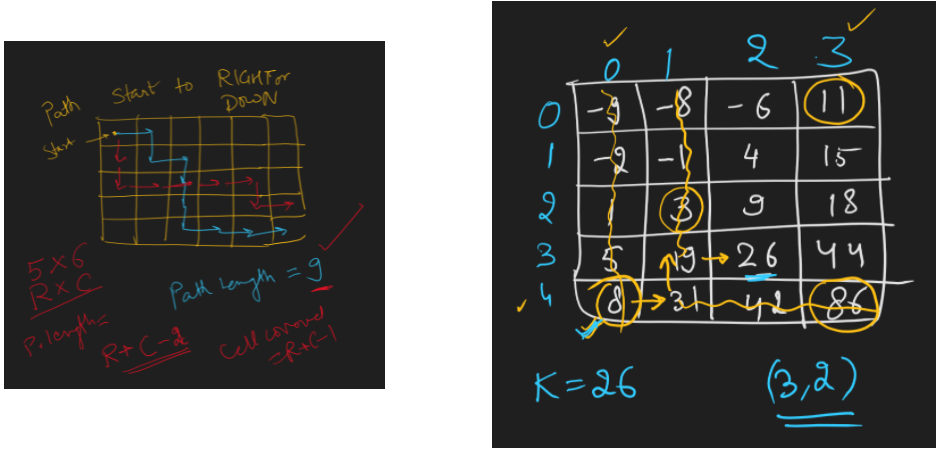
Time Complexity: RxC

[PROBLEM] Given a matrix, ROW and COL are sorted. Find and return index of given element K in the matrix. Assume, matrix has distinct elements.

FUNCTION

**list<int>** findlndexOfValue(list< list<int> > &m, **int k){**

}



Start from LAST ROW, FIRST COL [r-1, 0] cell

i=r-1, j=0

Now compare K with curCell

If curCell < K

Move to Right

else if curCell > K

Move to Up

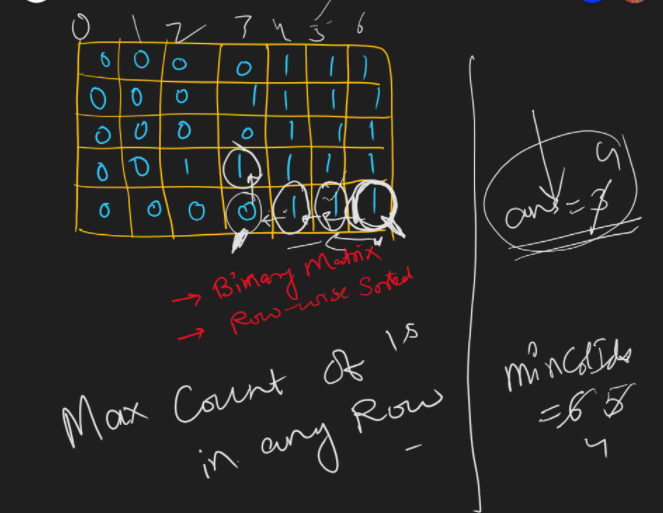
else curCell == K

We are done, return

<https://leetcode.com/problems/search-a-2d-matrix/>

<https://leetcode.com/problems/search-a-2d-matrix-ii/>

[PROBLEM] Given binary matrix. Count and return max ones in any ROW.



**APPROACH 1:**

**TC: r\*c**

for each row

Count 1s in the row, by inc count when we see a 1 maxCount = max(count, maxCount)

**APPROACH 2:**

TC: r\*(position of last 1 in row) = Worst case it will still be r\*c

for each row

Find 1st 1 in the row and using the count number of 1s without continuing the loop maxCount = max(c - firstOnelndexInRow, maxCount)

**APPROACH 3:** (Go column wise)

TC: (col number where we find a 1) \* r = Worst case it will still be r\*c

for each col

if we find 1 in any row

return the answer

Binary Search on Each Row : r \* logC

Bitwise Operators Solution: XOR or bitwise properties

**Approach 4 TC:**

Start from the bottom right cell,

if we see a 1

Move left

Else

Move up

Figure out how to get the count of 1s from the col we exit matrix from.

<https://practice.geeksforgeeks.org/problems/row-with-max-1s0023/1>

# What steps to take with different verdicts?

1. Compilation Error
   1. Read the error message, Find out line number, fix the line.
2. Runtime Error
   1. Index out of bounds, Divide by 0
3. Wrong Answer
   1. Check Constraints
   2. Try with LEAST, MAX and MID value for given inputs
   3. Think of corner test cases based on given constraints
4. Time Limit Exceeded
   1. Calculate Time Complexity of Code and check if it beats given constraints
   2. Check if there is any infinite loop
   3. Optimize your algorithm
   4. Use FASTIO (Input Reading and Output Printing take significant amount of time)
5. Presentation Error
   1. Output is correct but not exactly matching SPACES or NEWLINES with expected output