Lecture 23

2D Array or Matrix

① Creation of 2D Array

1) Inside memory 2D Array is stored in form of linear array only.

2) `int arr[3][3]; // make a matrix of 3x3`

3) which can be accessed using a formula like `Cxi + j`
   C → no. of col
   i → row , j → col index

4) Input in 2D Array
   `cin>> arr[i][j];`

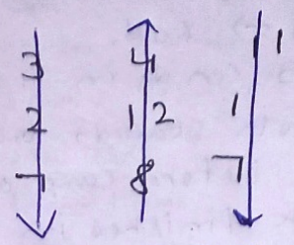
5) Output

```
for (i=0 → i<arr[i].size)
  { for (j=0 → arr[i].size)
    {
      cout<<arr[i][j];
    }
  }
```

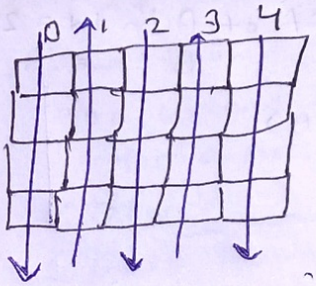
6) While passing 2D Array as a parameter, we need to give the column size mandatorily.

So that compiler can pre-calculate memory addresses of individual elements

7) Wave Print



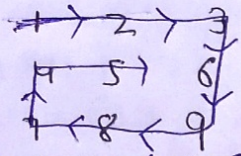
O/p: {3, 7, 7, 8, 12, 4, 1, 1, 7}



For every odd index we go up (bottom to top)
For every even index we go Top to bottom

So we can apply Top to bott & bott to top using for loop

8) Spirial Print



O/p={1, 2, 3, 6, 9, 8, 7, 4, 5}

→ Print starting Row
→ Print ending col
→ Print ending Row
→ starting col Print
(after printing each step move row & col)

row = matrix.size(); //size
col = matrix[0].size(); //size

9) Rotate Matrix by 90°

```
1 2 3        7 4 1
4 5 6   =>   8 5 2
7 8 9        9 6 3
```

```
(0,0) (0,1) (0,2)
  1     2     3
(1,0) (1,1) (1,2)
  4     5     6
(2,0) (2,1) (2,2)
  7     8     9
```

↓

```
(0,0) (0,1) (0,2)
  7     4     1
(1,0) (1,1) (1,2)
  8     5     2
(2,0) (2,1) (2,2)
  9     6     3
```

(2,0) → (0,0)
(1,0) → (0,1)
(0,0) → (0,2)
(2,1) → (1,0)

Swap (mat[i][j], mat[j][i])

```
1 4 7
2 5 8
3 6 9
```

reverse(mat[i].begin(), mat[i].end()) ↓ Reverse only row

```
7 4 1
8 5 2
9 6 3
```

10) Binary Search in 2D Array

```
 1  3  5  7
10 11 16 20
23 30 34 60
```

Condⁿ given: whole matrix is in sorted order, we need to find the target.

it looks like a linear Sorted array only, so we can apply binary search in 1D array only

start = 0;
end = row * col - 1;
mid = start + (end - start)/2

Case 1: arr[mid] == target
return 1

Case 2: arr[mid] < target
start = mid + 1

Case 3: arr[mid] > target
end = mid - 1

How to find mid element in 2D array?
we use column index for it
mid row = mid / col;
mid col = mid % col;
so element = matrix[mid/col][mid%col];

TC = O(log row * col)

11) Search in a 2D Matrix - II

Condn given: row each element is sorted in Row and Col manner both

1	4	7	11	15
2	[5]	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

target = 5

o/p = true

1) Let say if our target > 9 then we can say all values before 9 are worthless then we do row++

2) if our target < 9, we can say all values after 9 are worthless, so do col--

3) if we start our soln from last col and 0th row, then we can move accordingly.

```
int row = mat.size()
int col = mat[0].size()
int start = 0
int end = col - 1;
while (start < row && end >= 0)
{ int element = mat[start][end];
  if (ele == target)
      return 1;
  if (ele < target)
      start++;
  else
  { end--;
  }
  return 0;
}
```

— × — × — × ——

Lecture 24

Basic Maths for DSA

1) Sieve of Eratosthenes

Find prime no. in optimise way.

Let say. n = 40

find all prime no. less than 40

~~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) ~~30~~
(31) ~~32~~ ~~33~~ ~~34~~ ~~35~~ ~~36~~ (37) ~~38~~ 39 ~~40~~

① mark every no. as prime no. initially

② we know prime no. do not come in table of other numbers.

③ we reach i=2, we mark all no. which comes in table of 2 as non-prime, and keep 2 as prime.

④ we come at 3, we keep 3 as prime but mark all in table of 3 as non-prime

⑤ Same we go to 7 and do same

⑥ we do this till < 40

⑤ we mark prime[0] = prime[1] = false initially.

Q find prime no. till n in optimise way

```
{
  vector<bool> prime(n+1, 1);
  int cnt = 0;
  prime[0] = prime[1] = false;
  for(i=2; i<n; i++)
  {  if(prime[i])
     {  cnt++;
        for(j=2*i; j<n; j=j+i)
        {  prime[j] = false;
        }
     }
  }
  return cnt;
}
```

// TC : $\left(\frac{n}{2} + \frac{n}{3} + \frac{n}{7} \cdots \frac{n}{11} + \cdots\right)$

$n\left(\frac{1}{2} + \frac{1}{3} \cdots\cdots \frac{1}{11} + \cdots\right)$ ← HP of prime no.

$TC = O(n * \log(\log n))$

2) Segmented Sieve of Eromothonas

It is a space optimised version of simple sieve :

1) n is very large
2) we have a range [low, high] as i/p and high is very large.

Let say low = 6, High = 10

1) Generate all prime no. from 0 to floor($\sqrt{H}$).

floor($\sqrt{10}$) = 3

prime[] = {2, 3}

we do this using normal Sieve algorithm, TC = O(n log(log n))

SC = $O(\sqrt{H})$

2) Create an array of size (bool) (H-L+1), i.e (10-6+1) = 5 (initially all true)

// TC = O(H-L+1)

we overall ~~H~~ indexes like for 0 to 10, we will fill values in array of size 5 only

T	F	T	F	F
0	1	2	3	4
5	6	7	8	9

3) we mark multiple of 2, 3 as False in above array.

F	T	F	F	F

// TC = $O(H \log\log H)$

// SC = $O(\sqrt{H}) + O(H-L+1)$

(TC remains same but SC reduces)

3) GCD / HCF

Factor which is maximum and can Divide both the nos.

$gcd(a,b) = gcd(a-b,b)$

$\Downarrow$

$gcd(a\%b, b)$

$gcd(72,24) = gcd(48,24)$

$\downarrow$

$gcd(24,24)$

$\downarrow$

$gcd(0,24)$

24 is the answer as when one becomes 0, other becomes our answer

```
{   if (a==0)
        return b;
    if (b==0)
        return a;
    while (a!=b)
    { if (a>b)
        { a = a-b; }
      else
        { b = b-a; }
    }
    return a;
}
```

4) Relation b/w LCM and GCD

$LCM(a,b) * gcd(a,b) = a*b$

5) Modulo Arithmatics.

$a\%n \rightarrow [0 \rightarrow (n-1)]$

Print ans modulo 10^9+7?

It is done to bring ans within the range.

Properties of mod

1) $(a+b)\%m = a\%m + b\%m$

2) $a\%m - b\%m = (a-b)\%m$

3) $((a\%m)\%m)\%m = a\%m$

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

6) Fast Exponentiation

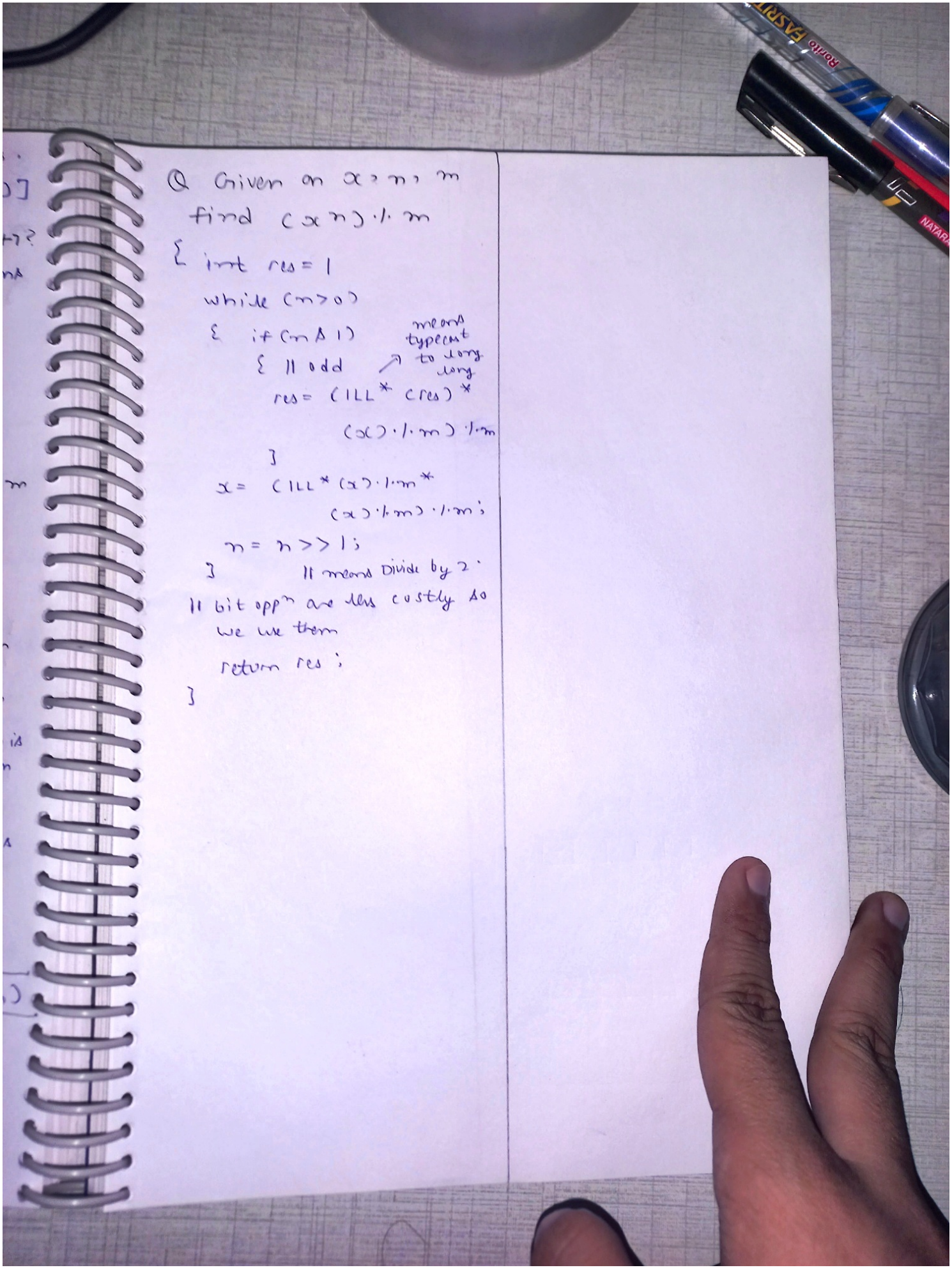
$a^b \rightarrow (a^{b/2})^2$ → if b is even

$(a^{b/2})^2 \times a$ → if b is odd

example: $2^{10} \rightarrow (2^5)^2$

$2^{11} \rightarrow (2^5)^2 \times 2$

TC of Fast expo: $O(\log n)$


Q Given an x, n, m
find (x^n) % m
{ int res = 1
while (n > 0)
{ if (n & 1)
{ // odd
res = (1LL * (res) *
(x) % m) % m
}
x = (1LL * (x) % m *
(x) % m) % m;
n = n >> 1;
} // means Divide by 2.
// bit oppn are less costly so
we use them
return res;
}
means typecast to long long