Day-3

1) Search in a 2D Matrix :-

LC :- each row is sorted and the last ele. in cur. row is smaller than first ele. in next row

GFG :- Each row is sorted, each col. is sorted.

① **Brute force :-** Do a linear traversal and search. TC :- $O(n^2)$ (n^2 = m*n)

② **Optimal :-** On every row Do binary search. TC :- $O(n \log(m))$

③ **optimised :-** target = 25

10	20	30	40
11	21	36	43
25	29	39	50
50	60	70	80

while (j >= 0 and i < n)

- → put pointer in top-right corner i.e. i = 0, j = m-1
- → if (target < m[i][j]) j--;
- → else if (target > m[i][j]) i++;
- → else - return true.

If not found then at last return false.

TC :- $O(m+n)$

ii) Pow(x,n) eg:- x = 2.00000, n = 10 o/p:- 1024.00000
x = 2.10000, n = 3, o/p:- 9.26100

① Brute force:- multiply x, n times, if n is +ve return ans, if n is -ve, then their is an edge case, you need to store in long long, for -ve → $\frac{1}{ans}$.

TC:- O(n)
SC:- O(1)

② Optimised:- x^n.

eg:- $2^{10} = (2\times2)^5 = (4)^5 = 1024$

$4^5 = 4 \times 4^4 = 256$

$4^4 = (4\times4)^2 = 16^2 = 256$

$16^2 = (16\times16)^1 = 256^1$

$256^1 = 256 \times (256)^0$

if (n is even) $x = x * x$, $n = n/2$

else → $ans = ans * x$; $n = n-1$

TC:- O(log n)
SC:- O(1)

code

```
double pow(double x, int n) {
    double ans = 1.0;
    long long nn = n;
    if (nn < 0) nn = nn * (-1);
    while (nn) {
        if (nn % 2) {
            ans = ans * x;
            nn = nn - 1;
        } else {
            x = x * x;
            nn = nn / 2;
        }
    }
    if (n < 0) ans = (double)(1.0) / (double) ans;
    return ans;
}
```

(iii) **Majority element** (> n/2 times), To find majority ele. in array, where if an ele. appears more than $\lfloor n/2 \rfloor$ times, then it is majority ele. eg:- [3,2,3] o/p:- 3

① **Brute force:** – for every ele. in array, check its occurence in array and if (occurence > n/2) then return that ele.

TC:- O(n²)
SC:- O(1)

② **Using Hashmap:** - Store ele's in map and if for any key value > n/2, then that will be your answer,

TC:- ~~O(n)~~ O(n log n) (map) / O(n) (if frequency array)
SC:- O(n).

③ **Moore Voting Algo:-**

7 7 5 7 5 1 | 5 7 | 5 5 7 7 | 5 5 5 5

cnt = 0 1 2 1 2 1 0
1 0
1 2 1 0 , 1 2 3 4
ele = 7 7 5 5 5

```
if (cnt == 0)
    ele = a[i];
if (ele == a[i])
    cnt++;
else
    cnt--;
```

Intution Behind Algo:- Since majority ele. > n/2 times.

7 7 5 7 5 1 | 5 7 | 5 5 7 7 | 5 5 5 5

7 → 3 ← maj.
5 → 2 } ← min.
1 → 1

maj == min, so majority is cancelled by minority ele.

Also, it is fixed that there is a majority ele.

→ So, if majority not found in left side, then last ele. will be majority

→ If majority appears on left prefix, then it count at max can be (len of leftprefix/2). Since it has to appear more than n/2 times, therefore it is bound to appear as a majority in the last suffix.

#code

```
{ count=0, candidate=0;
  for(int num : nums){
      if (count==0)
          candidate = num;
      if (num==candidate) count++;
      else count--;
  }
  return candidate;
}
```

TC:- O(n)
SC:- O(1)

(iv) **Majority element (> n/3 times)**, find all ele. that appears more than $[n/3]$ times. eg:- [3, 2, 3], o/p:- 3
eg:- [1, 1, 1, 3, 3, 2, 2, 2] o/p:- [1, 2]

① **Brute force**:- for every ele. check it's occurence, if it is ($> \frac{n}{3}$) times then add it into your ans vector.
TC:- $O(n^2)$
SC:- $O(1)$

② **using Hashmap**:- Put ele's in map, and then iterate on map, and for every key check it's value, if greater than $n/3$ times, then put in DS,
TC:- $O(n)$ or $O(n \log n)$
SC:- $O(n)$

③ **Boyer Moore Voting Algo**:- (At max you can have 2 majority ele, and at min 0)

num = [1|1|1|3|3|2|2|2]

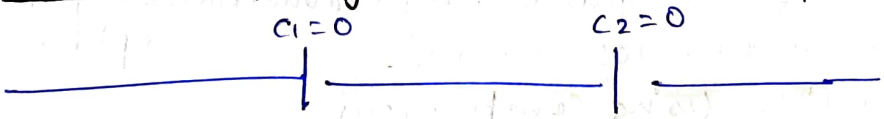
num1 = ~~1~~
num2 = ~~3~~ 2
c1 = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~2~~ 1
c2 = ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ 1

So,
num1 = 1, num2 = 2

```
for(ele : nums){
    if (ele == num1) c1++
    else if (ele == num2) c2++
    else if (c1 == 0){
        num1 = ele;
        c1 = 1;
    }else if (c2 == 0){
        num2 = ele;
        c2 = 1;
    }
    else{
        c1--;
        c2--;
    }
```

→ So, now for these two value check if there occurence in any $> \frac{n}{3}$ times or not and accordingly add on ans DS.

Intution Behind Algo:- This extension of previous question.



whenever $c_1 = 0 \rightsquigarrow$ num1 = ele, when, $c_2 = 0 \rightsquigarrow$ num2 = ele.

if (maj == min)
↓
ele's updated, if has to appear more than $\frac{n}{3}$ times.

#. code (on LC)

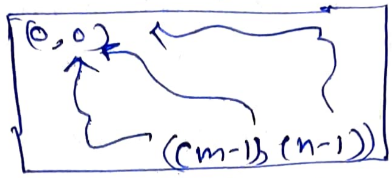
TC:- $O(n) + O(n) \cong O(n)$
SC:- $O(1)$

(Coodle · 29)

(V) Grid unique paths

[· Given m×n, to go from (0,0) → (m-1, n-1), find total unique paths. (we can only move right or down)

① **Brute force** (i.e Recursion), Instead of going (0,0) → (m-1, n-1), go opposite. (Top-down)



{ ① ↑ up
{ ② ← left

Recursion:- Trying out all possible ways.

```
f(i, j) {
    if (i == 0 and j == 0)
        return 1;
    if (i < 0 or j < 0)
        return 0;
    up = f(i-1, j);
    left = f(i, j-1);
    return up + left;
}
```

TC:- $O(2^{m \times n})$ → exponential
SC:- O(path length) + $O(2^{m \times n})$ → recursive stack space

O(path length) → O((m-1) + (n-1)).

② **optimised** (i.e **Memoization**) (overlapping subproblems)

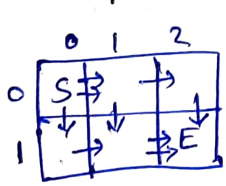
(v<v<int>> dp(m, v<int>(n,-1));) → dp array

```
f(i, j) {
    if (i == 0 and j == 0) return 1;
    if (i < 0 or j < 0) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    up = f(i-1, j);
    left = f(i, j-1);
    return dp[i][j] = up + left;
}
```

TC:- O(m×n)
SC:- O(path len) + O(m×n) → recursive stack space

These code can be further optimised by (Tabulation or space optimization)
But all of these will have TC:- O(m×n).

③ **Most optimised sol^n**:- Using combinations



{ RRD
{ DRR
{ RDR

Observation ①:- We always took 3 steps to reach S → E, i.e, (in general)
((m-1) + (n-1)) = [m + n - 2] steps

obs. ②:- (m-1) right step, (n-1) down step

→ 2 + 3 - 2 = [3] steps.

__ __ __ } to fill these 3 places, either we choose from 2 right steps, $^3C_2$ or 1 down step, $^3C_1$

$^3C_2 = 3$ Or $^3C_1 = 3$

R	R	B	} 3ways.
R	B	R	
B	R	R	

B	R	R	} → 3ways.
R	B	R	
R	R	B	

⇒ for $(m+n-2)$ directions either place $m-1$ right paths $\boxed{^{m+n-2}C_{m-1}}$

or

either place $n-1$ left paths. $\boxed{^{m+n-2}C_{n-1}}$

any of above will give right answer.

and shortcut to find ^nC_r, ⟶ eg:- $^{10}C_3 = \frac{8 \times 9 \times 10}{3 \times 2 \times 1}$

TC :- O(m-1) or O(n-1) ⟶ for $\boxed{ans = {}^{m+n-2}C_{n-1}}$

SC :- O(1)

for $\boxed{ans = {}^{m+n-2}C_{m-1}}$

#code

```
uniquepaths(int m, int n){
    int N = n+m-2;
    int r = m-1;
    double res = 1;
    for(int i=1; i<=r; i++)
        res = res * (N-r+i)/i;
    return (int)res;
}
```

vi) Reverse Pairs :-

$i<j$ and $nums[i] > 2*nums[j]$

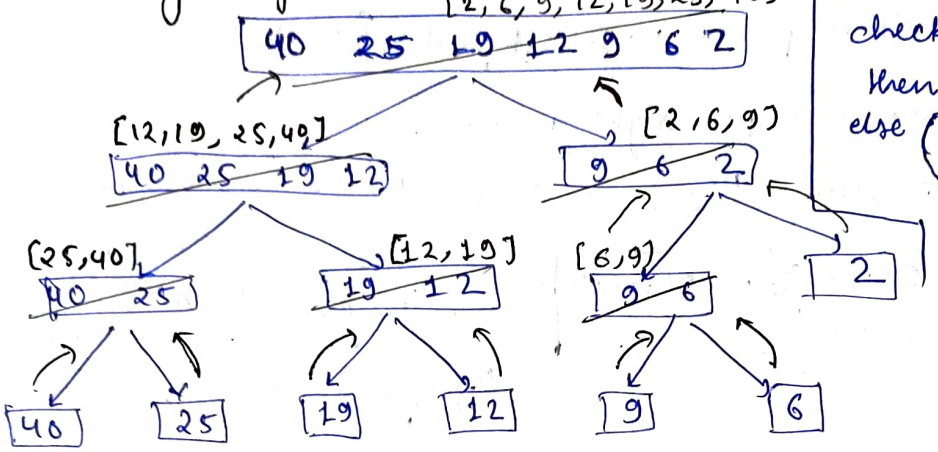
count all such pairs. eg:- [1,3,2,3,1], o/p:- 2 {(3,1),(3,1)}

① **Brute force** :- for every i in nums, check of every j, such that $i<j$ and satisfies the condition $nums[i] > 2*nums[j]$

TC :- $O(n^2)$
SC :- $O(1)$

(Here, we are considering all the pairs)

② **Using Merge Sort** :-

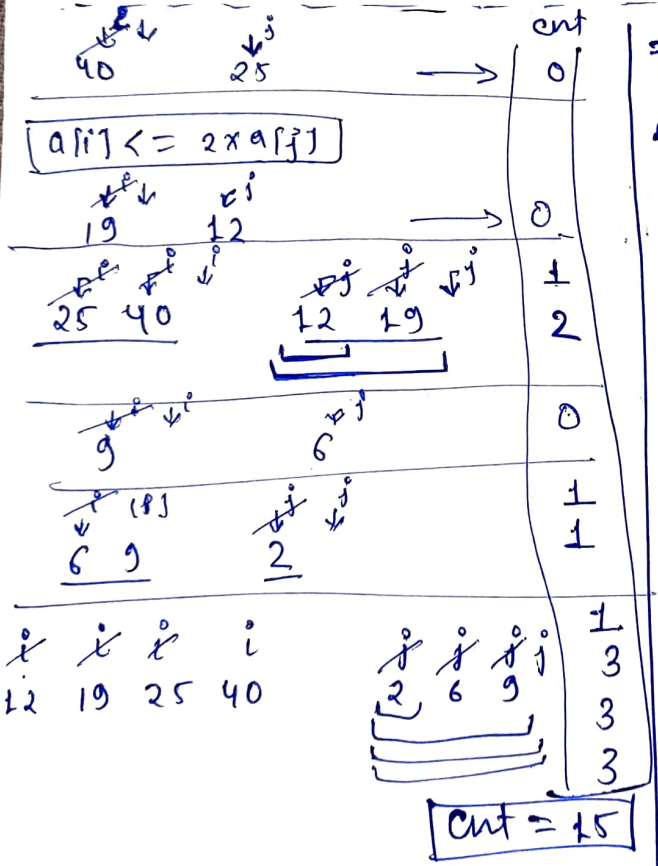


while merging, we are checking if $a[i] <= 2*a[j]$

then i++ & cnt+=0

else cnt+= j se pehle tk ka sb in 2nd part and j++

while j moved, then




```
#code

int merge(V<i>& nums, low,
    mid, high) {
    total = 0;
    j = mid + 1;
    for(i=low ---- i<=mid){
      while(j<=high and
          nums[i] > 2LL*nums[j]){
            j++;
      }
      total += (j-(mid+1));
    }
    V<i> t;
    left = low, right = mid+1;
    while(left<=mid and
          right<=high){
      if(nums[left]<=nums[right])
        t.push_back(nums[left++]);
      else
        t.p_b(nums[right++]);
    }
```

TC :- $o(n\log n) + O(n) + O(n)$
SC :- $O(n)$

for merging — for counting

```
        while (left <= mid)
            temp.push_back(nums[left++]);
        while (right <= high)
            temp.push_back(nums[right++]);
        for (i = low ; i <= high)
            nums[i] = temp[i - low];
        return total;
    }
int mergesort(vector<int>& nums, low, high)
    if (low >= high) return 0;
    int mid = (low + high)/2;
    int inv = Mergesort(nums, low, mid);
    inv += mergesort(nums, mid+1, high);
    inv += merge(nums, low, mid, high);
    return inv;
}

int reversepairs(vector<int> &ans){
    return mergesort(ans, 0, ans.size()-1);
}
```