Date__/__/------

// else high = mid-1

Q.2: Problem Name: Majority Element [n/2]

nums = [3,2,3], output: 3

// Brute force: { TC: O(N) SC: O(N)

// using hashmap

// put inside map

// check if (mp[nums[i]] > (n/2))

// return nums[i]

// Second Approach (optimal)

// Initialize two variable
- count
- element

Remarks // Traversing through nums array

// count is zero then initialize the current traversing integer of array as element.

// if they are different decrease count by 1.

{ TC: O(N)
  SC: O(1)

Q-3 : Majority Element [n/3]

// same as (n/2)

Q-4 : Problem Name : Unique Paths

m=3, n=7 output = 28 //

// Brute force:

// traverse every possible direction from left to top and right to bottom.

// Use recursion { TC: exponential
SC: exponential

Remarks

ate__/__/_____ No

// Optimal approach

// dynamic programming

// create a dp[m][n]

// for(i=0 to m)

for (j=0 to n)

// Base condition

// if (i==0 || j==0)

dp[i][j] = 1;

else

// dp[i][j] = dp[i-1][j] + dp[i][j-1]

// return dp[m-1][n-1]

TC : O(n-1) or O(m-1)

SC : O(1)

Remarks

Notes Date__/__/_____

Q-5 Problem Name : Reverse Pairs

nums = [1, 3, 2, 3, 1]
Output = 2

// condition of reverse pairs -
i < j and arr[i] > 2 * arr[j]

// Brute force
// create a variable pairs = 0;
// for (i = 0 to n)
for (j = i+1 to n)
if (arr[i] > 2 * arr[j])
Pairs++;
return pair;

TC : O(N²)

Remarks SC : O(1)

// optimal using Merge Sort

// call left recursion and right recursion and then call merge f

// After that need to merge the both left and right sub-array using temp vector.

// After this copy back to original array.

Tc: O(N) + O(N) + O(N log N)

Sc: O(N)

↳ temp vector.

Remarks