Striver's SDE Sheet

Day 2: Array ||

Rotate Image

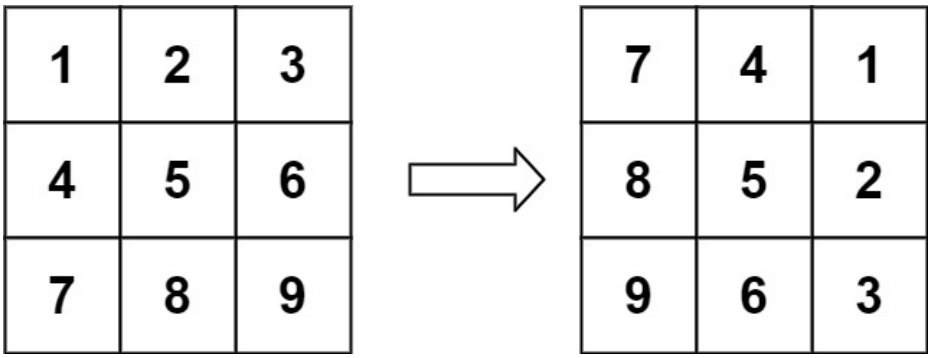
You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Ans:

Observation and Approach: (Pattern or can say process based question)

Eg:



Pattern observed: first take transpose of matrix then reverse the row of matrix

Code:

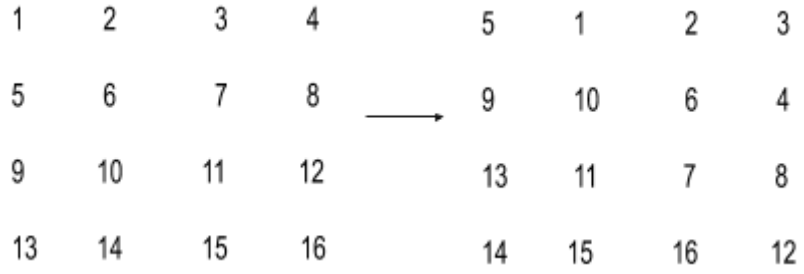
```
void rotate(vector<vector<int>>& m) {
       //Taking transpose of the matrix
       for(int i=0;i<m.size();i++)
           for(int j = 0 ;j<i ; j++)
             swap(m[i][j] , m[j][i]);

       //reversing each row of the matrix
       for(int i=0 ; i<m.size();i++)
           reverse(m[i].begin(), m[i].end());

   }
```

variation : Rotate Matrix

Eg:



Pattern observed: take a layer for example outside one shift the element to right and take first element from left to pass rightest element to bottom

So on for all different layer

Code: TC O(N*M)

```
int sr =0, er=n-1, sc=0, ec=m-1;
        //sr -> starting row, er -> ending row, sc -> starting column, ec -> ending column
        if(er==0 or ec ==0)return ;
        while(sr<er and sc<ec)
        {
            int temp = mat[sr][sr];
            for(int j = sc+1;j<=ec;j++){
                swap(temp,mat[sr][j]);  }
            for(int i=sr+1;i<=er;i++){
                swap(temp,mat[i][ec]);
            }
            for(int j= ec-1;j>=sc;j--){
                swap(temp,mat[er][j]);
            }
            for(int i = er-1;i>=sr;i--){
                swap(temp,mat[i][sc]);
            }
            sr++; er--; sc++; ec--;
        }
```

Different type patter question can formed like this (observer the pattern do what they want to do)

Merge Intervals

Given an array of intervals where intervals[i] = [start$_i$, end$_i$], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Eg:

Input: intervals = [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Ans:

Observation and approach: (process based question)

In this question we just have to sort the array according to staring time then travel to whole array and checking the condition that ending is greater than staring time then merge it

For eg:

[1 3] [2 6] -> 3>2 -> [1,6] ans

Code: TC O(NlogN) (already optimize)

```
vector<vector<int>> merge(vector<vector<int>>& intervals) {
        int n = intervals.size();
        sort(intervals.begin(), intervals.end());
        vector<vector<int>> output;
        for(auto interval : intervals){
            if(output.empty() || output.back()[1] < interval[0]){
                output.push_back(interval);
            }
            else{
                output.back()[1] = max(output.back()[1], interval[1]);
            }
        }
        return output;
    }
```

Variation: a variation of this question available at CSES problem set

Merge Sorted Array

You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be *stored inside the array* nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Ans:

Observation and approach: (process based question)

In this question brute force approach is to first insert element of num2 in num1 then apply sort which take O(NlogM)

And optimize solution is to maintain two pointer which keep track of array and store it in sorted way in nums1

Code: brute

```
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        for(int i=0; i<n; i++){
            nums1[m+i] = nums2[i];
        }
        sort(nums1.begin(), nums1.end());
    }
```

Code: optimize

```
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m-1; int j = n-1; int k = m+n-1;
        while(i>=0 && j>=0){
            if(nums1[i] > nums2[j]){
                nums1[k] = nums1[i]; i--; k--;
            }
            else{
                nums1[k] = nums2[j]; j--; k--;
            }
        }
        while(j>=0){
            nums1[k] = nums2[j];  j--; k--;
        }
```

Variation:

1) So many variation can be done like only fill element in zero position or only sort this much array to ascending and this much to descending this all can be handle by just give few condition in original approach

Find the Duplicate Number

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.

There is only one repeated number in nums, return *this repeated number*.

You must solve the problem without modifying the array nums and uses only constant extra space.

Ans:

Observation and approach : (equation and process based question)

In this question we can do a simple maths approach that is sum of all element -n*(n+1)/2 but it fails in input like 2,2,2,2 😓

Second approach came in my mind is to sort the array can travel to check duplicate element

Third approach came in my mind is to apply hash table but its take extra space which is not allowed in question

Fourth approach I get is to used fast slow pointer method which take o(n) time and o(1) space

Code:

```
public int findDuplicate_fastSlow(int[] nums) {
        int slow = 0;
        int fast = 0;
        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);
        slow = 0;
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast]; }
        return slow;}
```

Binary Search (saw some of different approach on internet and find this)

Note that the key is to find an integer in the array [1,2,…,n][1, 2, \dots, n][1,2,…,*n*] instead of finding an integer in the input array.

We can use the binary search algorithm, each round we guess one number, then scan the input array, narrow the search range, and finally get the answer.

According to the Pigeonhole Principle, n+1n + 1*n*+1 integers, placed in an array of length nn*n*, at least 111 integer will be repeated.

So guess a number first(the number midmid*mid* in the valid range [left,right][left, right][*left*,*right*]), count the elements of the array which is less than or equal to midmid*mid* in the array.

1. If cntcnt*cnt* is strictly greater than midmid*mid*. According to the Pigeonhole Principle, repeated elements are in the interval [left,mid][left, mid][*left*,*mid*];
2. Otherwise, the repeated element is in the interval [mid+1,right][mid + 1, right][*mid*+1,*right*].

With extra O(1)O(1)*O*(1) space, without modifying the input.

Code:

```
public static int findDuplicate_bs(int[] nums) {

    int len = nums.length;

    int low = 1;

    int high = len - 1;

    while (low < high) {

        int mid = low + (high - low) / 2;

        int cnt = 0;

        for (int i = 0; i < len; i++) {

            if (nums[i] <= mid) {

                cnt++;

            }

        }

        if (cnt <= mid) {

            low = mid + 1;

        } else {

            high = mid;

        }

    }

    return low;

}
```

Variation can be formed like what if repeat two number or all number repeat two time and only one element repeat one time and we have to find that number or some frequency related question

Missing and repeating numbers

You are given an array of size 'N'. The elements of the array are in the range from 1 to 'N'. Ideally, the array should contain elements from 1 to 'N'. But due to some miscalculations, there is a number R in the range [1, N] which appears in the array twice and another number M in the range [1, N] which is missing from the array. Your task is to find the missing number (M) and the repeating number (R).

Ans:

Observation and approach:

Eg:

Consider an array of size six. The elements of the array are { 6, 4, 3, 5,5,1}

The array should contain elements from one to six. Here, 2 is not present and 5 is occurring twice. Thus, 2 is the missing number (M) and 5 is the repeating number (R).

According to constrains a simple approach is to make a hash map to find repeating element from its frequency and missing element by making and array of visited , if num is present the enter 1 to corresponding number

Code:

```
// Write your code here
  map<int, int> map;
  int vis[n];
  pair<int, int> ans;

  for (int i = 0; i <= n; i++) {
    map[arr[i]]++;
    vis[arr[i]] = 1;
  }
  for (auto i : map) {
    if (i.second > 1) {
      ans.second = i.first;
      break;
    }
  }
  for (int i = 1; i <= n; i++) {
    if (vis[i] != 1) {
      ans.first = i;
    }
  }
  return ans;
```

note : watch out Striver's video solution , discussed some different method in it

Count Inversions

For a given integer array/list 'ARR' of size 'N' containing all distinct values, find the total number of 'Inversions' that may exist. An inversion is defined for a pair of integers in the array/list when the following two conditions are met.

Ans:

Observation and approach: (process based question){variation of merger sort }

In first go brute force approach can be done by apply two loops and check the conditions which take O(N^2) time

Code:

```
long long int count=0;
  for(int i=0;i<n;i++)
  {
     for(int j=1;j<n;j++)
     {
         if((arr[i] > arr[j])&&(i<j))
         count++;
     }
  }
  return count;
```

optimize code:

After seeing some hint of merge sort idea that strike in my mind is we have to add some change in merger of mergershort to get our answer but not able identify correctly 😓

so after watching tutorial https://youtu.be/AseUmwVNaoY

code:

```
int merge(int arr[],int temp[],int left,int mid,int right)
{
    int inv_count=0;
    int i = left;
    int j = mid;
    int k = left;
    while((i <= mid-1) && (j <= right)){
        if(arr[i] <= arr[j]){
            temp[k++] = arr[i++];
```

```
        }
        else
        {
            temp[k++] = arr[j++];
            inv_count = inv_count + (mid - i);
        }
    }

    while(i <= mid - 1)
        temp[k++] = arr[i++];

    while(j <= right)
        temp[k++] = arr[j++];

    for(i = left ; i <= right ; i++)
        arr[i] = temp[i];

    return inv_count;
}

int merge_Sort(int arr[],int temp[],int left,int right)
{
    int mid,inv_count = 0;
    if(right > left)
    {
        mid = (left + right)/2;

        inv_count += merge_Sort(arr,temp,left,mid);
        inv_count += merge_Sort(arr,temp,mid+1,right);

        inv_count += merge(arr,temp,left,mid+1,right);
    }
    return inv_count;
}
long long getInversions(long long *arr, int n)
{
int ans = merge_Sort(arr,temp,0,n-1);
return ans;
}
```

Thinks learn from above questions:

1) The application of algorithm is mainly to understand process means who its doing because from above question (inversion count) its directly implementation of merger sort algo Means if we know the process then at the time of problem solving our mind try to map approach of previous questions that we have solved and its likely have a chance to hit the right answer