13th June, 2022

SDE SHEET BY STRIVER

Page no.: ____/

* SHORT NOTES * Date: ____/____/____

## DAY.1 : ARRAYS

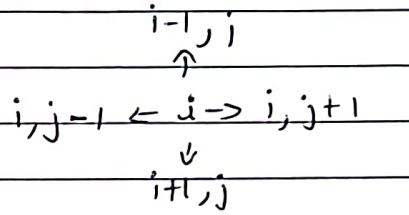
Problem :-

1] Set Matrix Zeroes

Statement :- Given a matrix if an element in the matrix is 0 then you will have to set its entire column and row to 0 and then return the matrix.

Approach 1 : Brute Force

Assuming all elements are positive. Traverse through the matrix and if you find an element with value 0, then change all elements in its row and cols to -1, except when the element is zero. Now, again traverse through the matrix again and if an element is -1 change it to zero, which is answer.




TC → O((N*M)*(N+M)). O(N*M) for traversing each element and (N+M) for traversing to row and columns having value 0.

SC → O(1)

Page no.: ____/
Date: ___/___/____/

Solution 2: Better approach

Take two dummy arrays one of size and other of size of column. Now traverse through the array, If matrix[i][j] == 0 then set dummy1[i] = 0 (for row) and dummy2[j] = 0 (for column). Now traverse through the array again and if dummy1[i] == 0 || dummy2[j] == 0 then arr[i][j] = 0, else continue

TC → O( N*M + N*M )
SC → O(N)

Solution 3: Optimal Approach
Instead, we take first and column / row as the array for checking whether the particular column or row has the value 0 or not. Since matrix[0][0] are overlapping. Therefore take seperate variable col0 (say) to check if the 0th column has 0 or not and use matrix[0][0] to check if 0th has 0 or not. Now, traverse from last to first and check matrix[i][0] || matrix[0][j] == 0 and true then mat[i][j] = 0 , else ++.

TC → O( 2(N*M) ) → O( 2*(N*M) ) as we are traversing 2 times in a matrix
SC → O(1)

Page no.: ____/

Date: ____/____/____

## 2] Program to Generate Pascal's Triangle

Problem Statement - Given an integer N, return the first N rows of Pascal's triangle.

In Pascal's Δ, each number is sum of the numbers directly above

Pseudocode :-

```
vector<vector<int>> r(numRows);
for(int i=0 ; i<numRows; i++) {
    r[i].resize(i+1);
    r[i][0] = r[i][i] = 1;
  for(int j=1; j<i; j++) {
    r[i][j] = r[i-1][j-1] + r[i-1][j];
  }
}
return r;
```

Time Complexity - $O(NumRows^2)$
SC → $O(numRows^2)$.

If Interviewers tell's us to find $N^{th}$ row Pascal element

N C formula $^{N-1}C_{C-1} = {}^{4}C_{2} = \frac{4\times3}{1\times2} = 6$

↳ optimised way

```
for(int i=0; i<k; i++) {
    res *= (n-i);
    res /= (i+1); }
```

$^{7}C_{3} = \frac{7\times5\times6}{1\times2\times3}$

Page no.: ____/
Date: ____/____/____/

3] next permutation - find next lexicographically greater permutation

Problem Statement :-
Given an arr[] of integers (strings), rearrange the numbers of the given array into the lexicographically next greater permutation of numbers.
If such an arrangement not possible, it must rearrange it as the lowest possible order (ie. sort in ascending order)

Solution 1: Brute Force
Approach :
Step 1 → Find all possible permutations and store it
Step 2 → Search for i/p in all possible permute's
Step 3 → print next permutation present right after it.

TC → O(N! x N)
SC → O(1)

Approach 2: Using in-built in C++
next_permutation(arr, arr+3);

Approach 3 :
Intuition - lies behind the lexicographical ordering of all possible permutations of a given array. There will always be an increasing sequence of all possible permutations when observed.

Page no.: ____/

Date: ____/____/____

Approach :

Step 1 : Linearly traverse from back such that $i^{th}$ index value < $(i+1)^{th}$ value store that in variable.

Step 2 : Traverse again, but if index value is less than 0 then reverse i/p to get mini. permutation. Now, if not this then find index such that has a value > previously found index . store it.

Step 3 : Swap values of step 1 and 2 indices

Step 4 : Reverse array from (step 1) index + 1 to end of array.

Pseudocode :-

```
int k, l ;
for ( k = n-2 ; k >= 0 , k-- )
    if( arr[k] < arr[k+1]
        break ; }

if ( k < 0 ) reverse ( arr.begin(), arr.end() );
else {
{ for ( l = n-1 ; l > k ; l-- )
    if( arr[l] > arr[k] )
        break ; }
swap ( arr[k], arr[l] );
reverse ( begin + k+1 ; end ) ;
}
```

Time Complexity :- $O(n) + O(n) + O(n) \simeq O(n)$

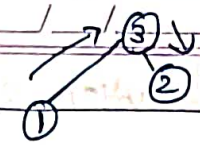
n is no. of elements in i/p array.

SC → O(1) , no extra storage

Page no.: ____/

Date: ____/____/____

Dry Run :- inp → {1, 3, 2}




| 1 | 3 | 2 |
|---|---|---|

i1, i2

| 1 | 3 | 2 |
|---|---|---|

i1 3 < 2 ×

| 1 | 3 | 2 |
|---|---|---|

i1 1 < 3 ✓

| 1 | 3 | 2 |
|---|---|---|

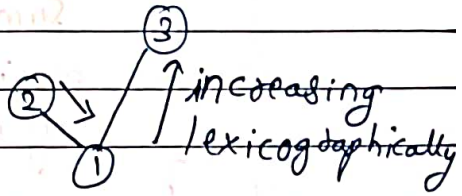
i1 i2 2 > 1

swap(i1, i2)

| 2 | 3 | 1 |
|---|---|---|

i1 → reverse( begin + i1 + 1, end )

⇒ | 2 | 1 | 3 | → answer




4] Kadane's Algorithm

→ Given an int arr, find contiguous subarray (containing atleast one number), which has the largest sum and return its sum and print the subarray.

Solution 1: Naive Approach

Use three for loops, we will get all possible subarrays in 2 loops, and their sum in another loop, and then return the max. of them.

TC → $O(n^3)$

SC → $O(1)$

Page no.: ____/
Date: ____/____/____

Solution 2: A Better approach

We can also do this problem by 2 loops, starting with (max_sum) variable which will have the final answer. Return msf (max_so_far)

TC → $O(n^2)$ , SC → O(1)
where, n is i/p array size

Solution 3: Kadane's Algorithm

Pseudocode :-

```
int sum = 0;
int maxi = INT_MIN;
for( auto it : nums ){
    sum += it;
    maxi = max( sum, maxi);
    if (sum < 0) sum = 0;
}
return maxi;
```

→ there's no point in keeping -ve sum bcz we have to find largest sum

TC → O(n)
SC → O(1)

5] Sort an array of 0's 1's and 2's. (Microsoft)

Problem Statement :- Given an array consisting of only 0s, 1s and 2s. Write a program to in-place sort the array without using inbuilt sort functions.

Page no.: ____/
Date: ___/___/____/

Solution 1 : Sorting
TC → O(nlogn)
SC → O(1)

Solution 2: Keeping count of values

Taking 3 variables to maintain the count of 0, 1 and 2.
Traverse the array, count the freq of 1, 0 and 2
2nd traverse and overwrite indices with 0 and next 'b' with 1 and remaining with 2.

TC → O(N) + O(N)
SC → O(1)

Soln - 3 3-Pointer Approach
This problem is a variation of the popular Dutch National Flag Algorithm.
Intuition - All values to the left would be 0 and to right will be 2 and in middle will be 1.

Pseudocode :-

```
int low = 0, mid = 0, high = n - 1;
while ( mid ≤ high)
switch ( nums[mid] ) {
  case 0 : swap ( nums[low ++], nums[mid++]
           break;
  case 1 : mid ++ ;  break;
  case 2 : swap ( nums[mid], nums[high --]);
           break;
}
```

TC → O(n)
SC → O(1)

Page no.: ____/

Date: ____/____/____

## 6] Stock Buy and Sell

Problem Statement → You are given an array of prices where prices [i] is the price of given stock on an ith day.

Return max. profit

Solution 1: Brute force

Intuition - We can simply use 2 for loops and track every transaction and maintain a variable maxPro to contain the max value among all transactions.

Approach - 1. i from 0 to n
2. j from i+1 to n
3. if $j^{th} > i^{th}$ compare and diff store in maxPro
4. Return maxPro

TC → $O(n^2)$

SC → $O(1)$

Solution 2: Optimal Solution

Intuition - Traverse the array. We can maintain a min. from start and compare it with every element of array, if its greater than the minimum then the difference and maintain as max, else update the minimum.

Page no.: ____/
Date: ___/___/____/

Approach - (Pseudocode)

```
int maxPro = 0
int minPrice = INT_MAX;
for( i=0 , i<n ) {
    minPrice = min( minPrice, arr[i] );
    maxPro = max( maxPro, arr[i] - minPrice );
    }
return maxPro;
```

TC → O(n) where, n is size of array.
SC → O(1)

Shaveb