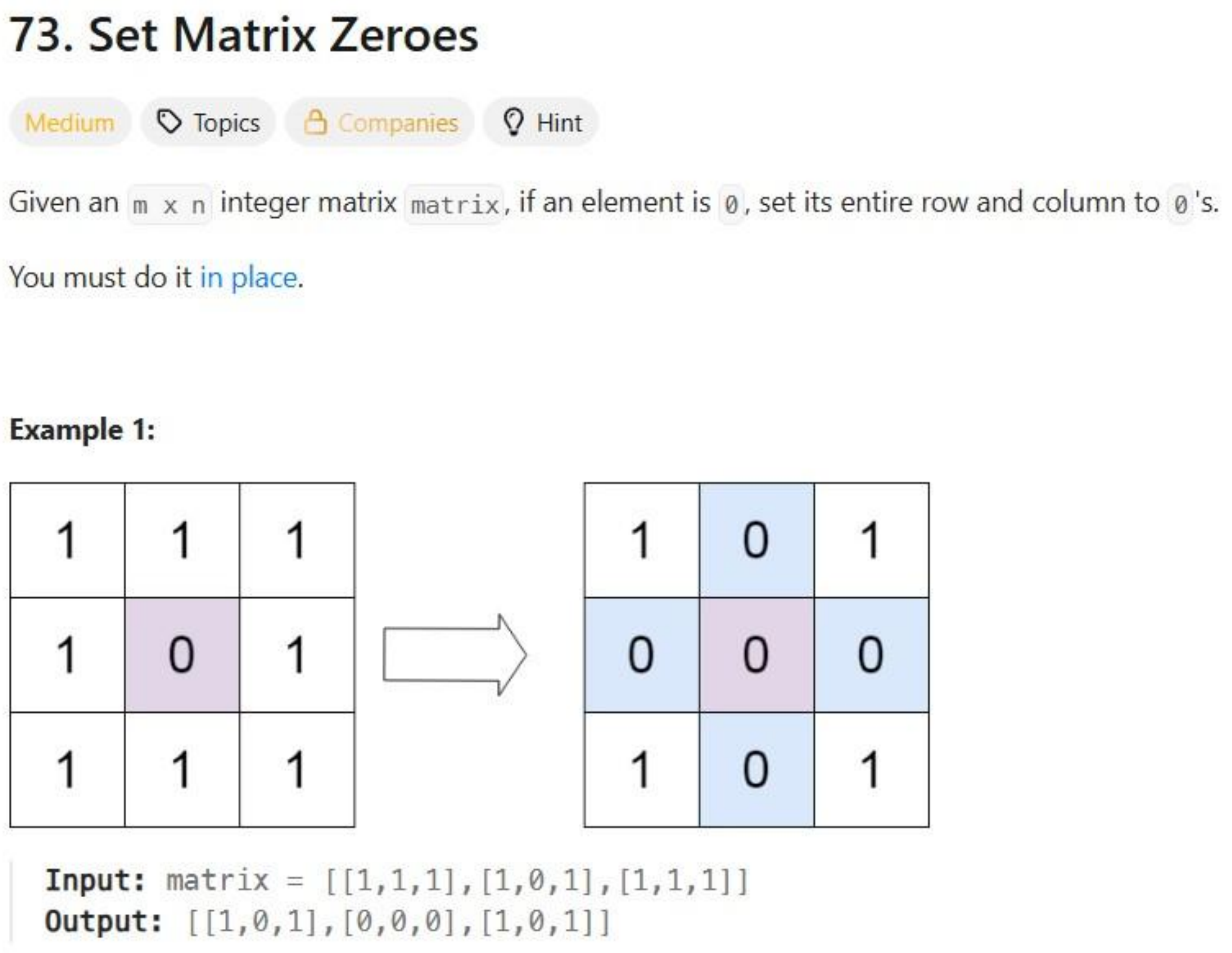
REVISION CHEAT SHEET

DAY 1



73. Set Matrix Zeroes

Medium · Topics · Companies · Hint

Given an `m x n` integer matrix `matrix`, if an element is `0`, set its entire row and column to `0`'s.

You must do it in place.

Example 1:

```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

DEV

Idea

Use the **first row and first column as markers** to record which whole rows/columns must become 0 — and keep two booleans to remember whether the first row / first column themselves must be zeroed.

Step-by-step

1. Read n = rows, m = cols. If matrix empty → return.
2. Scan first **row** → set firstRowZero = true if any matrix[0][j] == 0.
3. Scan first **col** → set firstColZero = true if any matrix[i][0] == 0.
4. For i = 1..n-1, j = 1..m-1: if matrix[i][j] == 0 then set matrix[i][0] = 0 and matrix[0][j] = 0 (use them as markers).
5. For i = 1..n-1, j = 1..m-1: if matrix[i][0] == 0 || matrix[0][j] == 0 then matrix[i][j] = 0.

6. If firstRowZero → zero the entire first row. If firstColZero → zero the entire first column.

#CODE

```
class Solution {
public:
   void setZeroes(vector<vector<int>>& matrix) {
       int n = matrix.size();
       int m = matrix[0].size();
       bool firstRowImpacted = false;
       bool firstColImpacted = false;
       //cheking for the col row
       for(int i = 0; i < n; i++){
          if(matrix[i][0] == 0){
              firstColImpacted = true;

break;
          }
       }
       //checking for first row impacted
       for(int j = 0; j < m; j++){
          if(matrix[0][j] == 0){
              firstRowImpacted = true;

break;
          }
       }
       //set matrix first element of rows and col 0;
       for(int i =0; i < n; i++){
          for(int j = 0; j < m; j++){
              if(matrix[i][j] == 0){
                  matrix[i][0] = 0;
matrix[0][j] = 0;
              }
          }
```

DEV JARWAL

```
        }
        //check if the first element of row and coloumn is zero then make whole 0
        for(int i = 1; i < n; i++){
            for(int j = 1; j < m; j++){
                if(matrix[i][0] == 0 || matrix[0][j] == 0){
                    matrix[i][j] = 0;
                }
            }
        }

        if(firstRowImpacted){
            for(int j = 0; j < m; j++){
                matrix[0][j] = 0;
           }
            }
        }

        if(firstColImpacted){
            for(int i = 0; i < n; i++){
                matrix[i][0] = 0;
            }
        }
    }
};
```

Complexity & quick notes

- Time: O(n * m) (one full pass + small extra passes)
- Extra space: O(1) (only two booleans)
- Use this when interviewer expects **in-place** solution. Simpler alternative: use two boolean arrays row[n] and col[m] → O(n+m) extra space (safe if in-place not required).

DEV JARWAL

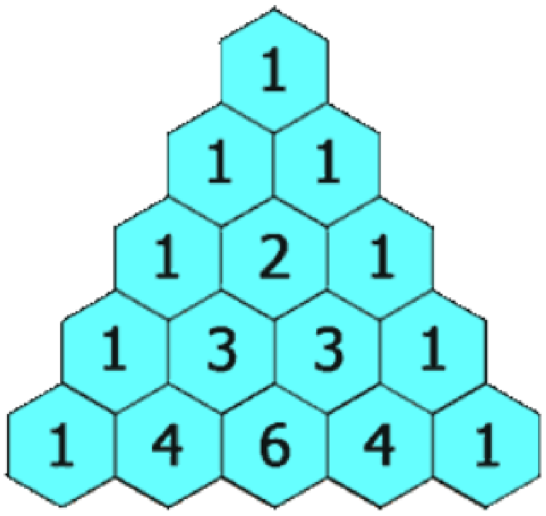
118. Pascal's Triangle

Solved ✓

Easy Topics Companies

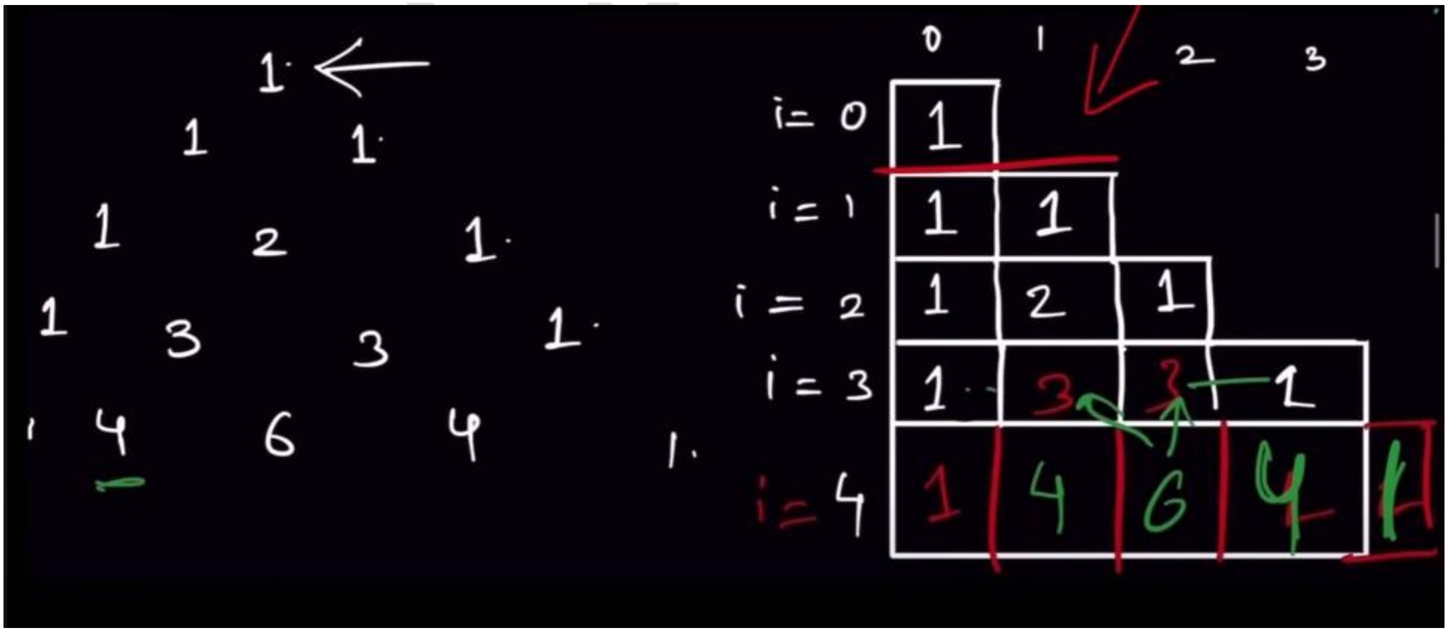
Given an integer `numRows`, return the first numRows of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



Example 1:

```
Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```



1. **Prepare the result container**
   Create a 2D vector named result that will hold numRows rows. At this point each row is empty; you will fill them one by one.

-
-

2. **Loop rows from i = 0 to i = numRows-1**
    - For each i, you are building the i-th row of Pascal's triangle (0-based). The i-th row
    - must have exactly i+1 elements.
3. **Allocate the row and set boundary 1s**
    - Allocate a vector of length i+1 and initialize every element to 1.
    - Why initialize to 1? Because the first and last elements of every row are always 1. Initializing all to 1 gives correct boundary values and makes interior replacements easy.
4. **Fill interior elements (only when i >= 2)**
    - For each position j from 1 to i-1 (these are the interior positions):
        - Compute result[i][j] = result[i-1][j-1] + result[i-1][j].
        - That uses the two values directly above the current spot: left-above and rightabove.
    - Do **not** touch j=0 and j=i because those are boundary 1s.
5. **Continue until all rows built**
    - Repeat allocation + interior filling for every row i.
    - After loop finishes, return result.

#CODE

```
class Solution {
public:
   vector<vector<int>> generate(int numRows) {
      vector<vector<int>> result(numRows);
      for(int i = 0; i < numRows; i++){
         result[i] = vector(i+1,1);
         for(int j = 1; j < i; j++){
            result[i][j] = result[i-1][j]+result[i-1][j-1];
         }
      }
      return result;
   }
   }
};
```

DEV JARWAL

31. Next Permutation

Solved ✓

Medium Topics Companies

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for `arr = [1,2,3]`, the following are all the permutations of `arr`: `[1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1]`.

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the **next permutation** of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of `arr = [1,2,3]` is `[1,3,2]`.
- Similarly, the next permutation of `arr = [2,3,1]` is `[3,1,2]`.
- While the next permutation of `arr = [3,2,1]` is `[1,2,3]` because `[3,2,1]` does not have a lexicographical larger rearrangement.

Given an array of integers `nums`, *find the next permutation of* `nums`.

The replacement must be **in place** and use only constant extra memory.

Example 1:

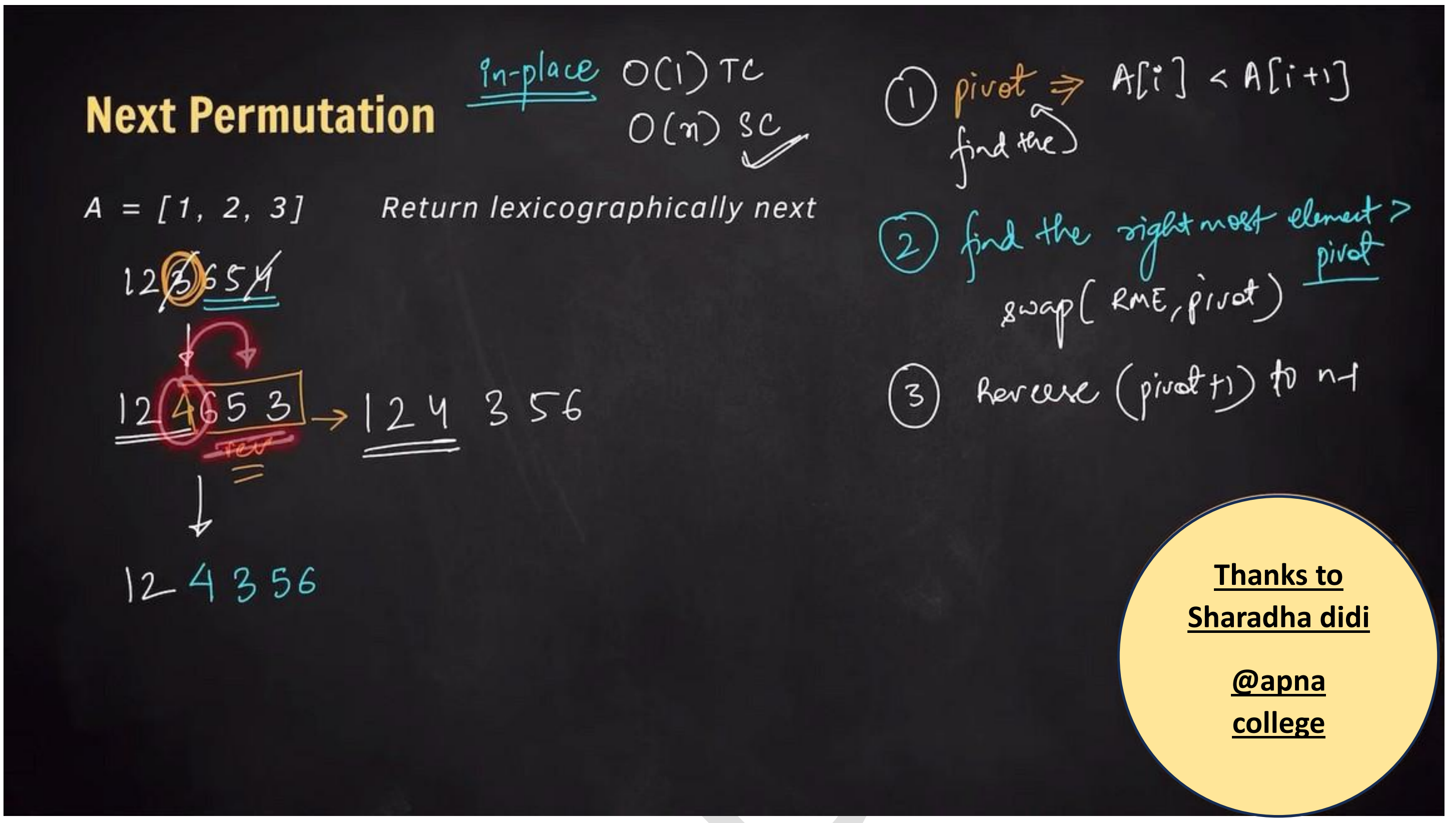
```
Input: nums = [1,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: nums = [3,2,1]
Output: [1,2,3]
```

Example 3:

```
Input: nums = [1,1,5]
Output: [1,5,1]
```



Next Permutation

in-place O(1) TC
O(n) SC ✓

A = [1, 2, 3]   Return lexicographically next

12~~3~~65~~4~~

12 4 6 5 3 → 124 356
rev

12 4 356

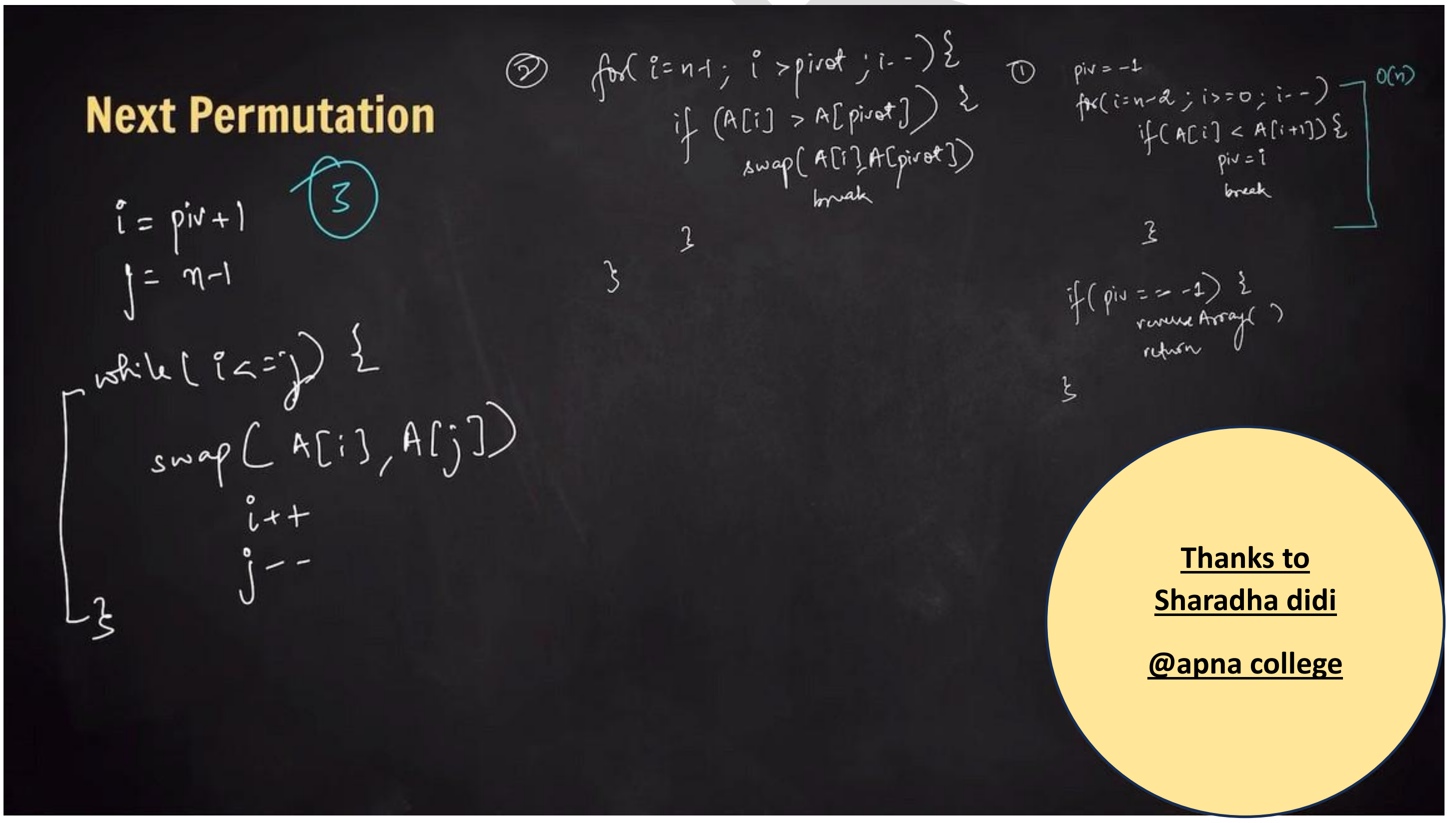
① pivot ⇒ A[i] < A[i+1]
find the

② find the right most element > pivot
swap(RME, pivot)

③ Reverse (pivot+1) to n-1

Thanks to
Sharadha didi

@apna
college



Next Permutation

③
```
i = piv+1
j = n-1

while ( i<=j) {
    swap( A[i], A[j])
        i++
        j--
}
```

②
```
for( i=n-1; i >pivot ; i--){
    if (A[i] > A[pivot]) {
        swap( A[i], A[pivot])
        break
    }
}
```

①
```
piv = -1
for( i=n-2 ; i>=0; i--)          O(n)
    if( A[i] < A[i+1]) {
        piv = i
        break
    }

if( piv == -1) {
    reverse Array( )
    return
}
```

Thanks to
Sharadha didi

@apna college

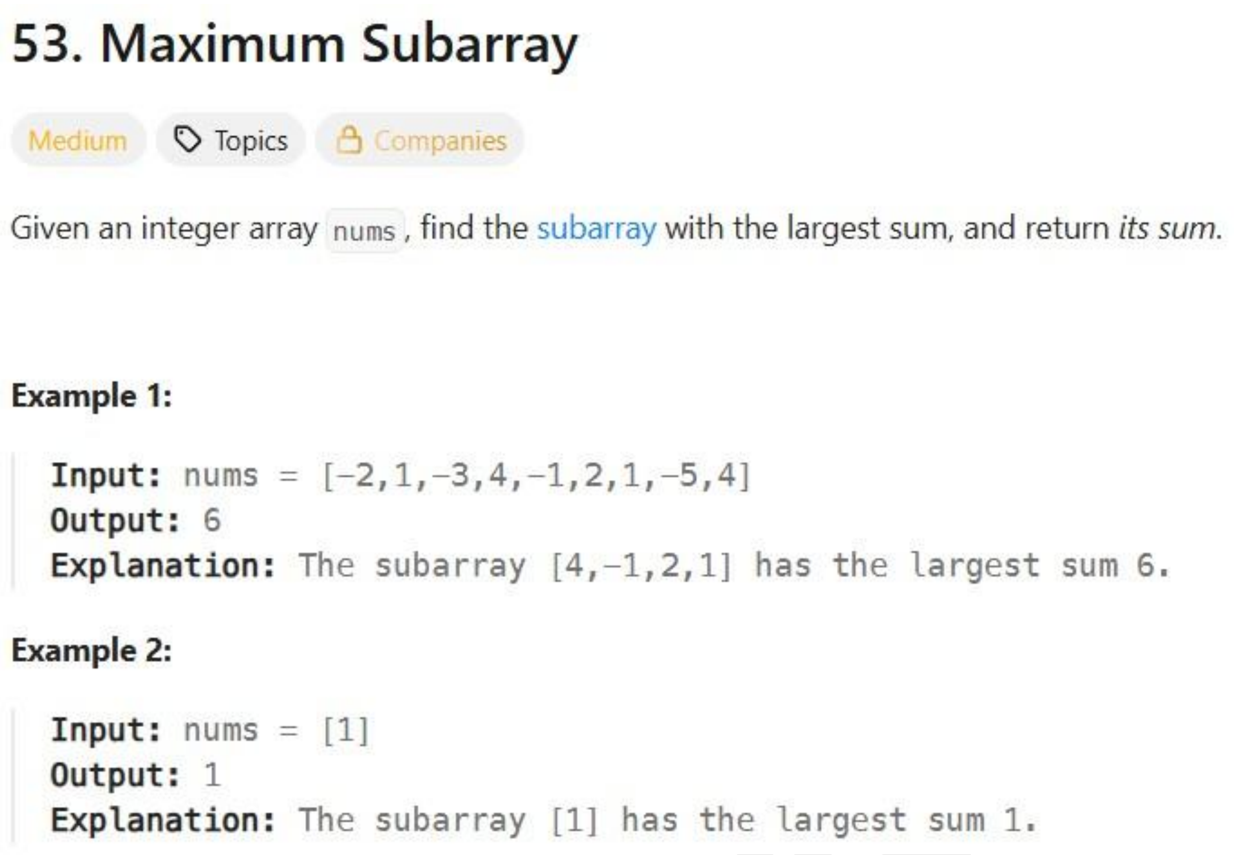
#CODE

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int piv = -1, n = nums.size();
        //step 1 is to find the pivot element
        for(int i = n - 2; i >= 0; i--){
            if(nums[i]<nums[i+1]){
                piv = i;

          break; }
            }
        }
        //if we didn't get the pivot that means array is sorted in increasing order
        //ATQ:-the array must be rearranged as the lowest possible order
        if(piv == -1){
            reverse(nums.begin(),nums.end());
            return;
        }
        //find the right most element which is greater than the pivot element
        for(int i = n - 1; i >= 0; i--){
            if(nums[i] > nums[piv]){
                swap(nums[i], nums[piv]);

break;
            }
        }
        int i = piv + 1, j = n - 1;
        while(i <= j){
            swap(nums[i],nums[j]);
            i++,j--;
        }
};
```

DEV JARWAL



53. Maximum Subarray

Medium | Topics | Companies

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

Step-by-step Explanation (each code line mapped)

1. **Function signature & inputs** ○ int maxSubArray(vector<int>& nums) → Input is an array of integers. Goal = find contiguous subarray with largest sum.
2. **Initialize variables** int n = nums.size(); → number of elements. int sum = 0; → keeps track of current subarray sum (running sum).
   - int maxSum = INT_MIN; → stores maximum sum found so far (start with very small).
3. **Loop through every element**
   - for (int i = 0; i < n; i++) → process array left → right.
4. **Add current element to running sum** sum +=
   - nums[i];
   - Meaning: extend the current subarray to include nums[i].
5. **Update maximum answer** maxSum =
   - max(sum, maxSum);
   - Compare the best seen so far with current running sum.
6. **Reset if running sum goes negative** if (sum
   - < 0) sum = 0;
   - Why? Because a negative prefix will only reduce future subarrays. So if sum < 0,
   - better to restart from next index.
7. **Return maximum**
   - After the loop, return maxSum; gives the largest subarray sum.

#CODE

```
class Solution {
public:
  int maxSubArray(vector<int>& nums) {
    int n = nums.size(), sum = 0, maxSum = INT_MIN;
    for(int i = 0; i < n; i++){
      sum+= nums[i];
      maxSum = max(sum,maxSum);
      if(sum < 0){
        sum = 0;
      }
    }
    return maxSum;
  }
};
```

DEV JARWAL

75. Sort Colors

Solved ✓

Medium | Topics | Companies | Hint

Given an array `nums` with `n` objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

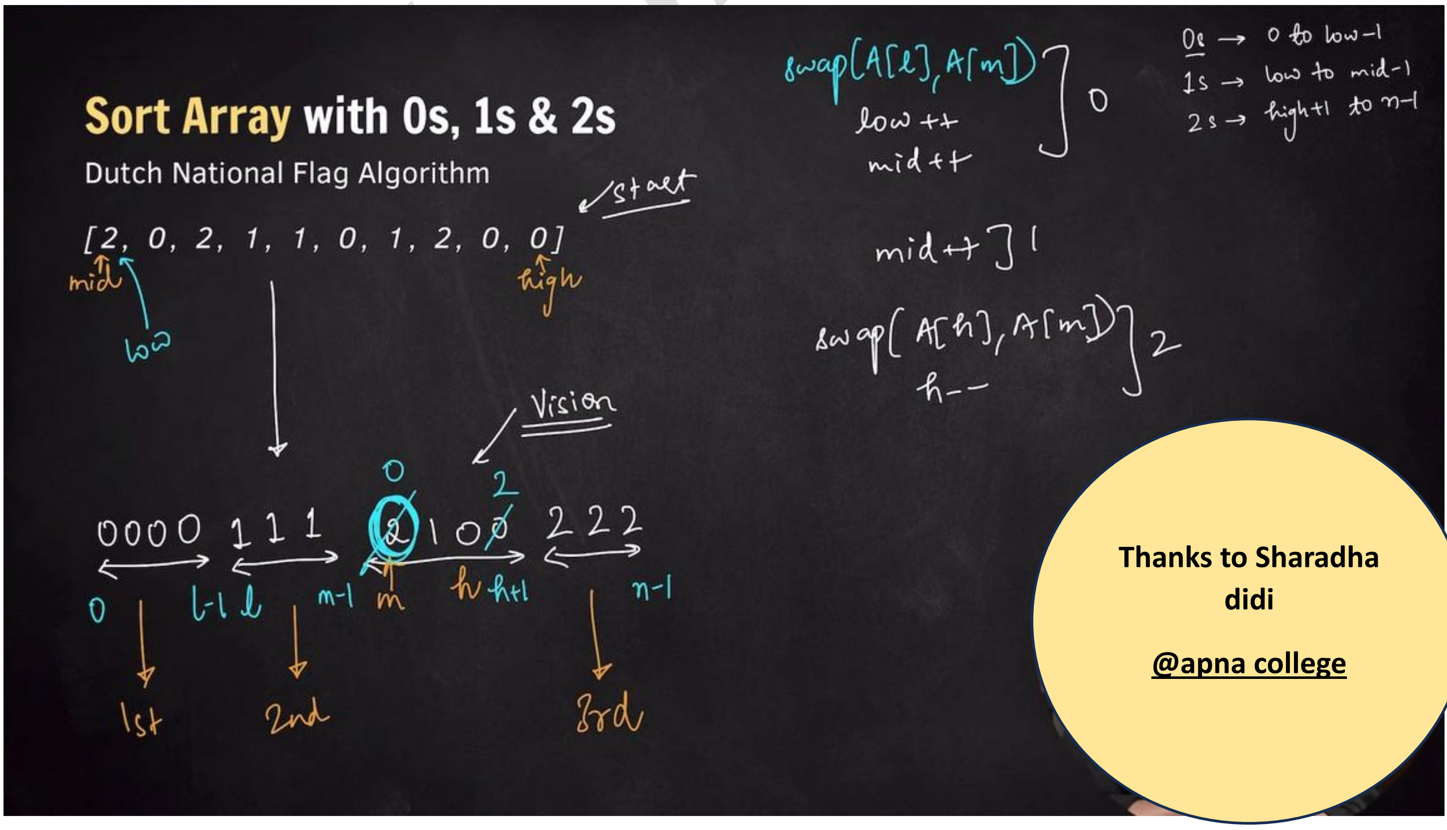
You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

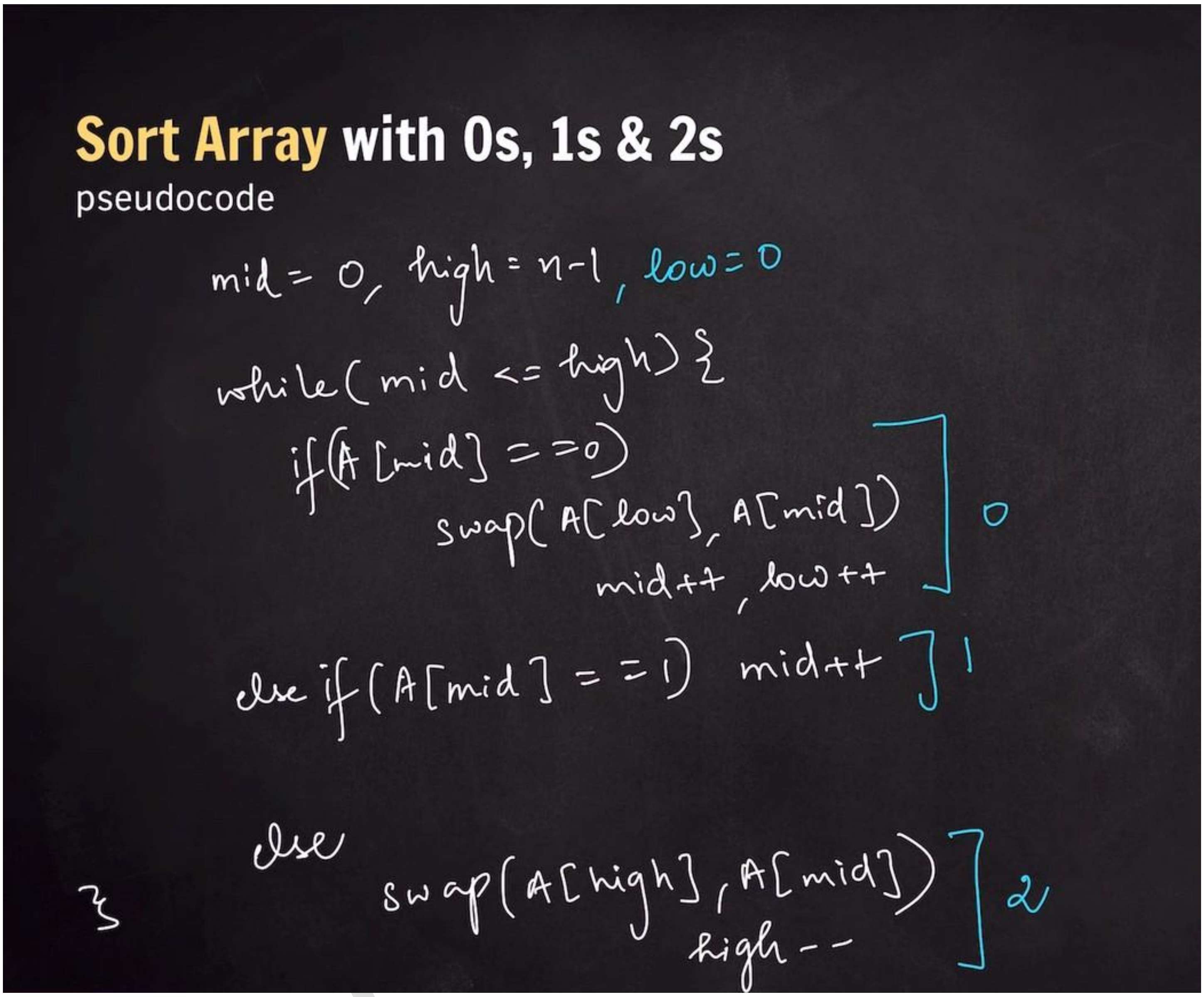
Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```



Thanks to Sharadha didi

@apna college



Sort Array with 0s, 1s & 2s

pseudocode

```
mid = 0, high = n-1, low = 0
while(mid <= high) {
    if(A[mid] == 0)
        swap(A[low], A[mid])
            mid++, low++          ] 0
    else if(A[mid] == 1) mid++    ] 1
    else
        swap(A[high], A[mid])
            high--                ] 2
}
```

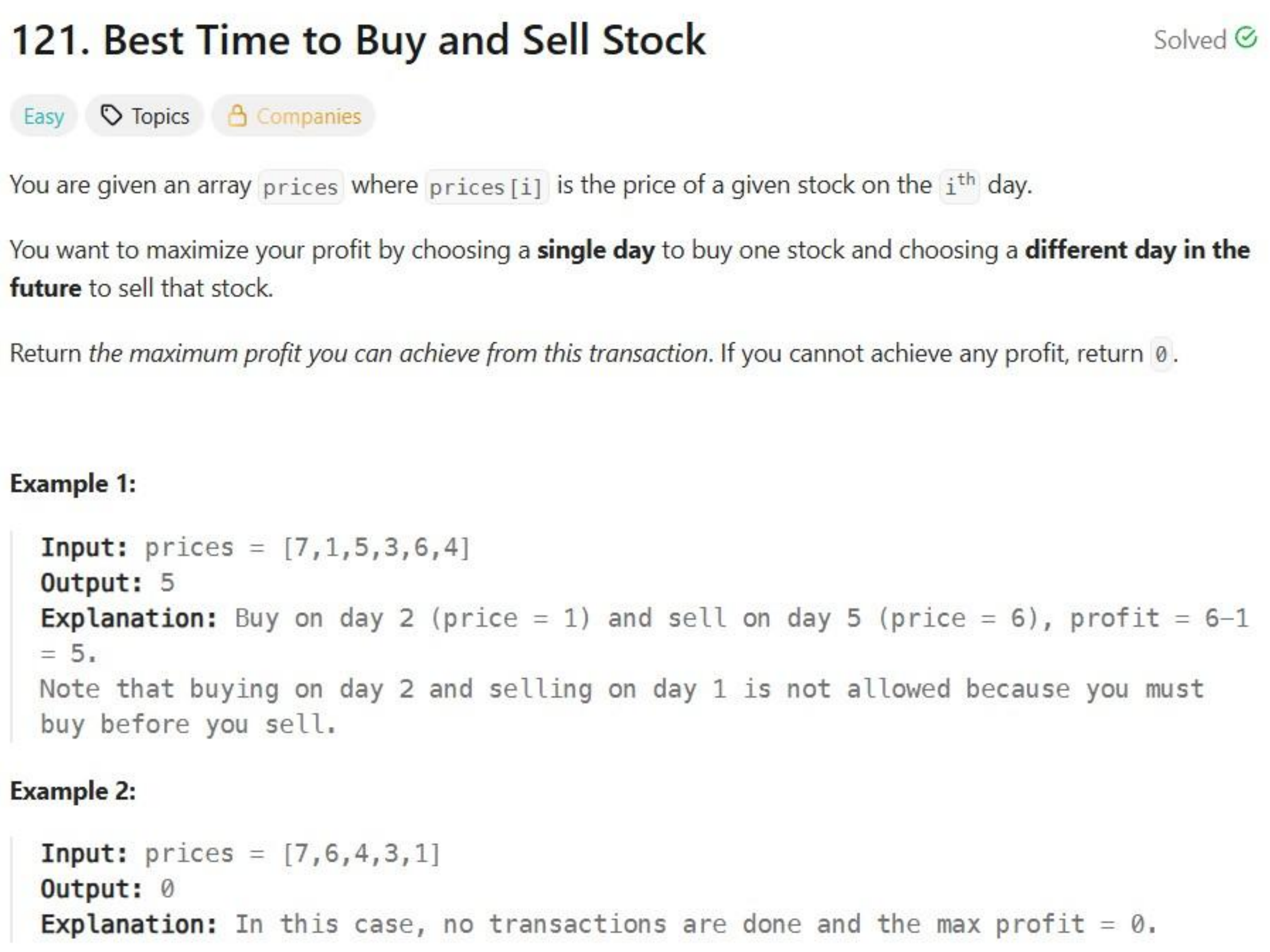
#CODE

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int n = nums.size();
        int low = 0, mid = 0, high = n - 1;
        while(mid <= high){
            if(nums[mid] == 0){
                swap(nums[low],nums[mid]);
                low++;
                mid++;
            }else if(nums[mid] == 1){
                mid++;
```

DEV

```
            }else{
                swap(nums[mid],nums[high]);
                high--;
            }
        }
    }
}
};
```

DEV JARWAL



121. Best Time to Buy and Sell Stock

Solved ✓

Easy Topics Companies

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1
= 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must
buy before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

DEV

Step-by-step Explanation (each code line mapped)

1. **Input & setup** int n = prices.size(); → number of days. int bestBuy = prices[0]; → track the lowest
    - price seen so far (best buying point).
    - int maxProfit = 0; → maximum profit found so far (start with 0, meaning no transaction is also valid).
2. - **Loop through days**
    - for (int i = 1; i < n; i++) → start from day 1 (since day 0 is already considered as initial best buy).
3. **Check potential profit** if (prices[i] > bestBuy) → only profitable if today's price is greater than lowest seen
    - so far.

      maxProfit = max(maxProfit, prices[i] - bestBuy); Profit =
    - today's price − lowest past price. Compare with previous
        - best profit.
4. **Update lowest buy price**
    - bestBuy = min(bestBuy, prices[i]);
    - Always update to the lowest stock price seen till now (better buy option).
5. **Return result**
    - After loop, return maxProfit.

#CODE

```
class Solution { public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int bestBuy = prices[0];
        int maxProfit = 0;
        for(int i = 1; i < n; i++){
            if(prices[i] > bestBuy){
                maxProfit = max(maxProfit, prices[i] - bestBuy);
            }
            bestBuy = min(bestBuy, prices[i]);
        }
        return maxProfit;
    }
};
```

DEV JARWAL