Rotate Array

Difficulty: **Medium** Accuracy: **37.06%** Submissions: **564K+** Points: **4**

Average Time: **20m**

Given an array *arr[].* Rotate the array to the left (counter-clockwise direction) by *d* steps, where *d* is a positive integer. Do the mentioned change in the **array in place**.

Note: Consider the array as circular.

Examples :

Input: arr[] = [1, 2, 3, 4, 5], d = 2
Output: [3, 4, 5, 1, 2]
Explanation: when rotated by 2 elements, it becomes 3 4 5 1 2.

Input: arr[] = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20], d = 3
Output: [8, 10, 12, 14, 16, 18, 20, 2, 4, 6]
Explanation: when rotated by 3 elements, it becomes 8 10 12 14 16 18 20 2 4 6.

C++ (12) Start Timer

```cpp
class Solution {
  public:
    // Function to rotate an array by d elements in counter-cloc
    void rotateArr(vector<int>& arr, int d) {
        // code here
        int len= arr.size();
        d=d%len;
        long long int temp[d];
        for(int i=0;i<d;i++)
        {
            temp[i]=arr[i];
        }
        for(int i=d;i<len;i++)
        {
            arr[i-d]=arr[i];
        }
        for(int i=0;i<d;i++)
        {
            arr[len-d+i]=temp[i];
        }
    }
};
```

Custom Input Compile & Run Submit

Next Permutation

Difficulty: **Medium** Accuracy: **40.66%** Submissions: **237K+** Points: **4**

Average Time: **20m**

Given an array of integers **arr[]** representing a permutation, implement the **next permutation** that rearranges the numbers into the **lexicographically next greater permutation**. If no such permutation exists, rearrange the numbers into the **lowest** possible order (i.e., sorted in ascending order).

Note: A permutation of an array of integers refers to a specific arrangement of its elements in a sequence or linear order.

Examples:

Input: arr[] = [2, 4, 1, 7, 5, 0]
Output: [2, 4, 5, 0, 1, 7]
Explanation: The next permutation of the given array is [2, 4, 5, 0, 1, 7].

Input: arr[] = [3, 2, 1]

C++ (12) | Start Timer

```
8       for(int i=n-1;i>0;i--)//found the pivot value
9       {
10          if(arr[i]>arr[i-1]){x=arr[i-1];
11              index = i-1;
12              break;
13          }
14      }
15      if(x==-1){
16          for(int i=0;i<n/2;i++)//last permutation given
17          {
18              int temp=arr[i];
19              arr[i]=arr[n-1-i];
20              arr[n-1-i]=temp;
21          }
22      }
23      else{
24          for(int i=n-1;i>=index;i--)//swapping with just larger no.
25          {
26              if(arr[i]>x){
27                  int temp= arr[i];
28                  arr[i]=x;
29                  arr[index]=temp;
30                  break;
31              }
32          }
33          for(int i=index+1 , j=n-1 ;i<j; j--,i++)//reversing suffix
34          {
35              swap(arr[i],arr[j]);
```

Custom Input | Compile & Run | Submit

Majority Element - More Than n/3

Difficulty: **Medium** Accuracy: **48.1%** Submissions: **191K+** Points: **4**
Average Time: **15m**

Given an array **arr[]** consisting of **n** integers, the task is to find all the array elements which occurs more than floor(n/3) times.

Note: The returned array of majority elements should be sorted.

Examples:

Input: arr[] = [2, 2, 3, 1, 3, 2, 1, 1]
Output: [1, 2]
Explanation: The frequency of 1 and 2 is 3, which is more than floor n/3 (8/3 = 2).

Input: arr[] = [-5, 3, -5]
Output: [-5]
Explanation: The frequency of -5 is 2, which is more than floor n/3 (3/3 = 1).

C++ (12) Start Timer

```cpp
class Solution {
  public:
    vector<int> findMajority(vector<int>& arr) {
        // Code here
        int floor= arr.size()/3;
        int n=arr.size();
        vector <int> count;
        sort(arr.begin(),arr.end());

        for(int i=0;i<n;)
        {
            int j=i+1;
            while(j<n && arr[i]==arr[j]){j++;}
            int freq=j-i;
            if(freq>floor)
            {
                count.push_back(arr[i]);
            }
            i=j;
        }

        return count;
    }
};
```