1.	**Maximum SubArray, Kadanes's Algorithm**	**[Easy]**
	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. Example 3: Input: nums = [5,4,-1,7,8] Output: 23 Explanation: The subarray [5,4,-1,7,8] has the largest sum 23. Constraints: • $1 <= nums.length <= 10^5$ • $-10^4 <= nums[i] <= 10^4$ Follow up: If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle.	

2.	**Move Zeros**	**[Easy]**
	A chocolate factory is packing chocolates into the packets. The chocolate packets here represent an array of N number of integer values. The task is to find the empty packets (0) of chocolate and push it to the end of the conveyor belt (array). 1<=N<=10^5 Example 1 : Input: N=8 and arr = [4,5,0,1,9,0,5,0]. Output: Arr = [4,5,1,9,5,0,0,0] There are 3 empty packets in the given set. These 3 empty packets represented as O should be pushed towards the end of the array	

3.	**Missing Number**	**[Easy]**

Given an array *'a'* of size *'n'*-1 with elements of range 1 to 'n'. The array does not contain any duplicates. Your task is to find the missing number.

For example:
Input:
'a' = [1, 2, 4, 5], 'n' = 5

Output :
3

Explanation: 3 is the missing value in the range 1 to 5.
Detailed explanation (Input/output format, Notes, Images)
Sample Input 1 :
4
1 2 3
Sample Output 1:
4
Explanation Of Sample Input 1:
4 is the missing value in the range 1 to 4.
Sample Input 2:
8
1 2 3 5 6 7 8
Sample Output 2:
4
Explanation Of Sample Input 2:
4 is the missing value in the range 1 to 8.
Expected time complexity:
The expected time complexity is O(n).
Constraints:
1 <= 'n' <= 10^6
1 <= 'a'[i] <= 'n'
Time Limit: 1 sec

4.	**Two Sum**	**[Easy]**

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.
You may assume that each input would have exactly one solution, and you may not use the same element twice.
You can return the answer in any order.
Example 1:
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
Example 2:
Input: nums = [3,2,4], target = 6
Output: [1,2]
Example 3:
Input: nums = [3,3], target = 6
Output: [0,1]
Constraints:
2 <= nums.length <= 104
-109 <= nums[i] <= 109
-109 <= target <= 109
Only one valid answer exists.

Follow-up: Can you come up with an algorithm that is less than O(n2) time complexity?

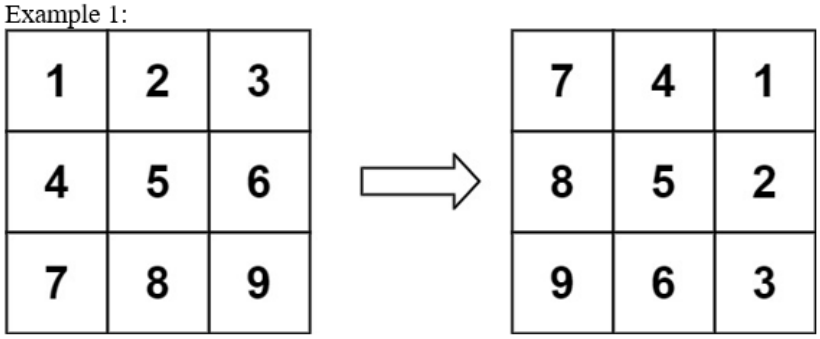
5.	**Remove Duplicates**	**[Easy]**
	You are given a sorted integer array 'arr' of size 'n'. You need to remove the duplicates from the array such that each element appears only once. Return the length of this new array. Note: Do not allocate extra space for another array. You need to do this by modifying the given input array in place with O(1) extra memory. For example: 'n' = 5, 'arr' = [1 2 2 2 3]. The new array will be [1 2 3]. So our answer is 3. Sample input 1: 10 1 2 2 3 3 3 4 4 5 5 Sample output 1: 5 Explanation of sample input 1: The new array will be [1 2 3 4 5]. So our answer is 5. Sample input 2: 9 1 1 2 3 3 4 5 5 5 Sample output 2: 5 Expected time complexity: The expected time complexity is O(n). Constraints : 1 <= 'n' <= 10^6 -10^9 <= 'arr[i]' <=10^9 Where 'arr[i]' is the value of elements of the array.	

6.	**Sort Colors**	**[Medium]**
	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] Constraints: • n == nums.length • 1 <= n <= 300 • nums[i] is either 0, 1, or 2. Follow up: Could you come up with a one-pass algorithm using only constant extra space?	

7.	**Rotate Image**	**[Medium]**

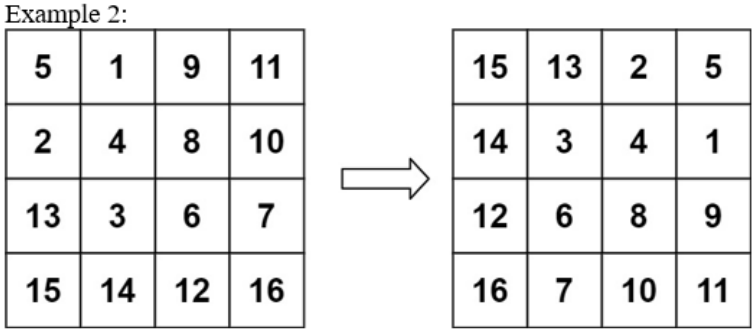
You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).
You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Example 1:



Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]

Example 2:



Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

Constraints:

- n == matrix.length == matrix[i].length
- 1 <= n <= 20
- -1000 <= matrix[i][j] <= 1000

8.	**Next Permutation**	**[Medium]**

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]

Constraints:

- 1 <= nums.length <= 100
- 0 <= nums[i] <= 100

9.	**Book Allotment Problem**	**[Hard]**
	Given an array *'arr'* of integer numbers, 'arr[i]' represents the number of pages in the 'i-th' book. There are *'m'* number of students, and the task is to allocate all the books to the students. Allocate books in such a way that: 1. Each student gets at least one book. 2. Each book should be allocated to only one student. 3. Book allocation should be in a contiguous manner. You have to allocate the book to 'm' students such that the maximum number of pages assigned to a student is minimum. If the allocation of books is not possible, return -1. Example: Input: 'n' = 4 'm' = 2 'arr' = [12, 34, 67, 90] Output: 113 Explanation: All possible ways to allocate the '4' books to '2' students are: 12 | 34, 67, 90 - the sum of all the pages of books allocated to student 1 is '12', and student two is '34+ 67+ 90 = 191', so the maximum is 'max(12, 191)= 191'. 12, 34 | 67, 90 - the sum of all the pages of books allocated to student 1 is '12+ 34 = 46', and student two is '67+ 90 = 157', so the maximum is 'max(46, 157)= 157'. 12, 34, 67 | 90 - the sum of all the pages of books allocated to student 1 is '12+ 34 +67 = 113', and student two is '90', so the maximum is 'max(113, 90)= 113'. We are getting the minimum in the last case. Hence answer is '113'. Detailed explanation (Input/output format, Notes, Images) Sample Input 1: 4 2 12 34 67 90 Sample Output 1: 113	

	Explanation of sample input 1: All possible ways to allocate the ‘4’ books to '2' students are: 12 | 34, 67, 90 - the sum of all the pages of books allocated to student 1 is ‘12’, and student two is ‘34+ 67+ 90 = 191’, so the maximum is ‘max(12, 191)= 191’. 12, 34 | 67, 90 - the sum of all the pages of books allocated to student 1 is ‘12+ 34 = 46’, and student two is ‘67+ 90 = 157’, so the maximum is ‘max(46, 157)= 157’. 12, 34, 67 | 90 - the sum of all the pages of books allocated to student 1 is ‘12+ 34 +67 = 113’, and student two is ‘90’, so the maximum is ‘max(113, 90)= 113’. We are getting the minimum in the last case. Hence answer is ‘113’. Sample Input 2: 5 4 25 46 28 49 24 Sample Output 2: 71 Explanation of sample input 2: All possible ways to allocate the ‘5’ books to '4' students are: 25 | 46 | 28 | 49 24 - the sum of all the pages of books allocated to students 1, 2, 3, and 4 are '25', '46', '28', and '73'. So the maximum is '73'. 25 | 46 | 28 49 | 24 - the sum of all the pages of books allocated to students 1, 2, 3, and 4 are '25', '46', '77', and '24'. So the maximum is '77'. 25 | 46 28 | 49 | 24 - the sum of all the pages of books allocated to students 1, 2, 3, and 4 are '25', '74', '49', and '24'. So the maximum is '74'. 25 46 | 28 | 49 | 24 - the sum of all the pages of books allocated to students 1, 2, 3, and 4 are '71', '28', '49', and '24'. So the maximum is '71'. We are getting the minimum in the last case. Hence answer is ‘71’. Expected time complexity: The expected time complexity is O(n * log(s)), where ‘n’ is the number of integers in the array ‘arr’ and ‘s’ is the sum of all the elements of ‘arr’. Constraints: 2 <= 'n' <= 10 ^ 3

	1 <= 'm' <= 10 ^ 3 1 <= 'arr[i]' <= 10 ^ 9 The sum of all arr[i] does not exceed 10 ^ 9. Where ‘n’ denotes the number of books and ‘m’ denotes the number of students. ‘arr[i]’ denotes an element at position ‘i’ in the sequence.

10.	**Painter's Partition Problem**	**[Hard]**
	Given an integer array 'A' of size 'N' and an integer 'K'. Split the array 'A' into 'K' non-empty subarrays such that the largest sum of any subarray is minimized. Your task is to return the minimized largest sum of the split. A subarray is a contiguous part of the array. Example: Input: 'N' = 5, 'A' = [1, 2, 3, 4, 5], 'K' = 3 Output: 6 Explanation: There are many ways to split the array 'A' into K consecutive subarrays. The best way to do this is to split the array 'A' into [1, 2, 3], [4], and [5], where the largest sum among the three subarrays is only 6. Detailed explanation (Input/output format, Notes, Images) Sample Input 1: 5 1 2 3 4 5 3 Sample Output 1: 6 Explanation Of Sample Input 1: Input: 'N' = 5, 'A' = [1, 2, 3, 4, 5], 'K' = 3 Output: 6 Explanation: There are many ways to split the array 'A' into K consecutive subarrays.	

	The best way to do this is to split the array 'A' into [1, 2, 3], [4], and [5], where the largest sum among the three subarrays is only 6. Sample Input 2: 3 3 5 1 3 Sample Output 2: 5 Explanation Of Sample Input 2: Input: 'N' = 3, 'A' = [3, 5, 1], 'K' = 3 Output: 5 Explanation: There is only one way to split the array 'A' into 3 subarrays, i.e., [3], [5], and [1]. The largest sum among these subarrays is 5. Sample Input 3: 6 10 4 5 10 9 10 4 Sample Output 3: 15 Constraints: 1 <= N <= 1e4 1 <= A[i] <= 1e5 1 <= K <= N Time Limit: 1-sec