

Question 1

Given an integer array **nums** of length **n** and an integer **target**, find three integers in **nums** such that the sum is closest to the target.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Example 1:

Input: **nums** = [-1,2,1,-4], **target** = 1

Output: 2

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

Question 2

Given an array **nums** of **n** integers, return *an array of all the unique quadruplets* **[nums[a], nums[b], nums[c], nums[d]]** such that:

- $0 \leq a, b, c, d < n$
- **a, b, c, and d** are distinct.
- **nums[a] + nums[b] + nums[c] + nums[d] == target**

You may return the answer in any order.

Example 1:

Input: **nums** = [1,0,-1,0,-2,2], **target** = 0

Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

Question 3

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 an 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]`

Question 4

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: `nums = [1,3,5,6]`, `target = 5`

Output: 2

Question 5

You are given a large integer represented as an integer array `digits`, where each `digits[i]` is the *i*th digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return *the resulting array of digits*.

Example 1:

Input: `digits = [1,2,3]`

Output: `[1,2,4]`

Explanation: The array represents the integer 123.
Incrementing by one gives $123 + 1 = 124$.
Thus, the result should be [1,2,4].

Question 6

Given a non-empty array of integers **nums**, every element appears *twice* except for one. Find that single one.
You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

Input: nums = [2,2,1]

Output: 1

Question 7

You are given an inclusive range [**lower**, **upper**] and a sorted unique integer array **nums**, where all elements are within the inclusive range.
A number **x** is considered missing if **x** is in the range [**lower**, **upper**] and **x** is not in **nums**.

Return the shortest sorted list of ranges that exactly covers all the missing numbers. That is, no element of **nums** is included in any of the ranges, and each missing number is covered by one of the ranges.

Example 1:

Input: nums = [0,1,3,50,75], lower = 0, upper = 99

Output: [[2,2],[4,49],[51,74],[76,99]]

Explanation: The ranges are:

[2,2]

[4,49]

[51,74]

[76,99]

Question 8

Given an array of meeting time **intervals** where **intervals[i] = [starti, endi]**, determine if a person could attend all meetings.

Example 1:

Input: intervals = [[0,30],[5,10],[15,20]]

Output: false