**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)

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def threeSumClosest(self, nums: List[int], target: int) -> int:

        nums=sorted(nums)

        diff=float('inf')

        for i in range(len(nums)-1):

            start=i+1

            end=len(nums)-1

            while(start<end):

                sum=nums[i]+nums[start]+nums[end]

                if sum==target:

                    return target

                elif abs(target-sum)<diff:

                    diff=abs(target-sum)

                    ans=sum

                if sum>target:

                    end-=1

                else:

                    start+=1

        return ans

**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 <= 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]]

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:

        nums.sort()

        res, quad = [], []

        def KSum(k, start, target):

            if k != 2:

                for i in range(start, len(nums) - k + 1):

                    if start < i and nums[i - 1] == nums[i]:

                        continue

                    quad.append(nums[i])

                    KSum(k - 1, i + 1, target - nums[i])

                    quad.pop()

                return

            l, r = start, len(nums) - 1

            while l < r:

                if nums[l] + nums[r] < target:

                    l += 1

                elif nums[l] + nums[r] > target:

                    r -= 1

                else:

                    res.append(quad + [nums[l], nums[r]])

                    l += 1

                    while l < r and nums[l - 1] == nums[l]:

                        l += 1

        KSum(4, 0, target)

        return res

**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]

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def nextPermutation(self, nums: List[int]) -> None:

        """

        Do not return anything, modify nums in-place instead.

        """

        def reverse(nums, start):

            i, j = start, len(nums) - 1

            while i < j:

                nums[i], nums[j] = nums[j], nums[i]

                i += 1

                j -= 1

        i = len(nums) - 2

        while i >= 0 and nums[i + 1] <= nums[i]:

            i -= 1

        if i >= 0:

            j = len(nums) - 1

            while nums[j] <= nums[i]:

                j -= 1

            nums[i], nums[j] = nums[j], nums[i]

        reverse(nums, i + 1)

**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

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def searchInsert(self, nums: List[int], target: int) -> int:

        if target in nums==True:

            return nums.index(target)

        else:

            nums.append(target)

            nums.sort()

        return nums.index(target)

**Question 5** Given an integer array nums, find three numbers whose product is maximum and return the maximum product.

**Example 1:**

Input: nums = [1,2,3]

Output: 6

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def maximumProduct(self, nums: List[int]) -> int:

        nums.sort()

        return max(nums[0]\*nums[1]\*nums[-1],nums[-1]\*nums[-2]\*nums[-3])

**Question 6** Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1. You must write an algorithm with O(log n) runtime complexity.

Input: nums = [-1,0,3,5,9,12], target = 9

Output: 4

**Explanation:** 9 exists in nums and its index is 4

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def search(self, nums: List[int], target: int) -> int:

        if target in nums:

            return nums.index(target)

        else: return -1

**Question 7** An array is monotonic if it is either monotone increasing or monotone decreasing. An array nums is monotone increasing if for all i <= j, nums[i] <= nums[j]. An array nums is monotone decreasing if for all i <= j, nums[i] >= nums[j]. Given an integer array nums, return true if the given array is monotonic, or false otherwise.

**Example 1:**

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

Output: true

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def isMonotonic(self, nums: List[int]) -> bool:

        dis=0

        ass=0

        eq=0

        j=0

        length=len(nums)

        while j<length-1:

            if nums[j]<nums[j+1]:

                ass+=1

            elif nums[j]==nums[j+1]:

                eq+=1

            elif nums[j]>nums[j+1]:

                dis+=1

            j+=1

        if ass+eq==length-1 or dis+eq==length-1:

            return True

        else: return False

**Question 8** You are given an integer array nums and an integer k. In one operation, you can choose any index i where 0 <= i < nums.length and change nums[i] to nums[i] + x where x is an integer from the range [-k, k]. You can apply this operation at most once for each index i. The score of nums is the difference between the maximum and minimum elements in nums. Return the minimum score of nums after applying the mentioned operation at most once for each index in it.

**Example 1:**

Input: nums = [1], k = 0

Output: 0

**Explanation:** The score is max(nums) - min(nums) = 1 - 1 = 0.

**Ans :** Solution from my leetcode : <https://leetcode.com/gopsa2001/>

class Solution:

    def smallestRangeI(self, nums: List[int], k: int) -> int:

        mx=max(nums)

        mn=min(nums)

        return max(0,(mx-k)-(mn+k))