**Kth Largest Element in an Array**

import heapq

def findKthLargest(nums, k):

# Create a min-heap with the first k elements of the array

min\_heap = nums[:k]

heapq.heapify(min\_heap)

# Iterate over the rest of the array

for num in nums[k:]:

if num > min\_heap[0]:

# If the current number is larger than the smallest in the heap, replace it

heapq.heappushpop(min\_heap, num)

# The root of the heap will be the kth largest element

return min\_heap[0]

# Input reading and function call

n = int(input()) # Number of elements in the array

nums = [int(input()) for \_ in range(n)] # Array elements

k = int(input()) # The kth largest number to find

# Output the result

print(findKthLargest(nums, k))

#### FizzBuzz

#### def fizzBuzz(n):

#### result = []

#### for i in range(1, n + 1):

#### if i % 3 == 0 and i % 5 == 0:

#### result.append("FizzBuzz")

#### elif i % 3 == 0:

#### result.append("Fizz")

#### elif i % 5 == 0:

#### result.append("Buzz")

#### else:

#### result.append(str(i))

#### return result

#### if \_\_name\_\_ == "\_\_main\_\_":

#### n = int(input())

#### output = fizzBuzz(n)

#### print(output)

#### Sort Array in Wave Form

def wave\_sort(arr):

arr.sort()

for i in range(0, len(arr) - 1, 2):

arr[i], arr[i + 1] = arr[i + 1], arr[i]

print(\*arr,sep="\n")

n = int(input())

arr = [int(input()) for \_ in range(n)]

wave\_sort(arr)

#### Intersection of Two Arrays

def intersection(nums1, nums2):

return list(set(nums1) & set(nums2))

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input())

nums1 = [int(input()) for \_ in range(n)]

m = int(input())

nums2 = [int(input()) for \_ in range(m)]

result = intersection(nums1, nums2)

print(sorted(result)) # Sorting to maintain consistent output format

#### Convert Integer to Excel Column Title

def convertToTitle(columnNumber):

result = []

while columnNumber > 0:

columnNumber -= 1

result.append(chr(columnNumber % 26 + ord('A')))

columnNumber //= 26

return ''.join(result[::-1])

if \_\_name\_\_ == "\_\_main\_\_":

columnNumber = int(input())

print(convertToTitle(columnNumber))

#### Find the Middle of a Linked List

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def findMiddle(head):

slow = fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

return slow.val

def createLinkedList(values):

if not values:

return None

head = ListNode(values[0])

current = head

for val in values[1:]:

current.next = ListNode(val)

current = current.next

return head

if \_\_name\_\_ == "\_\_main\_\_":

values = list(map(int, input().split()))

head = createLinkedList(values)

print(findMiddle(head))

#### Find the First Unique Character in a String

def firstUniqChar(s):

    char\_count = {}

    for char in s:

        if char in char\_count:

            char\_count[char] += 1

        else:

            char\_count[char] = 1

    for i in range(len(s)):

        if char\_count[s[i]] == 1:

            return i

    return -1

input\_string = input().strip()

print(firstUniqChar(input\_string))

#### scrambled string

def isScramble(s1, s2):

    memo = {}

    def helper(s1, s2):

        if (s1, s2) in memo:

            return memo[(s1, s2)]

        if s1 == s2:

            memo[(s1, s2)] = True

            return True

        if sorted(s1) != sorted(s2):

            memo[(s1, s2)] = False

            return False

        n = len(s1)

        for i in range(1, n):

            if helper(s1[:i], s2[:i]) and helper(s1[i:], s2[i:]):

                memo[(s1, s2)] = True

                return True

            if helper(s1[:i], s2[-i:]) and helper(s1[i:], s2[:-i]):

                memo[(s1, s2)] = True

                return True

        memo[(s1, s2)] = False

        return False

    return helper(s1, s2)

s1 = input().strip()

s2 = input().strip()

print(isScramble(s1, s2))

#### Wiggle Sort

#include <stdio.h>

void wiggleSort(int nums[], int n) {

    for (int i = 1; i < n; i++) {

        if ((i % 2 == 1 && nums[i] < nums[i - 1]) ||

            (i % 2 == 0 && nums[i] > nums[i - 1])) {

            int temp = nums[i];

            nums[i] = nums[i - 1];

            nums[i - 1] = temp;

        }

    }

}

int main() {

    int n;

    scanf("%d", &n);

    int nums[n];

    for (int i = 0; i < n; i++) {

        scanf("%d", &nums[i]);

    }

    wiggleSort(nums, n);

    for (int i = 0; i < n; i++) {

        printf("%d ", nums[i]);

    }

    printf("\n");

    return 0;

}

#### Count Inversions in an Array

def merge\_count\_split\_inv(arr, temp\_arr, left, right):

if left == right:

return 0

mid = (left + right) // 2

inv\_count = merge\_count\_split\_inv(arr, temp\_arr, left, mid)

inv\_count += merge\_count\_split\_inv(arr, temp\_arr, mid + 1, right)

inv\_count += merge\_and\_count(arr, temp\_arr, left, mid, right)

return inv\_count

def merge\_and\_count(arr, temp\_arr, left, mid, right):

i = left # Starting index for left subarray

j = mid + 1 # Starting index for right subarray

k = left # Starting index to be sorted

inv\_count = 0

while i <= mid and j <= right:

if arr[i] <= arr[j]:

temp\_arr[k] = arr[i]

i += 1

else:

temp\_arr[k] = arr[j]

inv\_count += (mid - i + 1)

j += 1

k += 1

while i <= mid:

temp\_arr[k] = arr[i]

i += 1

k += 1

while j <= right:

temp\_arr[k] = arr[j]

j += 1

k += 1

for i in range(left, right + 1):

arr[i] = temp\_arr[i]

return inv\_count

def count\_inversions(arr):

n = len(arr)

temp\_arr = [0] \* n

return merge\_count\_split\_inv(arr, temp\_arr, 0, n - 1)

# Input reading

n = int(input()) # Read the number of elements

arr = [int(input()) for \_ in range(n)] # Read the array elements

# Output the number of inversions

print(count\_inversions(arr))

#### Two Sum – Hash map

#include <stdio.h>

#include <stdlib.h>

// Structure for hash table entries

typedef struct {

int value;

int index;

} HashEntry;

// Simple hash function

int hashFunction(int key, int size) {

return abs(key) % size;

}

// Function to find two sum using hash map

void twoSum(int nums[], int n, int target) {

int hashSize = n \* 2; // Increase size to reduce collisions

HashEntry\* hashTable = (HashEntry\*)calloc(hashSize, sizeof(HashEntry));

for (int i = 0; i < n; i++) {

int complement = target - nums[i];

int hashIndex = hashFunction(complement, hashSize);

if (hashTable[hashIndex].value == complement) {

printf("%d %d\n", hashTable[hashIndex].index, i);

free(hashTable);

return;

}

hashIndex = hashFunction(nums[i], hashSize);

hashTable[hashIndex].value = nums[i];

hashTable[hashIndex].index = i;

}

free(hashTable);

printf("-1 -1\n"); // No valid pair found (shouldn't happen based on constraints)

}

int main() {

int n;

scanf("%d", &n);

int\* nums = (int\*)malloc(n \* sizeof(int));

for (int i = 0; i < n; i++) {

scanf("%d", &nums[i]);

}

int target;

scanf("%d", &target);

twoSum(nums, n, target);

free(nums);

return 0;

}

#### Two Sum – Hash map

import java.util.HashMap;

import java.util.Scanner;

public class TwoSum {

public static int[] twoSum(int[] nums, int target) {

HashMap<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < nums.length; i++) {

int complement = target - nums[i];

if (map.containsKey(complement)) {

return new int[] { map.get(complement), i };

}

map.put(nums[i], i);

}

return new int[] {};

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] nums = new int[n];

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

int target = scanner.nextInt();

int[] result = twoSum(nums, target);

if (result.length > 0) {

System.out.println(result[0] + " " + result[1]);

} else {

System.out.println("No solution");

}

scanner.close();

}

}

#### Delete a Node in a Doubly Linked List

class ListNode:

def \_\_init\_\_(self, val=0, prev=None, next=None):

self.val = val

self.prev = prev

self.next = next

def deleteNode(head, node\_val):

if not head:

return None # Return if the list is empty

current = head

while current:

if current.val == node\_val:

# Case 1: Deleting the head node

if current == head:

head = head.next

if head:

head.prev = None

return head

# Case 2: Deleting a middle or last node

if current.next:

current.next.prev = current.prev

if current.prev:

current.prev.next = current.next

return head # Return the updated head

current = current.next

return head # Return the head if node\_val was not found

def printList(head):

result = []

current = head

while current:

result.append(current.val)

current = current.next

print(result)

def createDoublyLinkedList(values):

if not values:

return None

head = ListNode(values[0])

current = head

for val in values[1:]:

new\_node = ListNode(val)

current.next = new\_node

new\_node.prev = current

current = new\_node

return head

if \_\_name\_\_ == "\_\_main\_\_":

values = list(map(int, input().split()))

node\_val = int(input())

head = createDoublyLinkedList(values)

modified\_head = deleteNode(head, node\_val)

printList(modified\_head)