**1)search in rotated array:**

def search(nums, target):

left, right = 0, len(nums) - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

return mid

# Check if left side is sorted

if nums[left] <= nums[mid]:

if nums[left] <= target < nums[mid]:

right = mid - 1 # Search left half

else:

left = mid + 1 # Search right half

else:

# Right side must be sorted

if nums[mid] < target <= nums[right]:

left = mid + 1 # Search right half

else:

right = mid - 1 # Search left half

return -1 # Target not found

# Input

n = int(input())

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

target = int(input())

# Output

print(search(nums, target))

**2)sum of digits using recursion:**

def sum\_of\_digits(n):

if n == 0:

return 0

return n % 10 + sum\_of\_digits(n // 10)

n = int(input())

print(sum\_of\_digits(n))

**3)** **sort integers by there last digit sorting:**

def sort\_by\_last\_digit(arr):

# Sort the list based on the last digit and then by the value

return sorted(arr, key=lambda x: (x % 10, x))

# Input: First, the number of elements in the list

n = int(input())

# Input: Then, the list of integers

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

# Sort the list and print each number on a new line

sorted\_arr = sort\_by\_last\_digit(arr)

# Print the sorted list

for num in sorted\_arr:

print(num)

**4)** **find the local maximum element searching:**

def find\_peak(arr):

n = len(arr)

# Binary search to find a peak element

left, right = 0, n - 1

while left <= right:

mid = (left + right) // 2

# Check if mid is a peak

if (mid == 0 or arr[mid] >= arr[mid - 1]) and (mid == n - 1 or arr[mid] >= arr[mid + 1]):

return mid

elif mid > 0 and arr[mid - 1] > arr[mid]:

# Peak must be on the left side

right = mid - 1

else:

# Peak must be on the right side

left = mid + 1

return -1 # This should never be hit because there is always a peak.

# Input: First, the number of elements in the list

n = int(input())

# Input: Then, the list of integers

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

# Find and print the index of a peak element

peak\_index = find\_peak(arr)

print(peak\_index)

**5)** **sort strings by number of distinct characters sorting:**

def sort\_strings\_by\_criteria(strings):

# Sort the list using a custom key

return sorted(strings, key=lambda s: (len(set(s)), len(s), s))

# Input: First, the number of strings

n = int(input())

# Input: Then, the list of strings

strings = [input().strip() for \_ in range(n)]

# Sort the strings using the custom sort function

sorted\_strings = sort\_strings\_by\_criteria(strings)

# Output: Print the sorted strings

for string in sorted\_strings:

print(string)

**6) Generate** **fibonacci numbers using recursion:**

def fibonacci(n: int) -> int:

# Base case: return 0 when n is 0, and return 1 when n is 1

if n == 0:

return 0

elif n == 1:

return 1

# Recursive case: return sum of the previous two Fibonacci numbers

else:

return fibonacci(n - 1) + fibonacci(n - 2)

# Input reading

n = int(input())

# Output the n-th Fibonacci number

print(fibonacci(n))

**7)** **remove repeating adjacent letters:**

def removeDuplicates(s: str) -> str:

stack = [] # Stack to keep track of characters

for char in s:

# If the stack is not empty and the top of the stack matches the current character, pop the stack

if stack and stack[-1] == char:

stack.pop()

else:

stack.append(char)

# The final result is the characters remaining in the stack

return ''.join(stack)

# Input reading

s = input()

# Output the final string after removing duplicates

print(removeDuplicates(s))

**8)** **find first unique character:**

def firstUniqChar(s: str) -> int:

# Create a dictionary to store the frequency of each character

freq = {}

# First pass: build the frequency map

for char in s:

if char in freq:

freq[char] += 1

else:

freq[char] = 1

# Second pass: find the first character with a frequency of 1

for i, char in enumerate(s):

if freq[char] == 1:

return i # Return the index of the first non-repeating character

return -1 # Return -1 if no non-repeating character exists

# Input reading

s = input()

# Output the index of the first non-repeating character

print(firstUniqChar(s))

#### **9)** **Sort 2D Points by Distance from Origin – sorting:**

def sort\_points(points):

# Sorting the points by their Manhattan distance, then by x, then by y

return sorted(points, key=lambda point: (abs(point[0]) + abs(point[1]), point[0], point[1]))

# Input reading

n = int(input()) # Number of points

points = []

for \_ in range(n):

x, y = map(int, input().split())

points.append((x, y))

# Sort the points

sorted\_points = sort\_points(points)

# Output the sorted points

for point in sorted\_points:

print(point[0], point[1])

**10)** **reverse a string using recursion:**

def reverse\_string(s: str) -> str:

# Base case: if the string is empty or has one character, return it as is.

if len(s) == 0:

return s

else:

# Recursive case: reverse the rest of the string and add the first character at the end

return reverse\_string(s[1:]) + s[0]

# Input reading

s = input()

# Output the reversed string

print(reverse\_string(s))

**11)** **find the first missing positive integer:**

def firstMissingPositive(nums):

n = len(nums)

# Step 1: Reorganize numbers such that each number is placed at its correct index

for i in range(n):

while 1 <= nums[i] <= n and nums[nums[i] - 1] != nums[i]:

# Swap nums[i] with nums[nums[i] - 1]

nums[nums[i] - 1], nums[i] = nums[i], nums[nums[i] - 1]

# Step 2: Find the first index where nums[i] != i + 1

for i in range(n):

if nums[i] != i + 1:

return i + 1

# Step 3: If all numbers from 1 to n are present, return n + 1

return n + 1

# Input reading

n = int(input()) # Number of elements

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

# Output the smallest missing positive integer

print(firstMissingPositive(nums))

**12)** **blanket validator:**

def isValid(s: str) -> str:

stack = []

# Dictionary to match the closing brackets with the corresponding opening brackets

matching\_brackets = {')': '(', '}': '{', ']': '['}

# Iterate through each character in the string

for char in s:

if char in matching\_brackets.values():

# If it's an opening bracket, push to stack

stack.append(char)

elif char in matching\_brackets:

# If it's a closing bracket, check if stack is empty or top of stack doesn't match

if stack and stack[-1] == matching\_brackets[char]:

stack.pop() # Pop the matching opening bracket from the stack

else:

return "NO"

# If stack is empty, all brackets were properly matched

return "YES" if not stack else "NO"

# Input reading

s = input()

# Output the result of the validity check

print(isValid(s))