**1)Write a program to Print Fibonacci Series using recursion**.

def fibonacci(n):

if n<=0:

return 0

elif n==1:

return 1

else:

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

def print\_fibonacci\_series(count):

for i in range(count):

print(fibonacci(i),end=' ,')

n\_terms=10

print\_fibonacci\_series(n\_terms)

**OUTPUT:** 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

**2.Write a program to check the given no is Armstrong or not using recursive function.**

num = 1634

order = len(str(num))

sum = 0

temp = num

while temp > 0:

digit = temp % 10

sum += digit \*\* order

temp //= 10

if num == sum:

print(num,"is an Armstrong number")

else:

print(num,"is not an Armstrong number")

**OUTPUT:** 1634 is an Armstrong number

**3.Write a program to find the GCD of two numbers using recursive factorization**

import math

print("The gcd of 60 and 48 is : ", end="")

print(math.gcd(60, 48))

**OUTPUT:** The gcd of 60 and 48 is : 12

**4.Write a program to get the largest element of an array.**

def find\_largest\_element(arr):

if not arr:

return None

return max(arr)

array = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

largest\_element = find\_largest\_element(array)

print(f"The largest element in the array is: {largest\_element}")

**OUTPUT:** The largest element in the array is: 9

**5.Write a program to find the Factorial of a number using recursion.**

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

number = 5

print(f"The factorial of {number} is {factorial(number)}")

**OUTPUT:** The factorial of 5 is 120

**6. Write a program for to copy one string to another using recursion**

def copy\_string\_recursively(source, index=0)

if index == len(source):

return ""

else:

return source[index] + copy\_string\_recursively(source, index + 1)

original\_string = "Hello, World!"

copied\_string = copy\_string\_recursively(original\_string)

print("Copied String:", copied\_string)

**OUTPUT:** Copied String: Hello, World!

**7. Write a program to print the reverse of a string using recursion**

def reverse(s):

str = ""

for i in s:

str = i + str

return str

s ="APPLE"

print("The original string is : ", end="")

print(s)

print("The reversed string(using loops) is : ", end="")

print(reverse(s))

**OUTPUT:** The original string is : APPLE

The reversed string(using loops) is : ELPPA

**8. Write a program to generate all the prime numbers using recursion.**

def Prime\_Number(n, i=2):

if n == i:

return True

elif n % i == 0:

return False

return Prime\_Number(n, i + 1)

n = 971

if Prime\_Number(n):

print("Yes,", n, "is Prime")

else:

print("No,", n, "is not a Prime")

**OUTPUT:** Yes, 971 is Prime

**9. Write a program to check a number is a prime number or not using recursion.**

def is\_prime(n, divisor=None):

if n <= 1:

return False

if divisor is None:

divisor = n - 1

if divisor == 1:

return True

if n % divisor == 0:

return False

return is\_prime(n, divisor - 1)

number = 29

if is\_prime(number):

print(f"{number} is a prime number.")

else:

print(f"{number} is not a prime number.")

OUTPUT: 29 is a prime number.

**10)Write a program to find the reverse of a given number using recursive**.

def reverse\_number(n,reversed\_num=0):

if n==0:

return reversed\_num

else:

last\_digit=n%10

reversed\_num=reversed\_num\*10+last\_digit

return reverse\_number(n//10,reversed\_num)

number=12345

reversed\_number=reverse\_number(number)

print(f"The reverse of {number} is {reversed\_number}.")

OUTPUT: The reverse of 12345 is 54321.

**11) Write a program to find the perfect number.**

def find\_perfect\_numbers(limit):

perfect\_numbers=[]

for num in range(1,limit+1):

divisors\_sum=sum([divisor for divisor in range(1,num)if num%divisor==0])

if divisors\_sum==num:

perfect\_numbers.append(num)

return perfect\_numbers

limit=10000

perfect\_nums=find\_perfect\_numbers(limit)

print("Perfect numbers up to",limit,"are:",perfect\_nums)

OUTPUT: Perfect numbers up to 10000 are: [6, 28, 496, 8128]

**12) Write C program that demonstrates the usage of these notations by analyzing the time complexity of some example algorithms.**

#include <stdio.h>

// O(n) - Linear Time Complexity

void linearTime(int n) {

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

printf("%d ", i);

}

}

// O(n^2) - Quadratic Time Complexity

void quadraticTime(int n) {

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

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

printf("%d ", i \* j);

}

}

}

// O(1) - Constant Time Complexity

void constantTime() {

printf("This runs in constant time.\n");

}

int main() {

int n = 5;

printf("Linear Time Complexity:\n");

linearTime(n);

printf("\nQuadratic Time Complexity:\n");

quadraticTime(n);

printf("\nConstant Time Complexity:\n");

constantTime();

return 0;

}

**13) Write C programs that demonstrate the mathematical analysis of non-recursive and recursive algorithms**.

#include <stdio.h>

unsigned long long factorial\_recursive(int n) {

if (n == 0) {

return 1;

}

return n \* factorial\_recursive(n - 1);

}

**14) Write C programs for solving recurrence relations using the Master Theorem, Substitution Method, and Iteration Method will demonstrate how to calculate the time complexity of an example recurrence relation using the specified technique**.

#include <stdio.h>

#include <math.h>

void master\_theorem() {

int a = 2;

int b = 2;

int n\_power = 1;

printf("Using Master Theorem:\n");

printf("Given T(n) = %dT(n/%d) + n\n", a, b);

printf("Comparing f(n) = n^%d with n^log\_b(a) = n^%.1f\n", n\_power, log\_base);

if (n\_power < log\_base) {

printf("T(n) = Theta(n^%.1f)\n", log\_base);

} else if (n\_power == log\_base) {

printf("T(n) = Theta(n^%.1f \* log(n))\n", log\_base);

} else {

printf("T(n) = Theta(n^%d)\n", n\_power);

}

}

int main() {

master\_theorem();

return 0;

}

**15)Given two integer arrays nums1 and nums2, return an array of their Intersection. Each element in the result must be unique and you may return the result in any order.**

def intersection(nums1, nums2):

set1 = set(nums1)

set2 = set(nums2)

return list(set1 & set2)

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

result = intersection(nums1, nums2)

print(result)

**16)Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.**

from collections import Counter

def intersect(nums1, nums2):

count1 = Counter(nums1)

count2 = Counter(nums2)

intersection = []

for num in count1:

if num in count2:

intersection.extend([num] \* min(count1[num], count2[num]))

return intersection

# Example usage

nums1 = [1, 2, 2, 1]

nums2 = [2, 2]

result = intersect(nums1, nums2)

print(result) # Output: [2, 2]

**17) Given an array of integers nums, sort the array in ascending order and return it.You must solve the problem without using any built-in functions in O(nlog(n)) time complexity and with the smallest space complexity possible.**

#include <stdio.h>

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

int nums[] = {34, 7, 23, 32, 5, 62};

int n = sizeof(nums) / sizeof(nums[0]);

quickSort(nums, 0, n - 1);

printf("Sorted array: ");

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

printf("%d ", nums[i]);

}

return 0;

}

**18) Given an array of integers nums, half of the integers in nums are odd, and the other half are even**.

#include <stdio.h>

void splitArray(int nums[], int size) {

printf("Odd Numbers: ");

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

if (nums[i] % 2 != 0) {

printf("%d ", nums[i]);

}

}

printf("\nEven Numbers: ");

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

if (nums[i] % 2 == 0) {

printf("%d ", nums[i]);

}

}

}

int main() {

int nums[] = {1, 2, 3, 4, 5, 6};

int size = sizeof(nums) / sizeof(nums[0]);

splitArray(nums, size);

return 0;

}

**19) Sort the array so that whenever nums[i] is odd, and whenever nums[i] is even, i is even. Return any answer array that satisfies this condition.**

def sort\_array\_by\_parity\_ii(nums):

odd = [num for num in nums if num % 2 != 0]

even = [num for num in nums if num % 2 == 0]

result = [0] \* len(nums)

result[::2] = even # Place even numbers at even indices

result[1::2] = odd # Place odd numbers at odd indices

return result

# Example usage

nums = [4, 2, 5, 7]

sorted\_array = sort\_array\_by\_parity\_ii(nums)

print(sorted\_array) # Output could be [4, 5, 2, 7] or any valid arrangement

**20)You are given a string s, and an array of pairs of indices in the string pairs where pairs[i] = [a, b] indicates 2 indices(0-indexed) of the string.You can swap the characters at any pair of indices in the given pairs any number of times. Return the lexicographically smallest string that s can be changed to after using the swaps.**

def smallestStringWithSwaps(s: str, pairs: List[List[int]]) -> str:

from collections import defaultdict

# Union-Find structure

parent = list(range(len(s)))

def find(x):

if parent[x] != x:

parent[x] = find(parent[x])

return parent[x]

def union(x, y):

parent[find(x)] = find(y)

# Create unions based on pairs

for a, b in pairs:

union(a, b)

# Group characters by their root parent

groups = defaultdict(list)

for i in range(len(s)):

root = find(i)

groups[root].append(i)

# Create a list of characters and sort them

result = list(s)

for indices in groups.values():

chars = sorted(result[i] for i in indices)

for i, char in zip(sorted(indices), chars):

result[i] = char

return ''.join(result)

# Example usage

s = "dcab"

pairs = [[0, 3], [1, 2]]

print(smallestStringWithSwaps(s, pairs)) # Output: "abcd"

**21)Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1.**

def can\_break(s1, s2):

# Sort both strings

sorted\_s1 = sorted(s1)

sorted\_s2 = sorted(s2)

# Check if sorted\_s1 can break sorted\_s2

s1\_breaks\_s2 = all(a >= b for a, b in zip(sorted\_s1, sorted\_s2))

# Check if sorted\_s2 can break sorted\_s1

s2\_breaks\_s1 = all(a >= b for a, b in zip(sorted\_s2, sorted\_s1))

return s1\_breaks\_s2 or s2\_breaks\_s1

# Example usage

s1 = "abc"

s2 = "cba"

print(can\_break(s1, s2)) # Output: True

**22)You are given a string s. s[i] is either a lowercase English letter or '?'. For a string t having length m containing only lowercase English letters, we define the function cost(i) for an index i as the number of characters equal to t[i] that appeared before it, i.e. in the range [0, i - 1]. The value of t is the sum of cost(i) for all indices i. For example, for the string t = "aab":**

**cost(0) = 0**

**cost(1) = 1**

**cost(2) = 0**

**Hence, the value of "aab" is 0 + 1 + 0 = 1. Your task is to replace all occurrences of '?' in s with any lowercase English letter so at the value of s is minimized.**

def calculate\_cost(t):

cost = 0

count = {}

for i in range(len(t)):

if t[i] in count:

cost += count[t[i]]

count[t[i]] += 1

else:

count[t[i]] = 1

return cost

# Example usage

t = "aab"

print(calculate\_cost(t)) # Output: 1

**23)You are given a string s. Consider performing the following operation until s becomes empty: For every alphabet character from 'a' to 'z', remove the first occurrence of that character in s (if it exists). For example, let initially s = "aabcbbca". We do the following operations: Remove the underlined characters s = "aabcbbca". The resulting string is s = "abbca". Remove the underlined characters s = "abbca". The resulting string is s = "ba". Remove the underlined characters s = "ba". The resulting** **string is s = "". Return the value of the string s right before applying the last operation. In the example above, answer is "ba".**

def last\_string\_before\_empty(s):

for char in range(ord('z'), ord('a') - 1, -1):

char = chr(char)

if char in s:

s = s[:s.index(char)] + s[s.index(char) + 1:]

return s

# Example

initial\_s = "aabcbbca"

result = last\_string\_before\_empty(initial\_s)

print(result) # Output: "ba"

24)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]

def max\_subarray\_sum(nums):

max\_sum = current\_sum = nums[0]

for num in nums[1:]:

current\_sum = max(num, current\_sum + num)

max\_sum = max(max\_sum, current\_sum)

return max\_sum

# Example usage

nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

result = max\_subarray\_sum(nums)

print(result) # Output: 6

Output: 6

Explanation: The subarray [4,-1,2,1] has the largest sum 6.

**25)You are given an integer array nums with no duplicates. A maximum binary tree can be built recursively from nums using the following algorithm: Create a root node whose value is the maximum value in nums. Recursively build the left subtree on the subarray prefix to the left of the maximum value. Recursively build the right subtree on the subarray suffix to the right of the maximum value. Return the maximum binary tree built from nums.**

class TreeNode:

def \_\_init\_\_(self, val):

self.val = val

self.left = None

self.right = None

def constructMaximumBinaryTree(nums):

if not nums:

return None

max\_val = max(nums)

max\_index = nums.index(max\_val)

root = TreeNode(max\_val)

root.left = constructMaximumBinaryTree(nums[:max\_index])

root.right = constructMaximumBinaryTree(nums[max\_index + 1:])

return root

**26)Given a circular integer array nums of length n, return the maximum possible sum of a non-empty subarray of nums.A circular array means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 1 + n) % n].A subarray may only include each element of the fixed buffer nums at most once. Formally, for a subarray nums[i], nums[i + 1], ..., nums[j], there does not exist i <= k1, k2 <= j with k1 % n == k2 % n.**

def maxSubarraySumCircular(nums):

def kadane(arr):

max\_sum = current\_sum = arr[0]

for num in arr[1:]:

current\_sum = max(num, current\_sum + num)

max\_sum = max(max\_sum, current\_sum)

return max\_sum

total\_sum = sum(nums)

max\_kadane = kadane(nums)

min\_kadane = kadane([-num for num in nums])

if total\_sum == min\_kadane: # All numbers are negative

return max\_kadane

return max(max\_kadane, total\_sum + min\_kadane)

# Example usage

nums = [1, -2, 3, -2]

print(maxSubarraySumCircular(nums)) # Output: 3

**27)You are given an array nums consisting of integers. You are also given a 2D array queries, where queries[i] = [posi, xi].For query i, we first set nums[posi] equal to xi, then we calculate the answer to query i which is the maximum sum of a subsequence of nums where no two adjacent elements are selected. Return the sum of the answers to all queries. Since the final answer may be very large, return it modulo 109 + 7. A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.**

def max\_sum\_non\_adjacent(nums, queries):

MOD = 10\*\*9 + 7

def max\_non\_adjacent\_sum(arr):

incl = 0

excl = 0

for num in arr:

new\_excl = max(incl, excl)

incl = excl + num

excl = new\_excl

return max(incl, excl) % MOD

total\_sum = 0

for pos, x in queries:

nums[pos] = x

total\_sum = (total\_sum + max\_non\_adjacent\_sum(nums)) % MOD

return total\_sum

# Example usage

nums = [1, 2, 3, 1]

queries = [[1, 3], [0, 2]]

result = max\_sum\_non\_adjacent\_sum(nums, queries)

print(result) # Output the final result

**28) Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).The distance between two points on the X-Y plane is the Euclidean distance (i.e., √(x1 - x2)2 + (y1 - y2)2). You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).**

import heapq

def kClosest(points, k):

# Calculate the squared distance and use a min-heap to find the k closest points

return heapq.nsmallest(k, points, key=lambda point: point[0]\*\*2 + point[1]\*\*2)

# Example usage

points = [[1, 3], [-2, 2], [5, 8], [0, 1]]

k = 2

closest\_points = kClosest(points, k)

print(closest\_points) # Output: [[-2, 2], [0, 1]]

**29) Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log** **(m+n)).**

def findMedianSortedArrays(nums1, nums2):

if len(nums1) > len(nums2):

nums1, nums2 = nums2, nums1

x, y = len(nums1), len(nums2)

low, high = 0, x

while low <= high:

partitionX = (low + high) // 2

partitionY = (x + y + 1) // 2 - partitionX

maxX = float('-inf') if partitionX == 0 else nums1[partitionX - 1]

minX = float('inf') if partitionX == x else nums1[partitionX]

maxY = float('-inf') if partitionY == 0 else nums2[partitionY - 1]

minY = float('inf') if partitionY == y else nums2[partitionY]

if maxX <= minY and maxY <= minX:

if (x + y) % 2 == 0:

return (max(maxX, maxY) + min(minX, minY)) / 2

else:

return max(maxX, maxY)

elif maxX > minY:

high = partitionX - 1

else:

low = partitionX + 1

raise ValueError("Input arrays are not sorted.")