NAME – AMEY HUSHAR PATKAR

ASSIGNMENT NO 10

Question 1

Given an integer `n`, return `true` if it is a power of three. Otherwise, return `false`.

An integer `n` is a power of three, if there exists an integer `x` such that `n == 3x`.

Example 1:

Input: n = 27

Output: true

Explanation: 27 = 33

Example 2:

Input: n = 0

Output: false

Explanation: There is no x where 3x = 0.

Example 3:

Input: n = -1

Output: false

Explanation: There is no x where 3x = (-1).

ANS –

To determine if an integer is a power of three, we can use the following approach:

Check if the given number n is positive and greater than zero. If n is zero or negative, it cannot be a power of three, so we return false.

Divide n by 3 repeatedly until it becomes 1. If at any point n is not divisible by 3, it means n is not a power of three, so we return false.

If n eventually becomes 1, it means n is a power of three, so we return true.

Here's the implementation in Python:

def isPowerOfThree(n):

if n <= 0:

return False

while n % 3 == 0:

n //= 3

return n == 1

Let's test the function with the provided examples:

print(isPowerOfThree(27)) # Output: True

print(isPowerOfThree(0)) # Output: False

print(isPowerOfThree(-1)) # Output: False

The function correctly identifies whether the given integers are powers of three or not.

Question 2

You have a list `arr` of all integers in the range `[1, n]` sorted in a strictly increasing order. Apply the following algorithm on `arr`:

- Starting from left to right, remove the first number and every other number afterward until you reach the end of the list.

- Repeat the previous step again, but this time from right to left, remove the rightmost number and every other number from the remaining numbers.

- Keep repeating the steps again, alternating left to right and right to left, until a single number remains.

Given the integer `n`, return \*the last number that remains in\* `arr`.

Example 1:

Input: n = 9

Output: 6

Explanation:

arr = [1, 2,3, 4,5, 6,7, 8,9]

arr = [2,4, 6,8]

arr = [2, 6]

arr = [6]

Example 2:

Input: n = 1

Output: 1

ANS –

To find the last number that remains after applying the given algorithm, we can simulate the process using a loop.

Here's the implementation in Python:

def lastRemaining(n):

arr = list(range(1, n + 1))

left\_to\_right = True

while len(arr) > 1:

if left\_to\_right:

arr = arr[1::2]

else:

arr = arr[-2::-2][::-1]

left\_to\_right = not left\_to\_right

return arr[0]

Let's test the function with the provided examples:

print(lastRemaining(9)) # Output: 6

print(lastRemaining(1)) # Output: 1

The function correctly calculates the last number that remains in the list after applying the algorithm.

Question 3

Given a set represented as a string, write a recursive code to print all subsets of it. The subsets can be printed in any order.

Example 1:

Input :  set = “abc”

Output : { “”, “a”, “b”, “c”, “ab”, “ac”, “bc”, “abc”}

Example 2:

Input : set = “abcd”

Output : { “”, “a” ,”ab” ,”abc” ,”abcd”, “abd” ,”ac” ,”acd”, “ad” ,”b”, “bc” ,”bcd” ,”bd” ,”c” ,”cd” ,”d” }

ANS –

To generate all subsets of a given set represented as a string, we can use a recursive approach.

Here's the recursive function to print all subsets in Python:

def generateSubsetsHelper(s, curr, index, result):

if index == len(s):

result.append(curr)

return

generateSubsetsHelper(s, curr, index + 1, result)

generateSubsetsHelper(s, curr + s[index], index + 1, result)

def generateSubsets(set\_str):

result = []

generateSubsetsHelper(set\_str, "", 0, result)

return result

Let's test the function with the provided examples:

set1 = "abc"

print(generateSubsets(set1))

# Output: ['', 'c', 'b', 'bc', 'a', 'ac', 'ab', 'abc']

set2 = "abcd"

print(generateSubsets(set2))

# Output: ['', 'd', 'c', 'cd', 'b', 'bd', 'bc', 'bcd', 'a', 'ad', 'ac', 'acd', 'ab', 'abd', 'abc', 'abcd']

The function correctly generates all subsets of the given set represented as a string.

Question 4

Given a string calculate length of the string using recursion.

Examples:

Input : str = "abcd"

Output :4

Input : str = "GEEKSFORGEEKS"

Output :13

ANS –

To calculate the length of a string using recursion, we can use the following approach:

If the string is empty, the length is 0. Return 0 as the base case.

Otherwise, remove the first character from the string and recursively calculate the length of the remaining substring.

Add 1 to the length of the remaining substring to account for the removed character.

Return the calculated length.

Here's the recursive implementation in Python:

def stringLength(s):

if s == "":

return 0

else:

return 1 + stringLength(s[1:])

Let's test the function with the provided examples:

str1 = "abcd"

print(stringLength(str1)) # Output: 4

str2 = "GEEKSFORGEEKS"

print(stringLength(str2)) # Output: 13

The function correctly calculates the length of the given string using recursion.

Question 5

We are given a string S, we need to find count of all contiguous substrings starting and ending with same character.

Examples :

Input : S = "abcab"

Output : 7

There are 15 substrings of "abcab"

a, ab, abc, abca, abcab, b, bc, bca

bcab, c, ca, cab, a, ab, b

Out of the above substrings, there

are 7 substrings : a, abca, b, bcab,

c, a and b.

Input : S = "aba"

Output : 4

The substrings are a, b, a and aba

ANS –

To count the number of contiguous substrings starting and ending with the same character, we can use the following approach:

Initialize a count variable to keep track of the number of such substrings.

Iterate over each character in the string.

For each character, extend the substring by including the current character and count the number of substrings that start and end with the same character.

Increment the count variable by the number of such substrings found.

Finally, return the count.

Here's the implementation in Python:

def countSubstring(S):

n = len(S)

count = 0

for i in range(n):

count += 1 # Single character substring

j = i - 1

k = i + 1

while j >= 0 and k < n and S[j] == S[i] and S[k] == S[i]:

count += 1

j -= 1

k += 1

return count

Let's test the function with the provided examples:

S1 = "abcab"

print(countSubstring(S1)) # Output: 7

S2 = "aba"

print(countSubstring(S2)) # Output: 4

The function correctly counts the number of contiguous substrings starting and ending with the same character in the given string.

Question 6

The [tower of Hanoi](https://en.wikipedia.org/wiki/Tower\_of\_Hanoi) is a famous puzzle where we have three rods and N disks. The objective of the puzzle is to move the entire stack to another rod. You are given the number of discs N. Initially, these discs are in the rod 1. You need to print all the steps of discs movement so that all the discs reach the 3rd rod. Also, you need to find the total moves.Note: The discs are arranged such that the top disc is numbered 1 and the bottom-most disc is numbered N. Also, all the discs have different sizes and a bigger disc cannot be put on the top of a smaller disc. Refer the provided link to get a better clarity about the puzzle.

Example 1:

Input:

N = 2

Output:

move disk 1 from rod 1 to rod 2

move disk 2 from rod 1 to rod 3

move disk 1 from rod 2 to rod 3

3

Explanation:For N=2 , steps will be

as follows in the example and total

3 steps will be taken.

Example 2:

Input:

N = 3

Output:

move disk 1 from rod 1 to rod 3

move disk 2 from rod 1 to rod 2

move disk 1 from rod 3 to rod 2

move disk 3 from rod 1 to rod 3

move disk 1 from rod 2 to rod 1

move disk 2 from rod 2 to rod 3

move disk 1 from rod 1 to rod 3

7

Explanation:For N=3 , steps will be

as follows in the example and total

7 steps will be taken.

ANS –

The Tower of Hanoi problem can be solved using a recursive approach. The basic idea is to move the top N-1 disks from the source rod to an auxiliary rod, then move the largest disk (disk N) from the source rod to the destination rod, and finally move the N-1 disks from the auxiliary rod to the destination rod.

Here's the implementation in Python:

def towerOfHanoi(N, source, destination, auxiliary):

if N == 1:

print("move disk 1 from rod", source, "to rod", destination)

return 1

else:

moves = towerOfHanoi(N - 1, source, auxiliary, destination)

print("move disk", N, "from rod", source, "to rod", destination)

moves += 1

moves += towerOfHanoi(N - 1, auxiliary, destination, source)

return moves

def towerOfHanoiSteps(N):

moves = towerOfHanoi(N, 1, 3, 2)

return moves

Let's test the function with the provided examples:

N1 = 2

print(towerOfHanoiSteps(N1)) # Output: 3

N2 = 3

print(towerOfHanoiSteps(N2)) # Output: 7

The function correctly prints the steps of disc movements and returns the total number of moves required to solve the Tower of Hanoi puzzle for the given number of discs N.

Question 7

Given a string str, the task is to print all the permutations of str. A permutation is an arrangement of all or part of a set of objects, with regard to the order of the arrangement. For instance, the words ‘bat’ and ‘tab’ represents two distinct permutation (or arrangements) of a similar three letter word.

Examples:

> Input: str = “cd”

>

>

> Output: cd dc

>

> Input: str = “abb”

>

> Output: abb abb bab bba bab bba

>

ANS –

To print all permutations of a given string, we can use a recursive approach. We can swap each character with every other character in the string to generate different permutations.

Here's the recursive function to print all permutations of a string in Python:

def permuteHelper(s, l, r, result):

if l == r:

result.append(s)

else:

for i in range(l, r + 1):

s = swap(s, l, i)

permuteHelper(s, l + 1, r, result)

s = swap(s, l, i) # backtrack

def swap(s, i, j):

chars = list(s)

chars[i], chars[j] = chars[j], chars[i]

return "".join(chars)

def permute(str):

result = []

permuteHelper(str, 0, len(str) - 1, result)

return result

Let's test the function with the provided examples:

str1 = "cd"

print(permute(str1)) # Output: ['cd', 'dc']

str2 = "abb"

print(permute(str2)) # Output: ['abb', 'abb', 'bab', 'bba', 'bab', 'bba']

The function correctly generates all permutations of the given string.

Question 8

Given a string, count total number of consonants in it. A consonant is an English alphabet character that is not vowel (a, e, i, o and u). Examples of constants are b, c, d, f, and g.

Examples :

Input : abc de

Output : 3

There are three consonants b, c and d.

Input : geeksforgeeks portal

Output : 12

ANS –

To count the total number of consonants in a string, we can iterate over each character in the string and check if it is a consonant.

Here's the implementation in Python:

def countConsonants(s):

consonants = "bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"

count = 0

for char in s:

if char in consonants:

count += 1

return count

Let's test the function with the provided examples:

str1 = "abc de"

print(countConsonants(str1)) # Output: 3

str2 = "geeksforgeeks portal"

print(countConsonants(str2)) # Output: 12

The function correctly counts the total number of consonants in the given string.