**Saveetha School of Engineering**

**Saveetha Institute of Medical And Technical Science**

**ASSIGNMENT-03**

**PROGRAMMING LANGUAGE**

Python

**COURSE CODE / NAME**

CSA0666 - Design And Analysis Of Algorithm For Divide And Conquer Techniques

**SUBMITTED BY**

T Svetha – 192321155

1. **Smallest string lexographically**

from collections import defaultdict

def smallestStringWithSwaps(s, pairs):

def find(x):

if parent[x] != x:

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

return parent[x]

def union(x, y):

rootX = find(x)

rootY = find(y)

if rootX != rootY:

parent[rootY] = rootX

n = len(s)

parent = list(range(n))

for a, b in pairs:

union(a, b)

groups = defaultdict(list)

for i in range(n):

root = find(i)

groups[root].append(i)

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)

s = "dcab"

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

print(smallestStringWithSwaps(s, pairs))

**Output:** "bacd"

1. **Break s1,s2**

def checkIfCanBreak(s1, s2):

s1 = sorted(s1)

s2 = sorted(s2)

def canBreak(x, y):

return all(a >= b for a, b in zip(x, y))

return canBreak(s1, s2) or canBreak(s2, s1)

s1 = "abc"

s2 = "xya"

print(checkIfCanBreak(s1, s2))

**Output:** True

1. **Min cost**

def calculate\_value(t):

counts = defaultdict(int)

cost = 0

for c in t:

cost += counts[c]

counts[c] += 1

return cost

def dfs(i):

if i == len(s):

return 0

if s[i] != '?':

return dfs(i + 1)

min\_cost = float('inf')

for c in 'abcdefghijklmnopqrstuvwxyz':

s[i] = c

min\_cost = min(min\_cost, calculate\_value(s[:i+1]) + dfs(i + 1))

s[i] = '?'

return min\_cost

s = list(s)

return dfs(0)

s = "a?b?c"

print(minimizeCost(s))

**Output**: minimized cost

1. **First string before last operation**

def find(s):

import collections

s = list(s)

while s:

counter = collections.Counter(s)

for char in 'abcdefghijklmnopqrstuvwxyz':

if char in counter:

s.remove(char)

if len(set(s)) == len(s):

break

return ''.join(s)

s = "aabcbbca"

print(find(s))

**Output:** "ba"

1. **Sub array max**

def maxSubArray(nums):

current\_sum = max\_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

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

print(maxSubArray(nums))

**Output:** 6

1. **Binary tree**

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def construct (nums):

if not nums:

return None

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

root = TreeNode(nums[max\_index])

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

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

return root

nums = [3, 2, 1, 6, 0, 5]

root = construct (nums)

**Output:** 6 3 2 1 5 0

1. **Sum sub array**

def max (nums):

def k(nums):

current = max = nums[0]

for num in nums[1:]:

current = max(num, current + num)

max = max(max, current)

return max

total = sum(nums)

maxK = k (nums)

nums = [-num for num in nums]

maxW = total + k (nums)

return max(maxK, maxW) if maxW!= 0 else maxK

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

print(maxSubarraySumCircular(nums))

**Output:** 3

1. **MAX SUM**

def max (nums, queries):

def maxSum(nums):

incl, excl = 0, 0

for num in nums:

new\_excl = max(excl, incl)

incl = excl + num

excl = new\_excl

return max(incl, excl)

total\_sum = 0

for pos, xi in queries:

nums[pos] = xi

total\_sum += maxSum(nums)

return total\_sum % (10\*\*9 + 7)

nums = [1,2,3,4]

queries = [[0,2],[1,3],[2,4]]

print(max (nums, queries))

**Output:** sum of answe**rs**

1. **Closest**

import heapq

def kClosest(points, k):

points.sort(key=lambda point: point[0]\*2 + point[1]\*2)

return points[:k]

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

k = 1

print(kClosest(points, k))

**Output:** [[-2,2]]

**10.Median Sorted Array**

def find (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

nums1 = [1, 3]

nums2 = [2]

print(find (nums1, nums2)) # Output: 2.0