Qno.1)

import numpy as np

# Define the Node class

class Node:

def \_\_init\_\_(self, data, idx):

self.data = data

self.idx = idx

# Define the linked list as an array of Node objects

elm0 = Node("Head", 1)

elm1 = Node("S", 2)

elm2 = Node("t", 3)

elm3 = Node("i", 4)

elm4 = Node("n", 5)

elm5 = Node("g", None)

ls = np.array([elm0, elm1, elm2, elm3, elm4, elm5])

# Define the insertElem function to insert an element at a specified index

def insertElem(ls, idx, char):

# Create a new Node object with the given char and index

new\_node = Node(char, None)

# Find the Node object at the given index

curr\_node = ls[idx-1]

# Link the new Node object to the next Node object

new\_node.idx = curr\_node.idx

# Link the previous Node object to the new Node object

curr\_node.idx = len(ls)

# Insert the new Node object into the array

ls = np.append(ls, new\_node)

return ls

# Test the insertElem function

print("Original linked list:")

for i in range(len(ls)):

print(ls[i].data, end="->")

print("None")

ls = insertElem(ls, 3, 'r')

print("Linked list after inserting 'r' at index 3:")

for i in range(len(ls)):

print(ls[i].data, end="->")

print("None")

Qno.2)

# Define the Node class

class Node:

def \_\_init\_\_(self, data, idx):

self.data = data

self.idx = idx

# Define the linked list as an array of Node objects

lst0 = Node(1, 1)

lst1 = Node(2, 2)

lst2 = Node(2, 3)

lst3 = Node(1, None)

lst0.idx = lst1.idx

lst1.idx = lst2.idx

lst2.idx = lst3.idx

ls = [lst0, lst1, lst2, lst3]

# Define the function to convert integer nodes to character nodes

def convertToChar(lst):

# Define the dictionary of integer to character mappings

mapping = {

1: 'A',

2: 'B',

3: 'C',

4: 'D',

5: 'E',

6: 'F',

7: 'G',

8: 'H',

9: 'I',

10: 'J',

11: 'K',

12: 'L',

13: 'M',

14: 'N',

15: 'O',

16: 'P',

17: 'Q',

18: 'R',

19: 'S',

20: 'T',

21: 'U',

22: 'V',

23: 'W',

24: 'X',

25: 'Y',

26: 'Z'

}

# Traverse the linked list and convert integer nodes to character nodes

for node in lst:

if node.data <= 26:

node.data = mapping[node.data]

else:

node.data = '#'

return lst

# Test the convertToChar function

lst0 = Node(1, 1)

lst1 = Node(2, 2)

lst2 = Node(2, 3)

lst3 = Node(1, None)

lst0.idx = lst1.idx

lst1.idx = lst2.idx

lst2.idx = lst3.idx

ls = [lst0, lst1, lst2, lst3]

print("Original linked list:")

for node in ls:

print(node.data, end="->")

print("None")

ls = convertToChar(ls)

print("Converted linked list:")

for node in ls:

print(node.data, end="->")

print("None")

Qno.3)

def partition(arr):

# Calculate the total sum of the array

total\_sum = sum(arr)

# If the total sum is odd, it cannot be partitioned into two subsets with equal sum

if total\_sum % 2 != 0:

return None

# Sort the array in descending order

arr = sorted(arr, reverse=True)

# Initialize two empty subsets

ls1 = []

ls2 = []

# Iterate over the array and add each element to the subset with the smaller sum

for num in arr:

if sum(ls1) < sum(ls2):

ls1.append(num)

else:

ls2.append(num)

# If the subsets have equal sum, return them

if sum(ls1) == sum(ls2):

return (ls1, ls2)

# Otherwise, return None

else:

return None

# Test the partition function

arr = [1, 3, 2, 1, 2, 1]

result = partition(arr)

if result is not None:

print("Subset 1:", result[0])

print("Subset 2:", result[1])

else:

print("The array cannot be partitioned into two subsets with equal sum.")

Qno.4)

def max\_value(a):

# Initialize the maximum value and the minimum values of a[u], a[v], a[w], and a[x]

max\_u = max\_v = max\_w = max\_x = float('-inf')

# Iterate over the array and update the maximum and minimum values

for i in range(len(a)):

# Update the maximum and minimum values of a[u]

max\_u = max(max\_u, a[i] + i)

# Update the maximum and minimum values of a[v]

max\_v = max(max\_v, max\_u - a[i] + i)

# Update the maximum and minimum values of a[w]

max\_w = max(max\_w, max\_v + a[i] - i)

# Update the maximum and minimum values of a[x]

max\_x = max(max\_x, max\_w - a[i] - i)

# Return the maximum value of a[u] - a[v] + a[w] - a[x]

return max\_x

# Test the max\_value function

a = [3, 9, 10, 1, 30, 40]

result = max\_value(a)

print("Maximum value:", result)

Qno.5)

from itertools import permutations

def valid\_sequence(seq, N):

for i in range(1, N+1):

first\_occurrence = seq.index(i)

second\_occurrence = first\_occurrence + i + 1

if second\_occurrence >= len(seq) or seq[second\_occurrence] != i:

return False

return True

def find\_sequences(N):

elements = [i+1 for i in range(N)] \* 2

sequences = set()

for perm in permutations(elements):

if valid\_sequence(perm, N):

sequences.add(perm)

return sequences

N = 3

sequences = find\_sequences(N)

for seq in sequences:

print("".join(map(str, seq)))