**# A Huffman Tree Node**

class node:

def \_\_init\_\_(self, freq, symbol, left=None, right=None):

self.freq = freq

self.symbol = symbol

self.left = left

self.right = right

self.huff = ' '

def printNodes(node, val=''):

newVal = val + str(node.huff)

if(node.left):

printNodes(node.left, newVal)

if(node.right):

printNodes(node.right, newVal)

if(not node.left and not node.right):

print(f"{node.symbol} -> {newVal}")

chars = ['a', 'b', 'c', 'd', 'e', 'f']

freq = [ 50, 10, 30, 5, 3, 2]

nodes = []

for x in range(len(chars)):

nodes.append(node(freq[x], chars[x]))

while len(nodes) > 1:

nodes = sorted(nodes, key=lambda x: x.freq)

left = nodes[0]

right = nodes[1]

left.huff = 0

right.huff = 1

newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

nodes.remove(left)

nodes.remove(right)

nodes.append(newNode)

printNodes(nodes[0])

OUTPUT: - a -> 0

b -> 100

d -> 1010

f -> 10110

e -> 10111

c -> 11

#Write a program non-recursive and recursive program to calculate Fibonacci numbers and a

#nalyze their time and space complexity.

#non\_recursive

a=int(input("Enter the terms"))

f=0

s=1

if a<=0:

print("series are", f)

else:

print(f,s,end=" ")

for x in range(2,a):

next=f+s

print(next,end=" ")

f=s

s=next

OUTPUT: - Enter the terms10

0 1 1 2 3 5 8 13 21 34

#Recursive program to calculate Fibonacci numbers

def fib(n):

if n<=1:

return n

else:

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

nterms=int(input("Enter the terms"))

print("fibo\_series")

for i in range(nterms):

print(fib(i))

OUTPUT: - Enter the terms10

0 1 1 2 3 5 8 13 21 34

# Assignment 3:Write a program to solve a fractional Knapsack problem using a greedy method

class ItemValue:

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

self.wt = wt

self.val = val

self.ind = ind

self.cost = val // wt

def \_\_lt\_\_(self, other):

return self.cost < other.cost

# Greedy Approach

class FractionalKnapSack:

@staticmethod

def getMaxValue(wt, val, capacity):

iVal = []

for i in range(len(wt)):

iVal.append(ItemValue(wt[i], val[i], i))

# sorting items by value

iVal.sort(reverse=True)

totalValue = 0

for i in iVal:

curWt = int(i.wt)

curVal = int(i.val)

if capacity - curWt >= 0:

capacity -= curWt

totalValue += curVal

else:

fraction = capacity / curWt

totalValue += curVal \* fraction

capacity = int(capacity - (curWt \* fraction))

break

return totalValue

# Driver Code

if \_\_name\_\_ == "\_\_main\_\_":

wt = [2,3,5,7,1,4,1]

val = [10,5,15,7,6,18,3]

capacity = 15

maxValue = FractionalKnapSack.getMaxValue(wt, val, capacity)

print("Maximum value in Knapsack =", maxValue)

OUTPUT:- Maximum value in Knapsack = 55.333333333333336

#Design n-Queens matrix having first Queen placed. Use backtracking to place remaining

#Queens to generate the final n-queen‘s matrix

global N

N = 4

def printSolution(board):

for i in range(N):

for j in range(N):

print(board[i][j], end = " ")

print()

def isSafe(board, row, col):

for i in range(col):

if board[row][i] == 1:

return False

for i, j in zip(range(row, -1, -1),

range(col, -1, -1)):

if board[i][j] == 1:

return False

for i, j in zip(range(row, N, 1),

range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solveNQUtil(board, col):

if col >= N:

return True

for i in range(N):

if isSafe(board, i, col):

board[i][col] = 1

if solveNQUtil(board, col + 1) == True:

return True

board[i][col] = 0

return False

def solveNQ():

board = [ [0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0] ]

if solveNQUtil(board, 0) == False:

print ("Solution does not exist")

return False

printSolution(board)

return True

solveNQ()

OUTPUT:-0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0

True

#Write a program for analysis of quick sort by using deterministic and randomized variant.

import random

def quicksort(arr, start , stop):

if(start < stop):

pivotindex = partitionrand(arr,\

start, stop)

quicksort(arr , start , pivotindex-1)

quicksort(arr, pivotindex + 1, stop)

def partitionrand(arr , start, stop):

randpivot = random.randrange(start, stop)

arr[start], arr[randpivot] = \

arr[randpivot], arr[start]

return partition(arr, start, stop)

def partition(arr,start,stop):

pivot = start

i = start + 1

for j in range(start + 1, stop + 1):

if arr[j] <= arr[pivot]:

arr[i] , arr[j] = arr[j] , arr[i]

i = i + 1

arr[pivot] , arr[i - 1] =\

arr[i - 1] , arr[pivot]

pivot = i - 1

return (pivot)

if \_\_name\_\_ == "\_\_main\_\_":

array = [10, 7, 8, 9, 1, 5]

quicksort(array, 0, len(array) - 1)

print(array)

OUTPUT:- [1, 5, 7, 8, 9, 10]