1.Graph colouring technique for undirected graph

Program:-

class Graph:

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

self.V = vertices

self.graph = [[0 for \_ in range(vertices)] for \_ in range(vertices)]

def is\_safe(self, v, colour, c):

for i in range(self.V):

if self.graph[v][i] == 1 and colour[i] == c:

return False

return True

def graph\_colouring\_util(self, m, colour, v):

if v == self.V:

return True

for c in range(1, m + 1):

if self.is\_safe(v, colour, c):

colour[v] = c

if self.graph\_colouring\_util(m, colour, v + 1):

return True

colour[v] = 0

def graph\_colouring(self, m):

colour = [0] \* self.V

if not self.graph\_colouring\_util(m, colour, 0):

return False

print("Solution exists. The colours assigned to vertices are:")

for c in colour:

print(c, end=" ")

return True

# Example Usage

g = Graph(4)

g.graph = [[0, 1, 1, 1],

[1, 0, 1, 0],

[1, 1, 0, 1],

[1, 0, 1, 0]]

m = 3

g.graph\_colouring(m)

2.Array of integers sorted in ascending order

Program:-

array = [2, 4, 6, 8, 10, 12, 14, 18]

print(array)

3.Robber planning to rob the houses along a street

Program:-

houses = [2, 7, 1, 5, 9, 3, 4]

def rob(houses):

if not houses:

return 0

if len(houses) == 1:

return houses[0]

dp = [0] \* len(houses)

dp[0] = houses[0]

dp[1] = max(houses[0], houses[1])

for i in range(2, len(houses)):

dp[i] = max(dp[i-1], dp[i-2] + houses[i])

return dp[-1]

max\_loot = rob(houses)

print("Maximum loot that can be robbed:", max\_loot)

4.Single source shortest path using Dijkstras algorithm

Program;-

import heapq

def dijkstra(graph, start):

distances = {node: float('infinity') for node in graph}

distances[start] = 0

queue = [(0, start)]

while queue:

current\_distance, current\_node = heapq.heappop(queue)

if current\_distance > distances[current\_node]:

continue

for neighbor, weight in graph[current\_node].items():

distance = current\_distance + weight

if distance < distances[neighbor]:

distances[neighbor] = distance

heapq.heappush(queue, (distance, neighbor))

return distances

# Example graph representation

graph = {

'A': {'B': 5, 'C': 3},

'B': {'A': 5, 'C': 2, 'D': 1},

'C': {'A': 3, 'B': 2, 'D': 4},

'D': {'B': 1, 'C': 4}

}

start\_node = 'A'

shortest\_distances = dijkstra(graph, start\_node)

print(shortest\_distances)

5.Selection sort algorithm;-

Program;-

def selection\_sort(arr):

n = len(arr)

for i in range(n):

min\_idx = i

for j in range(i+1, n):

if arr[j] < arr[min\_idx]:

min\_idx = j

arr[i], arr[min\_idx] = arr[min\_idx], arr[i]

return arr

# Example Usage

arr = [64, 25, 12, 22, 11]

sorted\_arr = selection\_sort(arr)

print("Sorted array:", sorted\_arr)

6.Sequential search:-

Program;-

def sequential\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

# Example of Sequential Search

arr = [3, 5, 2, 8, 9, 1]

target = 8

result = sequential\_search(arr, target)

if result != -1:

print(f"Target found at index: {result}")

else:

print("Target not found in the array.")

7.Binary search algorithm:-

Program;-

def sequential\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

# Example of Sequential Search

arr = [3, 5, 2, 8, 9, 1]

target = 8

result = sequential\_search(arr, target)

if result != -1:

print(f"Target found at index: {result}")

else:

print("Target not found in the array.")

8.Combination sum1:-

Program:-

def sequential\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

# Example of Sequential Search

arr = [3, 5, 2, 8, 9, 1]

target = 8

result = sequential\_search(arr, target)

if result != -1:

print(f"Target found at index: {result}")

else:

print("Target not found in the array.")

9.Merge Sort;-

Program:-

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

left = merge\_sort(left)

right = merge\_sort(right)

return merge(left, right)

def merge(left, right):

result = []

i = j = 0

while i < len(left) and j < len(right):

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

while i < len(left):

result.append(left[i])

i += 1

while j < len(right):

result.append(right[j])

j += 1

return result

10.Closest pair of points(divide and conquer)

Program:-

import math

def dist(p1, p2):

return math.sqrt((p1[0] - p2[0])\*\*2 + (p1[1] - p2[1])\*\*2)

def brute\_force(points):

min\_dist = float('inf')

for i in range(len(points)):

for j in range(i+1, len(points)):

if dist(points[i], points[j]) < min\_dist:

min\_dist = dist(points[i], points[j])

return min\_dist

def closest\_pair(points):

points.sort(key=lambda x: x[0])

return closest\_pair\_rec(points)

def closest\_pair\_rec(points):

n = len(points)

if n <= 3:

return brute\_force(points)

mid = n // 2

mid\_point = points[mid]

left\_points = points[:mid]

right\_points = points[mid:]

left\_min = closest\_pair\_rec(left\_points)

right\_min = closest\_pair\_rec(right\_points)

min\_dist = min(left\_min, right\_min)

strip = [point for point in points if abs(point[0] - mid\_point[0]) < min\_dist]

strip.sort(key=lambda x: x[1])

min\_strip = min\_strip\_dist(strip, min\_dist)

return min(min\_dist, min\_strip)

def min\_strip\_dist(strip, min\_dist):

min\_val = min\_dist

for i in range(len(strip)):

j = i + 1

while j < len(strip) and (strip[j][1] - strip[i][1]) < min\_val:

min\_val = dist(strip[i], strip[j])

j += 1

return min\_val

# Example Usage

points = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]

print(closest\_pair(points))