1. Write a program to sort a linear array using the bubble sort algorithm.

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

for j in range(0, n-i-1):

if arr[j] > arr[j+1]:

arr[j], arr[j+1] = arr[j+1], arr[j]

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

user\_input = input("Enter elements of the array separated by spaces: ")

my\_array = list(map(int, user\_input.split()))

bubble\_sort(my\_array)

print("Sorted array:", my\_array)

1. Write a program to find an element using a linear search algorithm.

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

user\_input = input("Enter a list of numbers separated by space: ")

my\_list = list(map(int, user\_input.split()))

target\_element = int(input("Enter the target element to search: "))

result = linear\_search(my\_list, target\_element)

if result != -1:

print(f"Element {target\_element} found at index {result}")

else:

print(f"Element {target\_element} not found in the list")

3. Write a program to sort a linear array using the merge sort algorithm.

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

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

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

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

input\_array = list(map(int, input("Enter space-separated integers for the array: ").split()))

print("Original Array:", input\_array)

merge\_sort(input\_array)

print("Sorted Array:", input\_array)

4.Write a program to find an element using the binary search algorithm

def binary\_search(arr, target):

low, high = 0, len(arr) - 1

while low <= high:

mid = (low + high) // 2

mid\_value = arr[mid]

if mid\_value == target:

return mid

elif mid\_value < target:

low = mid + 1

else:

high = mid - 1

return -1

sorted\_list = [int(x) for x in input("Enter a sorted list of numbers (space-separated): ").split()]

target\_element = int(input("Enter the target element to search: "))

result = binary\_search(sorted\_list, target\_element)

if result != -1:

print(f"Element {target\_element} found at index {result}")

else:

print(f"Element {target\_element} not found in the list")

5. Write a program to find a given pattern from text using the pattern matching algorithm.

import re

def find\_pattern():

text = input("Enter the text: ")

pattern\_to\_find = input("Enter the pattern to find: ")

matches = re.finditer(pattern\_to\_find, text)

for match in matches:

print(f"Pattern found at position {match.start()}-{match.end()}: {match.group()}")

find\_pattern()

6.Write a program to implement a queue data structure along with its typical operations.

class Queue:

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

self.items = []

def enqueue(self, item):

self.items.append(item)

def dequeue(self):

if not self.isEmpty():

return self.items.pop(0)

else:

print("Queue is empty. Cannot dequeue.")

def isEmpty(self):

return len(self.items) == 0

def size(self):

return len(self.items)

my\_queue = Queue()

while True:

user\_input = input("Enter an item to enqueue (press 'q' to stop): ")

if user\_input.lower() == 'q':

break

try:

item = int(user\_input)

my\_queue.enqueue(item)

except ValueError:

print("Invalid input. Please enter an integer.")

print("Size of the queue:", my\_queue.size())

while not my\_queue.isEmpty():

print("Dequeued item:", my\_queue.dequeue())

print("Size of the queue after dequeue:", my\_queue.size())

7. Write a program to solve n queen's problem using backtracking.

def is\_safe(board, row, col, N):

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 solve\_n\_queens\_util(board, col, N, solutions):

if col == N:

solutions.append([row[:] for row in board])

return

for i in range(N):

if is\_safe(board, i, col, N):

board[i][col] = 1

solve\_n\_queens\_util(board, col + 1, N, solutions)

board[i][col] = 0

def print\_all\_solutions(N):

board = [[0] \* N for \_ in range(N)]

solutions = []

solve\_n\_queens\_util(board, 0, N, solutions)

if not solutions:

print("No solution exists")

else:

print("Total number of solutions:", len(solutions))

print("One of the solutions:")

for row in solutions[0]:

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

def solve\_n\_queens():

N = int(input("Enter the size of the chessboard (N): "))

print\_all\_solutions(N)

solve\_n\_queens()

8. Consider a set S = {5, 10,12, 13, 15, 18) and d = 30. Write a program to solve the sum of subset problem.

def isSubsetSum(S, n, d):

dp = [[False for \_ in range(d + 1)] for \_ in range(n + 1)]

for i in range(n + 1):

dp[i][0] = True

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

for j in range(1, d + 1):

if S[i - 1] > j:

dp[i][j] = dp[i - 1][j]

else:

dp[i][j] = dp[i - 1][j] or dp[i - 1][j - S[i - 1]]

return dp[n][d]

S = list(map(int, input("Enter the array S (space-separated): ").split()))

d = int(input("Enter the target sum d: "))

n = len(S)

result = isSubsetSum(S, n, d)

if result:

print("Subset with the sum", d, "exists.")

else:

print("No subset with the sum", d, "exists.")

9. Write a program to solve the following 0/1 Knapsack using dynamic programming approach profits P = (15,25,13,23), weight W = (2,6,12,9), Knapsack C = 20, and the number of items n=4.

def knapsack\_01(P, W, C, n):

dp = [[0 for \_ in range(C + 1)] for \_ in range(n + 1)]

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

for w in range(C + 1):

if W[i - 1] <= w:

dp[i][w] = max(dp[i - 1][w], P[i - 1] + dp[i - 1][w - W[i - 1]])

else:

dp[i][w] = dp[i - 1][w]

max\_profit = dp[n][C]

return max\_profit

n = int(input("Enter the number of items: "))

P = [int(input(f"Enter profit for item {i + 1}: ")) for i in range(n)]

W = [int(input(f"Enter weight for item {i + 1}: ")) for i in range(n)]

C = int(input("Enter the capacity of the knapsack: "))

result = knapsack\_01(P, W, C, n)

print(f"Maximum profit: {result}")

10. Write a program to solve the Tower of Hanoi problem for the N disk.

def tower\_of\_hanoi(n, source, target, auxiliary):

if n > 0:

tower\_of\_hanoi(n-1, source, auxiliary, target)

print(f"Move disk {n} from {source} to {target}")

tower\_of\_hanoi(n-1, auxiliary, target, source)

n = int(input("Enter the number of disks: "))

tower\_of\_hanoi(n, 'A', 'C', 'B')