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)
```

2. 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')
```