Problem NO : 01

Problem Title : Write a program to sort a linear Array using bubble sort Algorithm.

Problem Description :- Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. We have to remember that bubble sort is not the most efficient sorting algorithm for large datasets, but it helps the illustrate the basi principle of the sorting algorithm.

Algorithm for Bubble sort :-

Step 1 : Begin with the first element of the array.

Step 2 : Compare the current element with the next element in the array.

Step 3 : If the current element is greater than the next element, swap them.

Step 4 : Move to the next pair of the elements in the array and repeat steps 2 and 3.

Step 5: Continue steps 2 to 4 until you reach the end of the array. At this point the largest element will be at the last position.

Step 6: Repeated steps 1 to 5 for the remaining unsorted elements in the array. After the first pass, the largest element in its final position.

Step 7:- Repeat steps 1-6 until the entire array is sorted.

Source Code in python :-

```
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]

input_array = input("Enter space-separated integers:")
unsorted_array = list(map(int, input_array.split()))

bubble_sort(unsorted_array)
print("Sorted array:", unsorted_array)
```

Sample input :-

Enter space-separated integers : 6 4 7 2 8

Sample output :-

Sorted array : [2, 4, 6, 7, 8]

Problem NO :- 02

Problem Title:- Write a program to find an element using linear search algorithm.

Problem Description:- The linear search algorithm is a simple method used to find the position of a specific element within a collection, such as an array. The problem involves searching through each element of the collection one by one until the desired is found or the entire collection has been traversed.

Algorithm for Linear Search:-

Step 1 : Begin with the first element of the array.

Step 2 : Compare the current element with the target element.

Step 3 : If they are equal, return the index of the current element.

Step 4 : If not, move to the next element in the array.

Step 5: Repeat steps 2 to 4 until the target element is found or the end of the array is reached.

Step 6: If the target element is not found, return a message indicating it's absence.

Source Code in python :-

```
def linear_search (arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  // return the index if the target
                      element is found.
        return -1  // if the target element is not found
// input from the terminal
array_input = input (" Enter the Array Elements separated
                    by space:")
my_array = list (map(int, array_input.split()))
target_element = int (input (" Enter the target element to
                    search:"))
result = linear_search (my_array, target_element)
```

```
if result != -1:
    print (f"Element {target_element} found at index
    {result}.")
else:
    print (f"Element {target_element} not found in the array.")
```

Sample Input:

Enter the Array Elements separate by the space:

5 6 7 3 8

Enter the target Element to search: 3

Sample output:

Element 3 found at index 3.

Problem NO : 03

Problem Title : Write a program to sort a linear array using the merge sort algorithm.

Problem Description : The Merge sort Algorithm is a popular and efficient comparison-based sorting algorithm that follows the divide-and-conquer paradism. The problem involves sorting an array of elements in non-decreasing order using the merge sort algorithm in corresponding.

input: An unsorted ele: array of elements.

Output: The same array with elements rearranged in non-decreasing order.

Algorithm for the Merge Sort:

Step 1 : If the length of the array is 0 or 1, it is considered sorted, so return.

Step 2 : Find the middle index of the array.

Step 3: Recursively apply merge sort to the left half of the array.

Step 4: Recursively apply merge sort to the right half of the array.

Step 5: Combine all the sorted lists into a single sorted list.

Step 6: The list is now sorted.

Source Code in python :-

```
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
def main():
    try:
        input_list = list(map(int, input("Enter space-separated numbers to be sorted: ").split()))
    except ValueError:
        print("Invalid input. Please Enter space-separated numbers.")
        return
```

```
    merge_sort (input_list)
    print ("Sorted list:", input_list)
if __name__ == "__main__":
    main()
```

Sample Input:-

Enter space-separated numbers to be sorted:
4 3 8 2 7

output:

Sorted list : [2, 3, 4, 7, 8]

Problem NO : 04

Problem Title: Write a program to find an element using the binary search algorithm.

Problem Description :- Binary search is a efficient algorithm used to locate a specific element in a sorted array or list. The process involves repeatedly dividing the search space in half until the target element is found or it is determined that the element is not present. It's a logarithmic time complexity algorithm, making it faster than linear search or any other search for large datasets.

Algorithm for Binary Search :-

Step 1 : Set the left pointer (low) to the start of the array, the right pointer (high) to the end of the array, and the middle index (mid) to the floor of the average of 'low' and 'high'.

Step 2 : If the 'low' index is greater than the 'high'

index, the element is not in the array. Return an appropiate indication like '-1' or a message.

Step 3: Compare the element at index 'mid' with the target element.

Step 4: If they are equal, you found the element. Return the index 'mid'.

Step 5: If the target element is less than the element at index 'mid' set 'high' to 'mid - 1'. Otherwise, set 'low' to 'mid + 1'

Step 6: Recalculate the middle index: 'mid = floor((low + high) / 2)'

Step 7: Go back to step 2.

Source Code in python :-

```
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  # target found, return the index
    elif mid_value < target:
        low = mid + 1
    else:
        high = mid - 1
    return -1  # target not found.

def main():
    try:
        input_list = list(map(int, input("Enter a sorted list of space-separated numbers: ").split()))
        target = int(input("Enter the number to search for: "))
    except ValueError:
        print("Invalid input. Please Enter a sorted list of space-separated numbers and the target number.")
        return
    result = binary_search(input_list, target)
    if result != -1:
        print(f"Target {target} found at index {result}.")
```

```
    else:
        print(f"Target {target} not found in the list.")
if __name__ == "__main__":
    main()
```

Sample Input:

Enter a sorted list of space-separated numbers: 5 37 8 4 6

Enter the number to search for: 4

Sample Output:

Target 4 not found in the list.

Problem NO-05

Problem Title:- Write a program to find a given pattern from text using the pattern matching algorithm.

Problem Description:- The problem of a pattern matching involves finding occurances of a specified pattern within a given text. The goal is to identify all positions where the pattern appears in the text. The task is fundamental in various applications such as text searching, data mining and bio-informatics.
A common approaches to solving this problem is by using pattern matching algorithm like the knuth-Morris-Prat (KMP) algorithm.

Algorithm for Pattern Matching:-

step 1:- set k := 1 and max := S-R+1

step 2:- Repeat steps 3 to 5 while k ≤ MAX

step 3:- Repeat for L=1 to R

if P[L] ≠ T [K+L-1]. then go to step 5.

Step 4:- Set INDEX = K and exit.

Step 5:- Set K := K+1

Step 6:- Set INDEX = 0

Source Code in Python :-

```
def Compute_Prefix_function (Pattern):
    m= len ( pattern)
    prefix_function = [0] * m
    k=0
    for q in the range (1, m):
        while k>0 and pattern [k] != pattern [q] :
            k = prefix_function [k-1]
        if pattern [k] == pattern [q] :
            k+=1
        Patter_function [q] = k
    return prefix_function
def pattern_matching (text, pattern):
    n = len (text)
    m = len ( pattern)
    pattern_function = Compute_prefix_function (pattern)
    positions = []
```

```
    q=0
    for i in range(n):
        while q>0 and pattern[q]!= text[i]:
            q= prefix_function[q-1]
        if pattern[q] == text[i]:
            q+=1
        if q==m:
            positions.append(i-m+1)
            q= prefix_function[q-1]
    return positions
if __name__=="__main__":
    text= input("Enter the text:")
    Pattern= input("Enter the pattern:")

    result = pattern_matching(text, pattern)

if result:
        print("Pattern found at positions:", result)
else:
        print("Pattern not found in the text.")
```

Sample Input:

Enter the text: my name is manob chanda

Enter the pattern: is manob

Sample Output:

Patten found at positions : [8]

Problem No : 06

Problem Title :- Write a program to implement a queue Data structure along with it's typical operations.

Problem Description :- A queue data structure is a linear Collection of elements that follows the first in first out (FIFO) principle, meaning that the element that is inserted first will be the first one to be removed.

Here is a pictorial representation of a queue along with ~~its~~ typical operations :-

① Queue Visualization :-

Front [1, 2, 3, 4, 5,] Rear.

② Typical operations :-

- Enqueue (Insertion):

  Enqueue (6)

  Front [1, 2, 3, 4, 5, 6] Rear.

- Dequeue ( )

  Front [2, 3, 4, 5, 6] Rear.

- Peek (Front Element):

  Peek() :->2

  Front [2, 3, 4, 5, 6] Rear.

- IsEmpty:

  IsEmpty ()-> False.

- Size : Returns the number of elements in the queue.

Algorithm for Queue :-

1. Initialize: Create an empty list to hold the elements of the queue.
2. Enqueue:- To add an element to the queue, Simply appen it to the end of the list.
3. Dequeue: For remove, pop the first element from the list.
4. Peek : To get the element at the front of the queue without removing it,
5. Size: Return the length of list, which represent

the number of elements of the queue.

5. Is Empty: Return 'True' if the length of the list is 0.

Source Code in python :-

```
class Queue:
    def __init__(self):
        self.s1=[]
        self.s2= []
    def push (self, x) :
        self .s1. append (x)
    def pop (self) :
        if not self.s1 and not self.s2 :
            print ("Queue is empty")
            return -1
        if not self.s2 :
            while self.s1:
                self.s2.append (self.s1. pop())
        topval = self. s2. pop()
        return topval
    def empty (self):
        return not self.s1 and not self.s2
```

```
if __name__ = "__main__":
    q = Queue()
    q.push(1)
    q.push(2)
    q.push(3)
    print(q.pop())
    print(q.pop())
    print(q.pop())
    print(q.pop())
```

Output:

```
1
2
3
Queue is empty
-1.
```

Problem NO: 07

Problem Title: Write a program to solve n queen's problem using backtracking.

Problem Description:- The N queen's problem is a classic problem in Computer science and combinatorial optimization. The objective is to place N chess queens on an N×N cheesboard in such a way that no two queens threaten each other. In other words, no two queens should share the same row, column, or diagonal. The challenge is to find all possible solutions to this problem.

Algorithm for N queen's Problem:-

Step 1: Start in the left most column.

Step 2:- If all queens are placed, returned true.

Step 3:- Try all rows in the current column. For each row, do the following:

a. If the queen can be placed safely in this

row and column, mark this cell and recursively check if placing the queen here leads to a solution.

(b) If placing the queen in this cell leads to a solution, return true.

(c) If placing the queen in this cell does not lead to a solution, unmark this cell and other rows.

Step 4:- If no row worked, return false.

Source Code in Python :-

```
def is_safe (board, row, col, n):
    # Check if there is a queen in the same row.
    for i in range (col):
        if board [row] [i] == 1 :
            return False,

    # Checkert upper diagonal on left side
    for i, j in zip (range (row, -1, -1), range (col, -1, -1)):
        if board [i] [j] == 1 :
            return False

    # Check lowert diagonal on left side
    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_until (board, col, n):
    if col >= n:
        return True
    for i in range (n):
        if is_safe (board, i, col, n):
            board [i] [col] = 1
            if solve_n_queens_until (board, col+1, n):
                return True
            board [i] [col] = 0
    return False
def solve_n_queen (n):
    board = [[0 for _ in range (n)] for _ in range(n)]
    if not solve_n_queens_until (board, 0, n):
        print ("Solution doesn't exist")
        return False.
    print ("Solution :")
    for row in board:
        print (row)
    return True
```

Example Usage:

n=4

Solve_n_queens(n)

Solution Output:-

Solution :

[0, 0, 1, 0]

[1, 0, 0, 0]

[0, 0, 0, 1]

[0, 1, 0, 0]

Phoblem No-8

Problem Title:- Consider a set S={5,10,12,13,15,18} and d=30. Write a program to solve the sum of the subset problem.

Problem Description:- The subset sum problem is a classic problem in computer science and combinational optimization. The goal is to determine whether there exists a subset of a given set of positive integers whose elements add up to a specified target sum.
In the specific python program provided, the problem is approached using dynamic programming. The function 'is_subset_sum' takes a set of positive integers.

Algorithm for Sum of the subset:-

Step 1:- Create 2D Array 'dp' of size '(n+1)' x (target+1). Where n is the size of the given set.

Step 2:- Set the first column of the 'dp' to 'true'.

Step 3:- Iterate over the set elements and the target sum. For each element and sum, Update 'dp[i][j]

Step 4 :- Use the following logic to update dp[i][j]

(ii) If the current element is less or equal to the current sum, Consider the choice of including or excluding the element.

(iii) If greater exclude the sum.

Step 5 :- After filling the table the final cell dp[n][target]

Step 6 :- print the value in dp[n][target].

Source Code in Python :-

```
dp is _subset_sum(set, n, target):
    dp = [[ False for _ in range (target+1)] for _ in rang (n+1)]
    for i in range (n+1):
        dp[i][0] = True
    for i in the range (1, n+1):
        for j in Range (1, target+1):
            if set[i-1] <= j :
                dp[i][j] = dp[i-1][j] or dp[i-1][j-set[i-1]]
            else :
                dp[i][j] = dp[i-1][j]
    return dp[n][target]
```

```
input_set = list(map(int, input("Enter the sets (comma separated): ").split(',')))
target_sum = int(input("Enter the target sum (d):"))

if is_subset_sum(input_set, len(input_set), target_sum):
    print("Subset with sum", target_sum, "exist in the given set:")
else:
    print("No subset with sum", target_sum, "found in the given set:")
```

Input:

Enter the sets (comma separated): 5,10,12,13,15,18

Enter the target sum (d): 30

Output:

Subset with sum 30 exists in the given set.

Problem NO : 09

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

Problem Description :- The 0/1 knapsack is a classic optimization problem in computer science and mathematics. Given a set of items, each with a specific weight and profit, and a knapsack with a fixed capacity, the objective is to determine the maximum profit that can be obtain from the selecting a subset of items to include in the knapsac

The problem is considered NP-hard, meaning that there is no known polynomial-time algorithm to solve it optimally in the general case.

Algorithm of knapsack Problem :-

Step 1 :- Input,

- profits array 'P'
- weights array 'W'
- knapsack capacity 'C'
- number of items 'n'.

Step 2:- Initialize a 2D array 'dp' with dimensions (n+1) x (c+1)' for subproblem solutions.

Step 3:- Base Case,

- Set 'dp[i][0] = 0 'for all 'i'.
- Set dp[0][w] = 0 for all 'w'.

Step 4:- Fill DP table,

- If 'w[i-1] <= w: set dp[i][w]
- else set, dp[i][w] to dp[i-1][w].

Step 5:- Return dp[n][c] as the maximum profit.

Source Code in Python:-

```
def knapsack_01(p,w,c,n):
    dp = [[0]* (c+1) for _ in range(n+1)]
        for i in range (n+1):
            for w in range (c+1):
                if i==0 or w==0:
                    dp[i][w]=0
            elif w[i-1] <=w:
                dp[i][w] = max(p[i-1]+dp[i-1][w-w[i-1]],
                                        dp[i-1][w])
            else:
                dp[i][w] = dp[i-1][w]
```

```
    else:
        dp[i][w] = dp[i-1][w]
select_items = []
i, j = n, c
while i>0 and j>0:
    if dp[i][j] != dp[i-1][j]:
        selected_items.append(i-1)
        j = w[i-1]
    i -= 1
return dp[n][c], selected_items[::-1]
profit = [15, 25, 13, 23]
weights = [2, 6, 12, 9]
knapsack_capacity = 20
num_items = 4
max_profit, selected_items = knapsack_01(profits, weights
knapsack capacity, num_items)
print("Maximum Profit:", max_profit)
print("Maximum Profit:" max selected items)
```

Input :-

profits = [15, 25, 13, 23]

Weights = [2, 6, 12, 9]

knapsack_capacity = 20

num_items = 4

Output :-

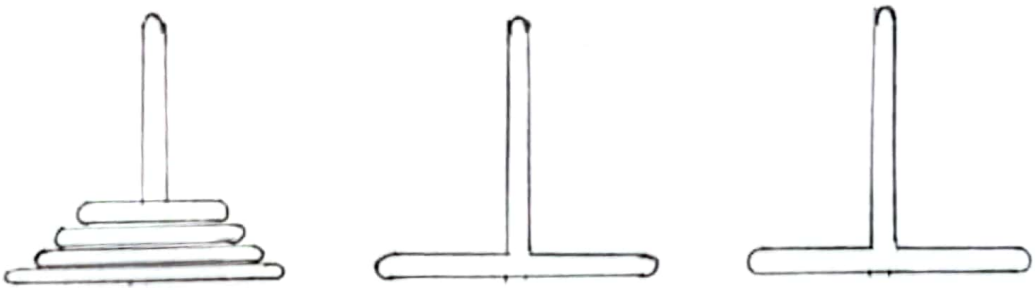
Maximum Profit : 63

selected items : [0, 1, 3]

Problem NO: 10

Problem Title :- Write a program to solve the Towerr of Hanoi problem forr N Desk.

Problem Description :- The Towerr of Hanoi is a classical problem in Computerr science and Mathematics. It involves three pegs and a numberr of disks of differrent sizes, which can slide onto any peg. The puzzle starrts with the disks in a neat stack in ascending orrderr of size on one peg, the smallest at the top.



Algorrithm forr Towerr of Hanoi :-

Step 1 :- Identify Base case :- If the numberr of disks is 1, simp move it frrom the sourrce peg to the destination peg.

2. Move n-1 Disks :- Recurrsively, move the top n-1 disks frrom the sourrce peg to the aunillarry peg, using the destination peg ar a temporrarry peg.

3. Move the Largest Disk :- Move the largest peg from the source peg to the destination peg.

4. Recursive Call :- Recursively move the n-1 disks from the auxillary peg to the destination peg, using the source peg as a temporary peg.

5. Repeat Untill all disk moved :- Repeat steps 2-4 untill all disks are moved from the source peg to the destination peg.

6. Implement Base Case :- Implement the base case and recursive steps in code to solve the tower of Hanoi problem from any given number of Disks.

Source Code in Python :-

```
def tower_of_hanoi(n, source, destination, auxiliary):
    if n==1:
        print(f"Move disk 1 from {source} to {destination}")
        return
    tower_of_hanoi(n-1, source, auxiliary, destination)
    print(f"Move disk {n} from {source} to {destination}")
```

```
    towers_of_hanoi(n-1, auxiliary, destination, source)

def main():
    n = int(input("Enter the numbers of disks: "))
    towers_of_hanoi(n, 'A', 'C', 'B')
if __name__=="__main__":
    main()
```

Input:

Enter the numbers of disks : 3

Output:

Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C