Practical – 1

A - Program to find sum of elements in an array

```
# Take user input to create array
arr = []
n = int(input("Enter the number of elements in the array: "))
print("Enter the elements of the array:")
for i in range(n):
    arr.append(int(input()))

# Calculate the sum of all elements in the array
sum = 0
for i in range(n):
    sum += arr[i]

# Print the sum of all elements in the array
print("The sum of all elements in the array is:", sum)
```

B - Program to find minimum or maximum element in an array

```
import time

# Take user input to create array
arr = []
```

```
n = int(input("Enter the number of elements in the array: "))
print("Enter the elements of the array:")
for i in range(n):
    arr.append(int(input()))

# Find the minimum and maximum elements in the array
start_time = time.time()
min_element = arr[0]
max_element = arr[0]
for i in range(1, n):
    if arr[i] < min_element:
        min_element = arr[i]
    if arr[i] > max_element:
        max_element = arr[i]
end_time = time.time()

# Print the minimum and maximum elements in the array and time complexity
print("The minimum element in the array is:", min_element)
print("The maximum element in the array is:", max_element)
print("Time complexity:", end_time - start_time, "seconds")
```

C - program to count number of even and odd elements in an array

```
# Take user input to create array
arr = []
n = int(input("Enter the number of elements in the array: "))
print("Enter the elements of the array:")
for i in range(n):
    arr.append(int(input()))

# Count the number of even and odd elements in the array
count_even = 0
count_odd = 0
for i in range(n):
    if arr[i] % 2 == 0:
        count_even += 1
    else:
        count_odd += 1

# Print the number of even and odd elements in the array
print("The number of even elements in the array is:", count_even)
print("The number of odd elements in the array is:", count_odd)
```

Practical - 2

Sum of row element, column element and diagonal element.

```
# Take user input to create square matrix
n = int(input("Enter the size of the square matrix: "))
print("Enter the elements of the matrix:")
matrix = []
for i in range(n):
    row = list(map(int, input().split()))
    matrix.append(row)

# Calculate the sum of row elements, column elements, and diagonal elements
sum_row = [sum(row) for row in matrix]
sum_column = [sum(column) for column in zip(*matrix)]
sum_diagonal1 = sum(matrix[i][i] for i in range(n))
sum_diagonal2 = sum(matrix[i][n-i-1] for i in range(n))

# Print the sum of row elements, column elements, and diagonal elements
print("Sum of row elements:", sum_row)
print("Sum of column elements:", sum_column)
print("Sum of diagonal elements:", [sum_diagonal1, sum_diagonal2])
```

B- Sum of two matrices.

```
# Take user input to create matrix A
n = int(input("Enter the number of rows of matrix A: "))
m = int(input("Enter the number of columns of matrix A: "))
print("Enter the elements of matrix A:")
A = []
for i in range(n):
    row = list(map(int, input().split()))
    A.append(row)

# Take user input to create matrix B
print("Enter the elements of matrix B:")
B = []
for i in range(n):
    row = list(map(int, input().split()))
    B.append(row)

# Add matrix A and matrix B to create matrix C
C = []
for i in range(n):
    row = []
    for j in range(m):
        row.append(A[i][j] + B[i][j])
    C.append(row)

# Print matrix C, the sum of matrix A and matrix B
print("The sum of matrix A and matrix B is:")
for row in C:
    print(*row)
```

C - Multiplication of two matrices.

```
# Take user input to create matrix A
n = int(input("Enter the number of rows of matrix A: "))
m = int(input("Enter the number of columns of matrix A: "))
print("Enter the elements of matrix A:")
A = []
for i in range(n):
    row = list(map(int, input().split()))
    A.append(row)

# Take user input to create matrix B
p = int(input("Enter the number of columns of matrix B: "))
print("Enter the elements of matrix B:")
B = []
for i in range(m):
    row = list(map(int, input().split()))
    B.append(row)

# Multiply matrix A and matrix B to create matrix C
C = []
for i in range(n):
    row = []
    for j in range(p):
        element = 0
        for k in range(m):
            element += A[i][k] * B[k][j]
        row.append(element)
    C.append(row)
```

```
# Print matrix C, the product of matrix A and matrix B
print("The product of matrix A and matrix B is:")
for row in C:
    print(*row)
```

Practical - 3

a. Program to create a list-based stack and perform various stack operations

```
stack = []  # create an empty list to use as a stack

# push items onto the stack
stack.append(1)
stack.append(2)
stack.append(3)

# print the stack
print("Stack:", stack)

# pop an item from the stack
item = stack.pop()
print("Popped item:", item)

# print the stack again
print("Stack:", stack)

# peek at the top item of the stack
top_item = stack[-1]
```

```
print("Top item:", top_item)


# check if the stack is empty
if not stack:
    print("Stack is empty")
else:
    print("Stack is not empty")


# get the size of the stack
size = len(stack)
print("Stack size:", size)


# clear the stack
stack.clear()
print("Cleared stack:", stack)
```

b. Program to create infix to postfix expression conversion using stack.

```
# define a function to convert infix to postfix
def infix_to_postfix(expression):
    # initialize an empty stack and an empty output string
    stack = []
    output = ""

    # define a dictionary to store operator precedence
    precedence = {"+": 1, "-": 1, "*": 2, "/": 2, "^": 3}
```

```
    # loop through each character in the expression
    for char in expression:
        # if the character is an operand, add it to the output string
        if char.isalnum():
            output += char
        # if the character is an operator
        elif char in precedence:
            # pop operators off the stack and add them to the output string
            # while they have higher or equal precedence
            while stack and stack[-1] != "(" and precedence[char] <= precedence.get(stack[-1],
0):
                output += stack.pop()
            # push the current operator onto the stack
            stack.append(char)
        # if the character is a left parenthesis, push it onto the stack
        elif char == "(":
            stack.append(char)
        # if the character is a right parenthesis, pop operators off the stack
        # and add them to the output string until a left parenthesis is found
        elif char == ")":
            while stack and stack[-1] != "(":
                output += stack.pop()
            # remove the left parenthesis from the stack
            if stack and stack[-1] == "(":
                stack.pop()

    # pop any remaining operators off the stack and add them to the output string
    while stack:
        output += stack.pop()
```

```
    return output


# example usage
expression = "a + b * c - d / e ^ f"
postfix_expression = infix_to_postfix(expression)
print("Infix expression:", expression)
print("Postfix expression:", postfix_expression)
```

Practical -4

a) Linear search

```
def linear_search(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1


# example usage
arr = [3, 5, 2, 8, 4, 9]
x = 8
index = linear_search(arr, x)
if index != -1:
    print(f"{x} found at index {index}")
else:
    print(f"{x} not found")
```

b) Binary search(Iterative method)

```
def binary_search(arr, x):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            left = mid + 1
        else:
            right = mid - 1
    return -1

# example usage
arr = [1, 3, 5, 7, 9, 11, 13]
x = 7
index = binary_search(arr, x)
if index != -1:
    print(f"{x} found at index {index}")
else:
    print(f"{x} not found")
```

c) Binary search(Recursive method)

```
def binary_search_recursive(arr, x, left, right):
    """
    Searches for the value x in the given sorted array using binary search (recursive method).
    Returns the index of x if found, or -1 if not found.
    """
    if left > right:
        return -1
    mid = (left + right) // 2
    if arr[mid] == x:
        return mid
    elif arr[mid] < x:
        return binary_search_recursive(arr, x, mid + 1, right)
    else:
        return binary_search_recursive(arr, x, left, mid - 1)

def binary_search(arr, x):
    return binary_search_recursive(arr, x, 0, len(arr) - 1)

# example usage
arr = [1, 3, 5, 7, 9, 11, 13]
x = 7
index = binary_search(arr, x)
if index != -1:
    print(f"{x} found at index {index}")
else:
    print(f"{x} not found")
```

Practical - 5

A – Bubble sort

```
def bubble_sort(arr):
    n = len(arr)
    # iterate over all elements in the array
    for i in range(n):
        # flag to keep track of whether a swap was made in this iteration
        swapped = False
        # iterate over unsorted part of the array
        for j in range(n - i - 1):
            # swap adjacent elements if they are in the wrong order
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # if no swaps were made in this iteration, the array is already sorted
        if not swapped:
            break

# example usage
arr = [5, 2, 8, 1, 3, 9, 4, 6, 7]
bubble_sort(arr)
print(arr)

#bubble sort has a worst-case time complexity of O(n^2)
```

B-Selection sort

```
def selection_sort(arr):
    n = len(arr)
    # iterate over all elements in the array
    for i in range(n):
        # find the minimum element in the unsorted part of the array
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        # swap the minimum element with the first unsorted element
        arr[i], arr[min_index] = arr[min_index], arr[i]

# example usage
arr = [5, 2, 8, 1, 3, 9, 4, 6, 7]
selection_sort(arr)
print(arr)

#selection sort has a worst-case time complexity of O(n^2)
```

Practical-6

Programs to select the Nth Max/Min element in a list by using various algorithms.

```
#Programs to seLect the Nth Max/Min element in a List by using various aLgorithms
import time
st=time.time()
el=([])
n=int(input("Enter length of the list:"))
for i in range(n):
   x=int(input("Enter Numbers : "))
   el.append(x)
print("el =",el)
print("Maximum Element : ",max(el))
print("Minimum Element : ",min(el))
ed=time.time()
final=ed-st
print("Time taken to execute code : ",final)
```

Practical -7

Programs to find a pattern in a given string - general way and brute force technique

```
n=int(input("Enter number of cities :"))
city=()
for i in range(n):
    c=input("Enter City :")
    city+=(c,)
print(city)
pat=input("Enter Pattern you want to search for?")
for c in city:
    if(c.find(pat)!=-1):
        print(c)
```

Practical 8

Programs on recursion like factorial, fibonacci, tower of hanoi. Compare algorithms to find factorial/fibonacci using iterative and recursive approaches.

A:- Factorial

```
def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n*recur_factorial(n-1)
#take input from the user
num = int(input("Enter a number: "))
#check is the number is negative
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num==0:
    print ("The factorial of 0 is 1")
else:
    print("The factorial of",num,"is",recur_factorial(num))
```

B -Fibonacci

```
def recur_fibo(n):
   if n <= 1:
      return n
   else:
      return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = 10
# check if the number of terms is vaLid
if nterms <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
      print(recur_fibo(i))
```

C - Tower of hanoi.

```
# tower of hanoi
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
   if n == 1:
      print ("Move disk 1 from rod",from_rod,"to rod",to_rod)
      return
   TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
   print("Move disk",n,"from rod",from_rod,"to rod",to_rod)
   TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)
# main
n = 3
TowerOfHanoi(n, 'A','C','B')
```

Practical 9 strassen's algorithm

```
import numpy as np
def strassen_algorithm(x, y):
    if x.size == 1 or y.size == 1:
        return x * y
    n = x.shape[0]
    if n % 2 == 1:
        x = np.pad(x, (0, 1), mode="constant")
        y = np.pad(y, (0, 1), mode="constant")



    m = int(np.ceil(n / 2))
    a = x[: m, : m]
    b = x[: m, m:]
    c = x[m:, : m]
    d = x[m:, m:]
    e = y[: m, : m]
    f = y[: m, m:]
    g = y[m:, : m]
    h = y[m:, m:]
    p1 = strassen_algorithm(a, f - h)
    p2 = strassen_algorithm(a + b, h)
    p3 = strassen_algorithm(c + d, e)
    p4 = strassen_algorithm(d, g - e)
    p5 = strassen_algorithm(a + d, e + h)
    p6 = strassen_algorithm(b - d, g + h)
    p7 = strassen_algorithm(a - c, e + f)
    result = np.zeros((2 * m, 2 * m), dtype=np.int32)
    result[: m, : m] = p5 + p4 - p2 + p6
```

```
    result[: m, m:] = p1 + p2
    result[m:, : m] = p3 + p4
    result[m:, m:] = p1 + p5 - p3 - p7
    return result[: n, : n]
if __name__ == "__main__":
    x = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
    y = np.array([[-1, 0, 0], [0, -1, 0], [0, 0, -1]])
    print("Matrix multiplication result: ")
    print(strassen_algorithm(x, y))
```