**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))