**Practical no 1**

#Accept And Display matrix

a = int(input("Enter number of row: "))

b=int(input("Enter number of column: "))

# empty list for matrix

m=[]

# taking matrix input

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

l=[]

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

n=int(input(f"Enter a[{i}][{j}] element: "))

l.append(n)

m.append(l)

#printing in matrix for

for i in range(a):

for j in range(b):

if(j==b-1):

print(m[i][j], end=" ")

else:

print(m[i][j], end=" ")

if i!=a-1:

print()

OUTPUT:

======== RESTART: D:\Python Practicals\Prac1.0 Accept And Display.py ========

Enter number of row: 2

Enter number of column: 2

Enter a[1][1] element: 4

Enter a[1][2] element: 6

Enter a[2][1] element: 8

Enter a[2][2] element: 9

4 6

8 9

//Addition of matrix

import numpy as np

#First Matrix

print("FIRST MATRIX")

matrix1 = []

rows = int(input("Number Of Rows: "))

cols = int(input("Number Of Columns: "))

for r in range(rows):

row = []

matrix1.append(row)

for c in range(cols):

row.append(int(input("Matrix 1-> R: {} C: {}\n".format(r+1, c+1))))

for i in matrix1:

print(i)

print("\n")

#Second Matrix

print("SECOND MATRIX")

matrix2 = []

rows2 = int(input("Number Of Rows: "))

cols2 = int(input("Number Of Columns: "))

for r in range(rows2):

lis = []

matrix2.append(lis)

for c in range(cols2):

lis.append(int(input("Matrix 1-> R: {} C: {}\n".format(r+1, c+1))))

for j in matrix2:

print(j)

print("\n")

addition=[[0 for x in range(cols2) ] for y in range(rows)]

for i in range(0,rows):

for j in range(0,rows2):

for k in range(0,cols):

addition[i][j]=matrix1[i][j]+matrix2[i][j]

m1=np.matrix(matrix1)

m2=np.matrix(matrix2)

m3=np.matrix(addition)

print(m1)

print("\n")

print(m2)

print("\n")

print(m3)

OUTPUT:

============= RESTART: D:\Python Practicals\Prac1.1 Addition.py =============

FIRST MATRIX

Number Of Rows: 2

Number Of Columns: 2

Matrix 1-> R: 1 C: 1

1

Matrix 1-> R: 1 C: 2

4

Matrix 1-> R: 2 C: 1

7

Matrix 1-> R: 2 C: 2

5

[1, 4]

[7, 5]

SECOND MATRIX

Number Of Rows: 2

Number Of Columns: 2

Matrix 1-> R: 1 C: 1

7

Matrix 1-> R: 1 C: 2

9

Matrix 1-> R: 2 C: 1

4

Matrix 1-> R: 2 C: 2

2

[7, 9]

[4, 2]

[[1 4]

[7 5]]

[[7 9]

[4 2]]

[[ 8 13]

[11 7]]

//subtraction of matrix

import numpy as np

#First matrix

print("FIRST MATRIX")

matrix1 = []

rows = int(input("Number Of Rows: "))

cols = int(input("Number Of Columns: "))

for r in range(rows):

row = []

matrix1.append(row)

for c in range(cols):

row.append(int(input("Matrix 1-> R: {} C: {}\n".format(r+1, c+1))))

for i in matrix1:

print(i)

print("\n")

#Second Matrix

print("SECOND MATRIX")

matrix2 = []

rows2 = int(input("Number Of Rows: "))

cols2 = int(input("Number Of Columns: "))

for r in range(rows2):

lis = []

matrix2.append(lis)

for c in range(cols2):

lis.append(int(input("Matrix 1-> R: {} C: {}\n".format(r+1, c+1))))

for j in matrix2:

print(j)

print("\n")

sub=[[0 for x in range(cols2) ] for y in range(rows)]

for i in range(0,rows):

for j in range(0,rows2):

for k in range(0,cols):

sub[i][j]=matrix1[i][j]-matrix2[i][j]

m1=np.matrix(matrix1)

m2=np.matrix(matrix2)

m3=np.matrix(sub)

print(m1)

print("\n")

print(m2)

print("\n")

print(m3)

output:

============ RESTART: D:\Python Practicals\Prac1.2 Subtration.py ============

FIRST MATRIX

Number Of Rows: 2

Number Of Columns: 2

Matrix 1-> R: 1 C: 1

4

Matrix 1-> R: 1 C: 2

8

Matrix 1-> R: 2 C: 1

9

Matrix 1-> R: 2 C: 2

3

[4, 8]

[9, 3]

SECOND MATRIX

Number Of Rows: 2

Number Of Columns: 2

Matrix 1-> R: 1 C: 1

2

Matrix 1-> R: 1 C: 2

6

Matrix 1-> R: 2 C: 1

3

Matrix 1-> R: 2 C: 2

6

[2, 6]

[3, 6]

[[4 8]

[9 3]]

[[2 6]

[3 6]]

[[ 2 2]

[ 6 -3]]

//division of matrix

import numpy as np

#First matrix

print("FIRST MATRIX")

matrix1 = []

rows = int(input("Number Of Rows: "))

cols = int(input("Number Of Columns: "))

for r in range(rows):

row = []

matrix1.append(row)

for c in range(cols):

row.append(int(input("Matrix 1-> R: {} C: {}\n".format(r+1, c+1))))

for i in matrix1:

print(i)

print("\n")

#Second Matrix

print("SECOND MATRIX")

matrix2 = []

rows2 = int(input("Number Of Rows: "))

cols2 = int(input("Number Of Columns: "))

for r in range(rows2):

lis = []

matrix2.append(lis)

for c in range(cols2):

lis.append(int(input("Matrix 1-> R: {} C: {}\n".format(r+1, c+1))))

for j in matrix2:

print(j)

print("\n")

addition=[[0 for x in range(cols2) ] for y in range(rows)]

for i in range(0,rows):

for j in range(0,rows2):

for k in range(0,cols):

addition[i][j]=matrix1[i][j]/matrix2[i][j]

m1=np.matrix(matrix1)

m2=np.matrix(matrix2)

m3=np.matrix(addition)

print(m1)

print("\n")

print(m2)

print("\n")

print(m3)

output:

FIRST MATRIX

Number Of Rows: 2

Number Of Columns: 2

Matrix 1-> R: 1 C: 1

3

Matrix 1-> R: 1 C: 2

5

Matrix 1-> R: 2 C: 1

7

Matrix 1-> R: 2 C: 2

2

[3, 5]

[7, 2]

SECOND MATRIX

Number Of Rows: 2

Number Of Columns: 2

Matrix 1-> R: 1 C: 1

3

Matrix 1-> R: 1 C: 2

4

Matrix 1-> R: 2 C: 1

6

Matrix 1-> R: 2 C: 2

8

[3, 4]

[6, 8]

[[3 5]

[7 2]]

[[3 4]

[6 8]]

[[1. 1.25 ]

[1.16666667 0.25 ]]

//multiplication of matrix

import numpy as np

r1=int(input('\nEnter the no. of rows of matrix I : '))

c1=int(input('Enter the no. of columns of matrix I : '))

print('\nEnter ',r1\*c1,' elements of matrix')

M1= [[0 for x in range(c1)] for y in range(r1)]

for i in range(0,r1):

for j in range(0,c1):

M1[i][j]=int(input())

r2=int(input('\nEnter the no. of rows of matrix II : '))

c2=int(input('Enter the no. of columns of matrix II : '))

print('\nEnter ',r2\*c2,' elements of matrix')

M2= [[0 for x in range(c2)] for y in range(r2)]

for i in range(0,r2):

for j in range(0,c2):

M2[i][j]=int(input())

multi= [[0 for x in range(c2)] for y in range(r1)]

for i in range(0,r1):

for j in range(0,c2):

for k in range(0,c1):

multi[i][j]=multi[i][j]+M1[i][k]\*M2[k][j]

m1=np.matrix(M1)

m2=np.matrix(M2)

multi1=np.matrix(multi)

print("\nMatrix I")

print(m1)

print("\nMatrix II")

print(m2)

print("\nMultiplication of Matrix I and Matrix II")

print(multi1)

OUTPUT:

========== RESTART: D:\Python Practicals\Prac1.3 Multiplication.py ==========

Enter the no. of rows of matrix I : 2

Enter the no. of columns of matrix I : 2

Enter 4 elements of matrix

3

5

7

4

Enter the no. of rows of matrix II : 2

Enter the no. of columns of matrix II : 2

Enter 4 elements of matrix

2

1

4

3

Matrix I

[[3 5]

[7 4]]

Matrix II

[[2 1]

[4 3]]

Multiplication of Matrix I and Matrix II

[[26 18]

[30 19]]

**Practical 2**

Aim : Write Python program to sort n names using Quick sort algorithm. Discuss the complexity of algorithm used.

def QuickSort(alist,start,end):

if start<end:

loc=partition(alist,start,end)

QuickSort(alist,start,loc)

QuickSort(alist,loc+1,end)

def partition (alist,start,end):

pivot=alist[start]

left=start+1

loc=end

flag=0

while flag!=1:

while left<=loc and alist[left]<=pivot:

left=left+1

while alist[loc]>=pivot and loc>=left:

loc=loc-1

if loc<left:

flag=1

else:

temp=alist[left]

alist[left]=alist[loc]

alist[loc]=temp

temp=alist[start]

alist[start]=alist[loc]

alist[loc]=temp

return loc

alist=[]

n=int(input("Enter number of elements:"))

while(n>0):

x=int(input("Enter elements:"))

alist.append(x)

n=n-1

print("Original list:",alist)

QuickSort(alist,0,len(alist)-1)

print("Sorted list:",alist)

OUTPUT:

============== RESTART: D:\Python Practicals\Prac2 QuickSort.py ==============

Enter number of elements:5

Enter elements:7

Enter elements:5

Enter elements:9

Enter elements:4

Enter elements:2

Original list: [7, 5, 9, 4, 2]

Sorted list: [2, 4, 5, 7, 9]

**Practical 3**

Aim: Write Python program to sort n numbers using Merge sort algorithm. Discuss the complexity of algorithm used.

def mergeSort(mylist):

print("splitting ",mylist)

if len(mylist)>1:

mid=len(mylist)//2

left=mylist[ :mid]

right=mylist[mid: ]

mergeSort(left)

mergeSort(right)

i=0

j=0

k=0

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

if left[i]<right[j]:

mylist[k]=left[i]

i=i+1

else:

mylist[k]=right[j]

j=j+1

k=k+1

while i<len(left):

mylist[k]=left[i]

i=i+1

k=k+1

while j<len(right):

mylist[k]=right[j]

j=j+1

k=k+1

mylist=[ ]

n=int(input("Enter the elements in the list: "))

while(n>0):

x=int(input("Enter elements:"))

mylist.append(x)

n=n-1

print("Original list ",mylist)

mergeSort(mylist)

print("Sorted list ",mylist)

OUTPUT:

============== RESTART: D:\Python Practicals\Prac3 MergeSort.py ==============

Enter the elements in the list: 5

Enter elements:6

Enter elements:8

Enter elements:4

Enter elements:9

Enter elements:3

Original list [6, 8, 4, 9, 3]

splitting [6, 8, 4, 9, 3]

splitting [6, 8]

splitting [6]

splitting [8]

splitting [4, 9, 3]

splitting [4]

splitting [9, 3]

splitting [9]

splitting [3]

Sorted list [3, 4, 6, 8, 9]

>>>

**Practical No: 4**

Aim: Write Python Program For Inserting An Element Into Binary Tree.

Code:

from drawtree import draw\_bst

class Node:

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

self.left=None

self.right=None

self.val=key

def insert(root,node):

if root is None:

root=node

else:

if root.val<node.val:

if root.right is None:

root.right=node

else:

insert(root.right,node)

else:

if root.left is None:

root.left=node

else:

insert(root.left,node)

def inorder(root):

if root:

inorder(root.left)

order.append(root.val)

inorder(root.right)

order=[]

a=int(input("Enter the root of the tree: "))

r=Node(a)

l=[a]

draw\_bst(l)

ch=str(input("Do you want to insert more node?(y/n): "))

while ch=='y':

a=int(input("Enter the node to insert: "))

insert(r,Node(a))

l.append(a)

draw\_bst(l)

ch=str(input("Do you want to insert more node?(y/n): "))

inorder(r)

print('\nInorder: ')

print(order)

Output:

====================== RESTART: D:/Foa Pracs/Pract4.py ======================

Enter the root of the tree: 7

7

Do you want to insert more node?(y/n): y

Enter the node to insert: 5

7

/

5

Do you want to insert more node?(y/n): y

Enter the node to insert: 8

7

/ \

5 8

Do you want to insert more node?(y/n): y

Enter the node to insert: 2

7

/ \

5 8

/

2

Do you want to insert more node?(y/n): y

Enter the node to insert: 4

7

/ \

5 8

/

2

\

4

Do you want to insert more node?(y/n): y

Enter the node to insert: 6

7

/ \

5 8

/ \

2 6

\

4

Do you want to insert more node?(y/n): n

Inorder:

[2, 4, 5, 6, 7, 8]

**Practical No: 5**

Aim: Write Python program for deleting an element (assuming data is given) from binary tree.

Code:

class Node:

# Constructor to create a new node

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

self.key = key

self.left = None

self.right = None

# A utility function to do inorder traversal of BST

def inorder(root):

if root is not None:

inorder(root.left)

print (root.key)

inorder(root.right)

# A utility function to insert a new node with given key in BST

def insert( node, key):

# If the tree is empty, return a new node

if node is None:

return Node(key)

# Otherwise recur down the tree

if key < node.key:

node.left = insert(node.left, key)

else:

node.right = insert(node.right, key)

# return the (unchanged) node pointer

return node

# Given a non-empty binary search tree, return the node

# with minum key value found in that tree. Note that the

# entire tree does not need to be searched

def minValueNode( node):

current = node

# loop down to find the leftmost leaf

while(current.left is not None):

current = current.left

return current

# Given a binary search tree and a key, this function

# delete the key and returns the new root

def deleteNode(root, key):

# Base Case

if root is None:

return root

# If the key to be deleted is smaller than the root's

# key then it lies in left subtree

if key < root.key:

root.left = deleteNode(root.left, key)

# If the kye to be delete is greater than the root's key

# then it lies in right subtree

elif(key > root.key):

root.right = deleteNode(root.right, key)

# If key is same as root's key, then this is the node

# to be deleted

else:

# Node with only one child or no child

if root.left is None :

temp = root.right

root = None

return temp

elif root.right is None :

temp = root.left

root = None

return temp

# Node with two children: Get the inorder successor

# (smallest in the right subtree)

temp=minValueNode(root.right)

# Copy the inorder successor's content to this node

root.key = temp.key

# Delete the inorder successor

root.right = deleteNode(root.right , temp.key)

return root

root = None

root = insert(root, 50)

root = insert(root, 30)

root = insert(root, 20)

root = insert(root, 40)

root = insert(root, 70)

root = insert(root, 60)

root = insert(root, 80)

print ("Inorder traversal of the given tree")

inorder(root)

ch=input("Do you want to delete anything(y/n) :")

while ch=="y":

num=int(input("Enter number to delete:"))

root = deleteNode(root, num)

print ("\nDeleted ",num)

if root is None:

ch='n'

print ("\nTree is empty now...\n")

else:

print("tree after deletion of element is")

inorder(root)

ch=input("Do you want to delete anything(y/n) :")

output:

====================== RESTART: D:/Foa Pracs/Pract5.py ======================

Inorder traversal of the given tree

20

30

40

50

60

70

80

Do you want to delete anything(y/n) :y

Enter number to delete:60

Deleted 60

tree after deletion of element is

20

30

40

50

70

80

Do you want to delete anything(y/n) :y

Enter number to delete:50

Deleted 50

tree after deletion of element is

20

30

40

70

80

Do you want to delete anything(y/n) :n

**Practical No: 6**

Aim: Write Python program for checking whether a given graph G has simple path from source to destination.

Code:

# program to check if there is exist a path between two vertices of a graph.

from collections import defaultdict

#This class represents a directed graph using adjacency list representation

class Graph:

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

self.V= vertices #No. of vertices

self.graph = defaultdict(list) # default dictionary to store graph

# function to add an edge to graph

def addEdge(self,u,v):

self.graph[u].append(v)

# Use BFS to check path between s and d

def isReachable(self, s, d):

# Mark all the vertices as not visited

visited =[False]\*(self.V)

# Create a queue for BFS

queue=[]

# Mark the source node as visited and enqueue it

queue.append(s)

visited[s] = True

while queue:

#Dequeue a vertex from queue

n = queue.pop(0)

# If this adjacent node is the destination node,

# then return true

if n == d:

return True

#Else, continue to do BFS

for i in self.graph[n]:

if visited[i] == False:

queue.append(i)

visited[i] = True

# If BFS is complete without visited d

return False

# Create a graph given in the above diagram

g = Graph(4)

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

g.addEdge(2, 3)

g.addEdge(3, 3)

u =0; v = 3

if g.isReachable(u, v):

print("There is a path from %d to %d" % (u,v))

else :

print("There is no path from %d to %d" % (u,v))

u = 3; v = 1

if g.isReachable(u, v) :

print("There is a path from %d to %d" % (u,v))

else :

print("There is no path from %d to %d" % (u,v))

Output:

===================== RESTART: D:/Foa Pracs/Pract6.py ======================

There is a path from 0 to 3

There is no path from 3 to 1

>>>

**Practical No: 7**

Aim : Write Python program for finding the smallest and largest elements in an array A of size n using Selection algorithm.

Discuss Time complexity.

Code:

import sys

A=[]

p=int(input("Enter number of elements : "))

n=p

while(n>0):

x=int(input("Enter element : "))

A.append(x)

n=n-1

# Traverse through all array elements

for i in range(len(A)):

# Find the minimum element in remaining

# unsorted array

min\_idx = i

for j in range(i+1, len(A)):

if A[min\_idx] > A[j]:

min\_idx = j

A[i], A[min\_idx] =A[min\_idx], A[i]

# Swap the found minimum element with

# the first element

print("Smallest element : %d" %int(A[0]))

print("Largest element : %d" %int(A[p-1]))

print ("Sorted array in Ascending Order: ",A)

Output:

======================= RESTART: D:\pract7 FOA.py =======================

Enter number of elements : 5

Enter element : 53

Enter element : 24

Enter element : 20

Enter element : 86

Enter element : 35

Smallest element : 20

Largest element : 86

Sorted array in Ascending Order: [20, 24, 35, 53, 86]

>>>

**Practical No: 8**

Aim: Write Python program for finding the second largest element in an array A of size n using Tournament Method. Discuss Time Complexity.

Code:

def second\_smallest\_in\_array(A):

comparisonCount = 0

#indexes that are to be compared

idx=range(0,len(A))

#list of knockout for all elements

knockout=[[] for i in idx]

#play tournaments,until we have only one node left

while len(idx)>1:

#index of nodes that win this tournament

idx1=[]

#nodes in idx odd,if yes then last automatically goes to next round

odd=len(idx)%2

#iterate over even indexes,as we do a paired tournament

for i in range(0,len(idx)- odd,2):

firstIndex=idx[i]

secondIndex=idx[i+1]

comparisonCount+= 1

# perform tournament

if A[firstIndex]<A[secondIndex]:

#firstIndex qualifies for next round

idx1.append(firstIndex)

#add A[secondIndex] to knockout list of firstIndex

knockout[firstIndex].append(A[secondIndex])

else:

idx1.append(secondIndex)

knockout[secondIndex].append(A[firstIndex])

if odd == 1:

idx1.append(idx[i+2])

#perform new tournament

idx = idx1

print("Smallest element=",A[idx[0]])

print("Total comparasion",comparisonCount)

print("Nodes knowed off by the smallest=",knockout[idx[0]],"\n")

#compute second smallest

a=knockout[idx[0]]

if len(a)>0:

v=a[0]

for i in range(1,len(a)):

comparisonCount+=1

if v > a[i]:

v=a[i]

print("Second smallest element=",v)

print("Total comparision=",comparisonCount)

n=int(input("Input how many elements in array??: "))

A=[]

print("Enter numbers in array: ")

for i in range(n):

m=input()

A.append(m)

print("array is: ",A)

print(second\_smallest\_in\_array(A))

Output:

======================RESTART: D:/Pract8\_FOA.py =========================

Input how many elements in array??: 10

Enter numbers in array:

2

4

3

7

3

0

8

4

11

1

array is: ['2', '4', '3', '7', '3', '0', '8', '4', '11', '1']

Smallest element= 0

Total comparasion 9

Nodes knowed off by the smallest= ['3', '4', '2', '1']

Second smallest element= 1

Total comparision= 12

None

>>>

**Practical No : 9**

Write Python program for implementing Huffman Coding Algorithm. Discuss the complexity of algorithm.

Code:

import heapq

import time

from collections import defaultdict

start=time.time()

def encode(frequency):

heap = [[weight, [symbol, '']] for symbol, weight in frequency.items()]

print("\nHeap:",heap)

heapq.heapify(heap)

print("\nI am heap:",heapq)

while len(heap) > 1:

lo = heapq.heappop(heap)

print("\nLOW:",lo)

hi = heapq.heappop(heap)

print("HIGH:",hi)

for pair in lo[1:]:

pair[1] = '0' + pair[1]

for pair in hi[1:]:

pair[1] = '1' + pair[1]

heapq.heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])

return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))

data = ("MyString")

frequency = defaultdict(int)

for symbol in data:

frequency[symbol] += 1

print("\nFreq:",frequency)

huff = encode(frequency)

print ("\nSymbol".ljust(10) + "Weight".ljust(10) + "Huffman Code")

for p in huff:

print (p[0].ljust(10) + str(frequency[p[0]]).ljust(10) + p[1])

end=time.time()

diff=end-start

print("Time Complexity",diff)  
Output:

================= RESTART: C:/Users/CS21/Desktop/huffman.py =================

Freq: defaultdict(<class 'int'>, {'M': 1})

Freq: defaultdict(<class 'int'>, {'M': 1, 'y': 1})

Freq: defaultdict(<class 'int'>, {'M': 1, 'y': 1, 'S': 1})

Freq: defaultdict(<class 'int'>, {'M': 1, 'y': 1, 'S': 1, 't': 1})

Freq: defaultdict(<class 'int'>, {'M': 1, 'y': 1, 'S': 1, 't': 1, 'r': 1})

Freq: defaultdict(<class 'int'>, {'M': 1, 'y': 1, 'S': 1, 't': 1, 'r': 1, 'i': 1})

Freq: defaultdict(<class 'int'>, {'M': 1, 'y': 1, 'S': 1, 't': 1, 'r': 1, 'i': 1, 'n': 1})

Freq: defaultdict(<class 'int'>, {'M': 1, 'y': 1, 'S': 1, 't': 1, 'r': 1, 'i': 1, 'n': 1, 'g': 1})

Heap: [[1, ['M', '']], [1, ['y', '']], [1, ['S', '']], [1, ['t', '']], [1, ['r', '']], [1, ['i', '']], [1, ['n', '']], [1, ['g', '']]]

I am heap: <module 'heapq' from 'C:\\Users\\CS21\\AppData\\Local\\Programs\\Python\\Python37-32\\lib\\heapq.py'>

LOW: [1, ['M', '']]

HIGH: [1, ['S', '']]

LOW: [1, ['g', '']]

HIGH: [1, ['i', '']]

LOW: [1, ['n', '']]

HIGH: [1, ['r', '']]

LOW: [1, ['t', '']]

HIGH: [1, ['y', '']]

LOW: [2, ['M', '0'], ['S', '1']]

HIGH: [2, ['g', '0'], ['i', '1']]

LOW: [2, ['n', '0'], ['r', '1']]

HIGH: [2, ['t', '0'], ['y', '1']]

LOW: [4, ['M', '00'], ['S', '01'], ['g', '10'], ['i', '11']]

HIGH: [4, ['n', '00'], ['r', '01'], ['t', '10'], ['y', '11']]

Symbol Weight Huffman Code

M 1 000

S 1 001

g 1 010

i 1 011

n 1 100

r 1 101

t 1 110

y 1 111

Time Complexity 0.18458199501037598

**Practical 10**

Write Python program for implementing Strassen's Matrix multiplication using Divide and Conquer method. Discuss the complexity of algorithm.

Code:

import numpy as np

import time

start=time.time()

N=int(input('\nEnter the value of N for NxN matrix : '))

#checking condition

if N % 2 != 0:

print("Not Possible.")

else:

print('\nEnter ',N\*N,' elements of matrix A ')

A= [[0 for x in range(N)] for y in range(N)]

for i in range(0,N):

for j in range(0,N):

A[i][j]=int(input())

print('\nEnter ',N\*N,' elements of matrix B ')

B= [[0 for x in range(N)] for y in range(N)]

for i in range(0,N):

for j in range(0,N):

B[i][j]=int(input())

a= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

b= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

c= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

d= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

a[i][j]=A[i][j]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

b[i][j]=A[i][j+int(N/2)]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

c[i][j]=A[i+int(N/2)][j]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

d[i][j]=A[i+int(N/2)][j+int(N/2)]

a=np.matrix(a);

b=np.matrix(b);

c=np.matrix(c);

d=np.matrix(d);

e= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

f= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

g= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

h= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

e[i][j]=B[i][j]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

f[i][j]=B[i][j+int(N/2)]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

g[i][j]=B[i+int(N/2)][j]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

h[i][j]=B[i+int(N/2)][j+int(N/2)]

e=np.matrix(e);

f=np.matrix(f);

g=np.matrix(g);

h=np.matrix(h);

p1= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

p2= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

p3= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

p4= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

p5= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

p6= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

p7= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

p1=np.matmul(a,(f-h))

p2=np.matmul((a+b),h)

p3=np.matmul((c+d),e)

p4=np.matmul(d,(g-e))

p5=np.matmul((a+d),(e+h))

p6=np.matmul((b-d),(g+h))

p7=np.matmul((a-c),(e+f))

d1= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

d2= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

d3= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

d4= [[0 for x in range(int(N/2))] for y in range(int(N/2))]

d1=p5+p4-p2+p6

d2=p1+p2

d3=p3+p4

d4=p1+p5-p3-p7

d1=np.matrix(d1);

d2=np.matrix(d2);

d3=np.matrix(d3);

d4=np.matrix(d4);

C= [[0 for x in range(N)] for y in range(N)]

for i in range(0,int(N/2)):

for j in range(0,int(N/2)):

C[i][j]=d1.item(i,j)

for i in range(0,int(N/2)):

for j in range(int(N/2),N):

C[i][j]=d2.item(i,j-int(N/2))

for i in range(int(N/2),N):

for j in range(0,int(N/2)):

C[i][j]=d3.item(i-int(N/2),j)

for i in range(int(N/2),N):

for j in range(int(N/2),N):

C[i][j]=d4.item(i-int(N/2),j-int(N/2))

print("\nMatrix A")

print(np.matrix(A))

print("\nMatrix B")

print(np.matrix(B))

print("\nMatrix C")

print(np.matrix(C))

end=time.time()

diff=end-start

print("Time Complexity Of Algorithm: ", diff)

Output:

================ RESTART: C:\Users\CS21\Desktop\STRESSANS.py ================

Enter the value of N for NxN matrix : 2

Enter 4 elements of matrix A

2

3

4

5

Enter 4 elements of matrix B

6

7

8

9

Matrix A

[[2 3]

[4 5]]

Matrix B

[[6 7]

[8 9]]

Matrix C

[[36 41]

[64 73]]

Time Complexity Of Algorithm: 8.265885353088379

>>>