M.Sc C.S - I
SEM I
E-Journal

Roll No.	**006**
Name	**HEMAN SHAKTHI MOHAN UDAIYAR**
Subject	**DATA DESIGN ALGORITHM**

Thakur Educational Trust's (Regd.)
Thakur College of Science & Commerce
UGC Recognised • Affiliated to University Of Mumbai
(NAAC Accredited with Grade "A" [3rd Cycle] & ISO 9001:2015 Certified)

CERTIFICATE

This is here to certify that Mr./~~Ms~~. HEMAN SHAKTHI MOHAN UDAIYAR, Seat Number 006 of M.Sc. I Computer Science, has satisfactorily completed the required number of experiments prescribed by the UNIVERSITY OF MUMBAI during the academic year 2021 – 2022.

Date:
Place: Mumbai

Teacher In-Charge Head of Department

External Examiner

INDEX

PRACTICAL NO: 1

27-08-2021

PRACTICAL NO. 1

Aim : write a program to implement insertion sort and find the running time of the algorithm

Example:

Input : [53, 16, 65, 21, 43, 34]
output : [16, 21, 34, 43, 53, 65]

Algorithm :

```
for i = 2 to list length
    val = list[i]
    x = i-1
    while x > 0 and list[x] > val
        list[x+1] = list[x]
        x = x - 1
    list[x+1] = val
```

code :

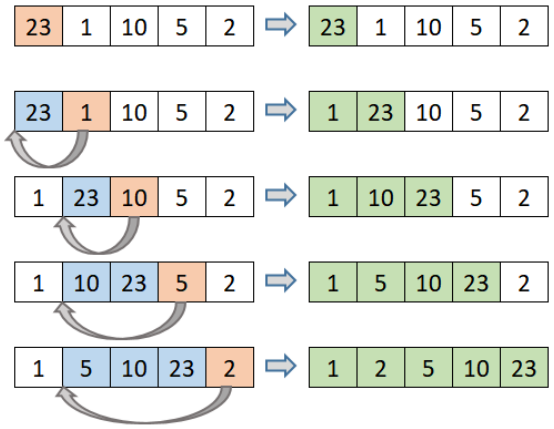
```
def insertionSortAlgo(arr):
    for i in range(1, len(mylist)):
        key = mylist[i]
        j = i-1
        while j>=0 and key < mylist[j]:
            mylist[j+1] = mylist[j]
            j -= 1
        mylist[j+1] = key
mylist = input('Enter the values :').split()
mylist = [int(x) for x in mylist]
insertionSortAlgo(mylist)
print('Sorted Array is', mylist)
```

conclusion : Running time of insertion sort O(n^2)

Practical 1 - InsertionSort.py - C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practic...

File Edit Format Run Options Window Help

```
def insertionSortAlgo(arr):
        for i in range(1, len(mylist)):
                key = mylist[i]
                j = i-1
                while j >=0 and key < mylist[j] :
                                mylist[j+1] = mylist[j]
                                j -= 1
                mylist[j+1] = key

mylist = input('Enter the values: ').split()
mylist = [int(x) for x in mylist]
insertionSortAlgo(mylist)
print ("Sorted Array is:",mylist)
```



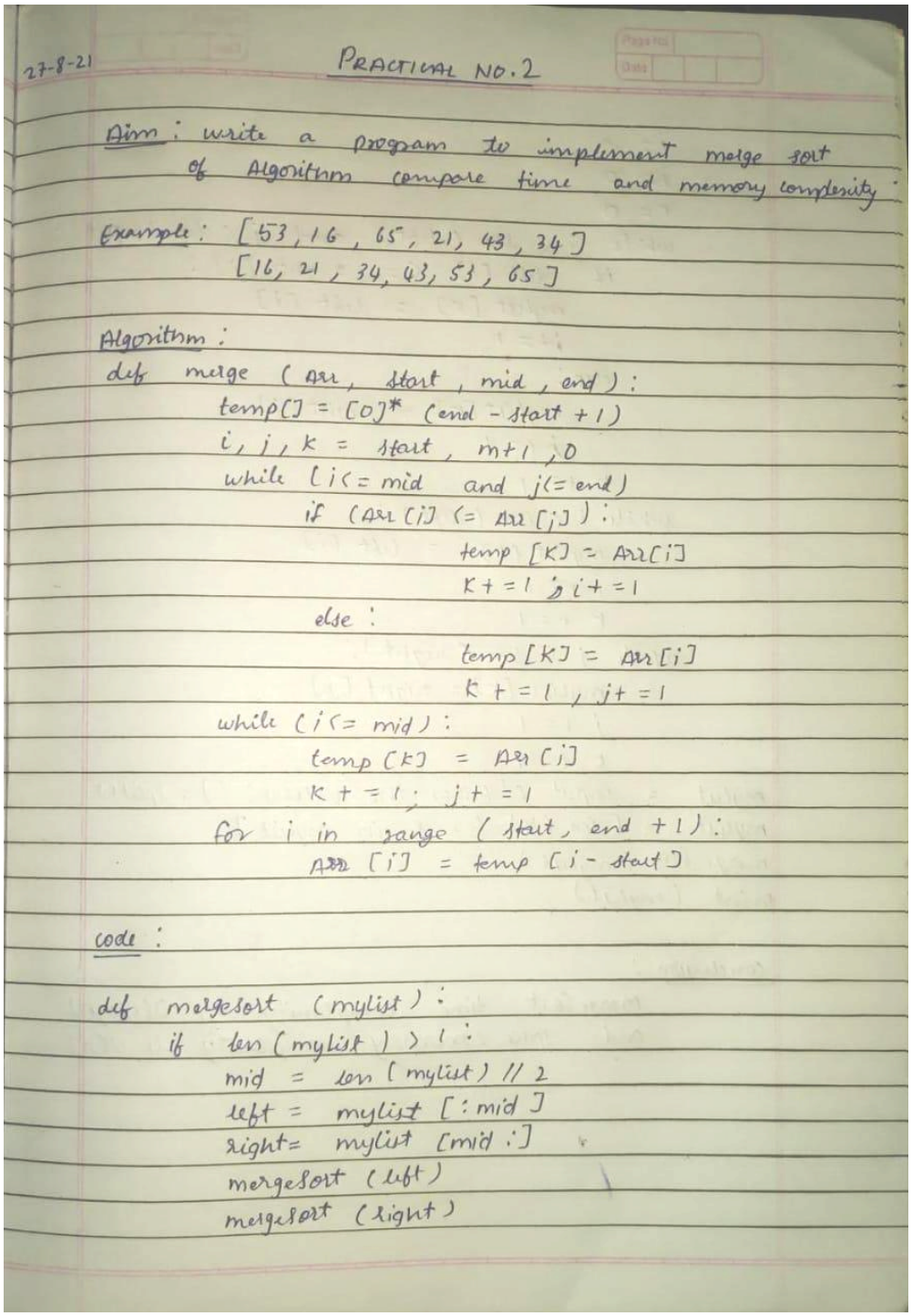
OUTPUT :

IDLE Shell 3.9.7

File Edit Shell Debug Options Window Help

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practical 1 - Inse
rtionSort.py
Enter the values: 23 1 10 5 2
Sorted Array is: [1, 2, 5, 10, 23]
>>>
```

PRACTICAL NO: 2A



27-8-21

PRACTICAL NO. 2

Aim : write a program to implement merge sort of Algorithm compare time and memory complexity.

Example : [53, 16, 65, 21, 43, 34]
[16, 21, 34, 43, 53, 65]

Algorithm :

```
def merge (Arr, start, mid, end):
    temp[] = [0]* (end - start + 1)
    i, j, k = start, m+1, 0
    while (i<= mid and j<= end)
        if (Arr[i] <= Arr[j]):
            temp [K] = Arr[i]
            K+=1 ; i+=1
        else :
            temp [K] = Arr[i]
            K+=1 , j+=1
    while (i<= mid):
        temp [k] = Arr[i]
        K+=1 ; j+=1
    for i in range (start, end+1):
        Arr [i] = temp [i - start]
```

code :

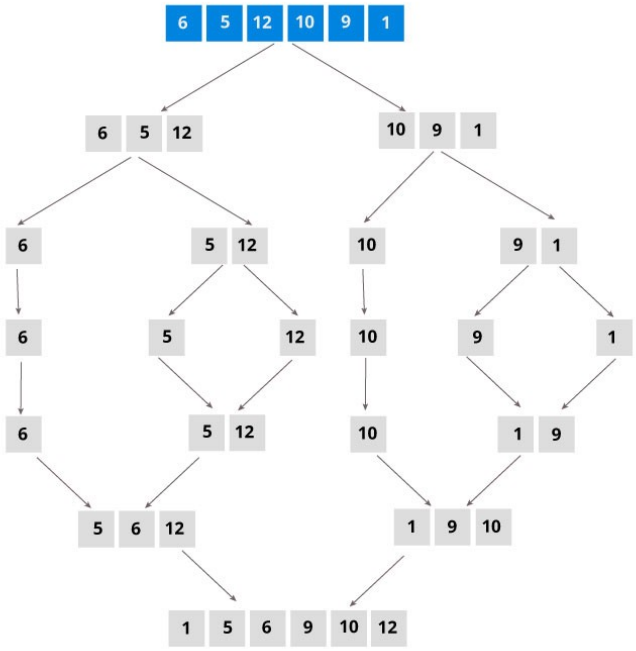
```
def mergesort (mylist):
    if len(mylist) > 1:
        mid = len(mylist) // 2
        left = mylist [:mid]
        right= mylist [mid:]
        mergesort (left)
        mergesort (right)
```

Practical 2 - MergeSort.py - C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practical ...

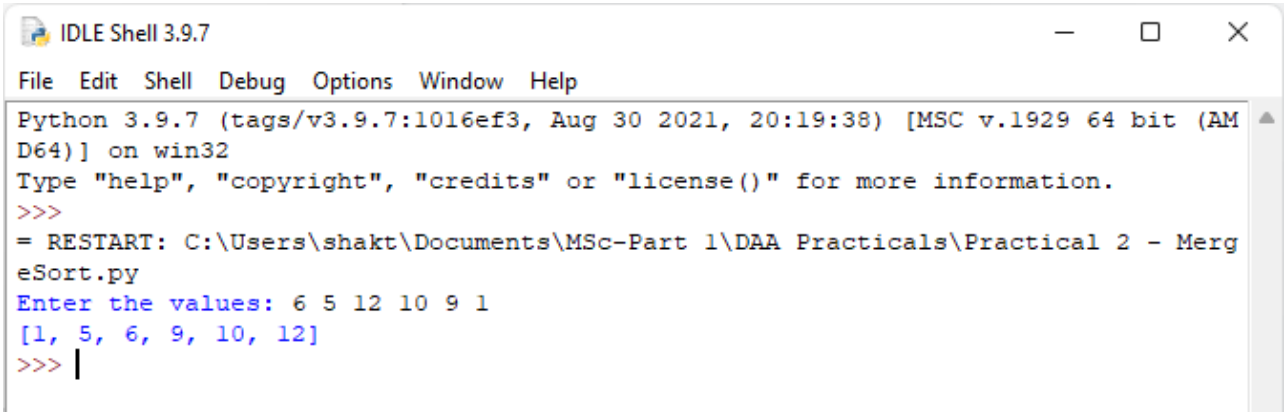
File Edit Format Run Options Window Help

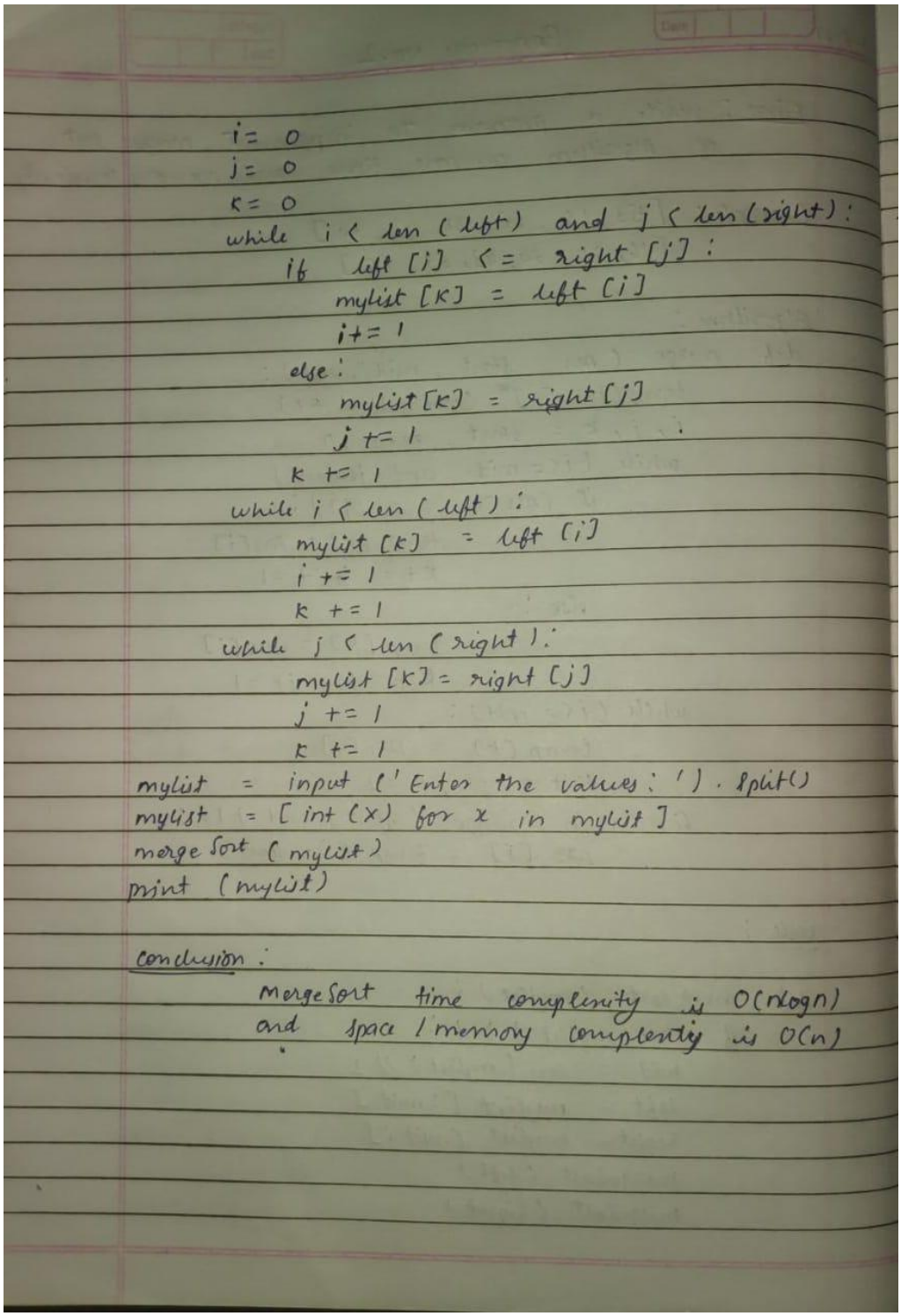
```
def mergeSort(myList):
    if len(myList) > 1:
        mid = len(myList) // 2
        left = myList[:mid]
        right = myList[mid:]
        mergeSort(left)
        mergeSort(right)
        i = 0
        j = 0
        k = 0
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
              myList[k] = left[i]
              i += 1
            else:
                myList[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            myList[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            myList[k]=right[j]
            j += 1
            k += 1
myList = input('Enter the values: ').split()
myList = [int(x) for x in myList]
mergeSort(myList)
print(myList)
```

Ln: 1 Col: 0


6 5 12 10 9 1
6 5 12
10 9 1
6
5 12
10
9 1
6
5
12
10
9
1
6
5 12
10
1 9
5 6 12
1 9 10
1 5 6 9 10 12

OUTPUT :


IDLE Shell 3.9.7
File Edit Shell Debug Options Window Help
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practical 2 - MergeSort.py
Enter the values: 6 5 12 10 9 1
[1, 5, 6, 9, 10, 12]
>>>



```
i = 0
j = 0
k = 0
while i < len(left) and j < len(right):
    if left[j] <= right[j]:
        mylist[k] = left[i]
        i+= 1
    else:
        mylist[k] = right[j]
        j += 1
    k += 1
while i < len(left):
    mylist[k] = left[i]
    i += 1
    k += 1
while j < len(right):
    mylist[k] = right[j]
    j += 1
    k += 1
mylist = input('Enter the values: ').split()
mylist = [int(x) for x in mylist]
mergeSort(mylist)
print(mylist)
```

Conclusion:

MergeSort time complexity is O(nlogn) and space / memory complexity is O(n)

PRACTICAL NO: 2B

Compare the time and memory complexity

```
>>>
= RESTART: C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practical 1 - Inse
rtionSort.py
Sorted Array is: [1, 3, 7, 12, 13, 14, 16, 18, 21, 21, 24, 33, 36, 36, 38, 40, 4
1, 42, 42, 42, 44, 45, 50, 50, 56, 59, 60, 63, 64, 66, 66, 67, 67, 68, 69, 70, 7
2, 73, 75, 76, 80, 80, 82, 85, 86, 90, 93, 93, 95, 98, 99, 104, 106, 106, 109, 1
17, 118, 120, 123, 125, 125, 125, 125, 129, 133, 139, 144, 150, 151, 153, 158, 1
59, 162, 162, 167, 168, 168, 171, 172, 175, 175, 180, 187, 188, 193, 194, 197, 2
06, 207, 219, 220, 221, 221, 226, 228, 238, 239, 241, 243, 245]

Runtime of the program is 0.0009989738464355469
Consumed memory in MB :  23.33203125 MB
>>>
```

```
>>>
= RESTART: C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practical 2 - MergeSort.py
Sorted Array is: [1, 3, 7, 12, 13, 14, 16, 18, 21, 21, 24, 33, 36, 36, 38, 40, 41, 42, 42, 42
, 44, 45, 50, 50, 56, 59, 60, 63, 64, 66, 66, 67, 67, 68, 69, 70, 72, 73, 75, 76, 80, 80, 82,
85, 86, 90, 93, 93, 95, 98, 99, 104, 106, 106, 109, 117, 118, 120, 123, 125, 125, 125, 125, 1
29, 133, 139, 144, 150, 151, 153, 158, 159, 162, 162, 167, 168, 168, 171, 172, 175, 175, 180,
187, 188, 193, 194, 197, 206, 207, 219, 220, 221, 221, 226, 228, 238, 239, 241, 243, 245]

Runtime of the program is 0.0009999275207519531
Consumed memory in MB :  23.265625 MB
>>>
```

PRACTICAL NO: 3

05-09-2021

PRACTICAL NO. 3

Aim: write a program on strassen's algorithm for matrix multiplication and analyse its complexity

Example: [57 41 39 75] [69 54 12 4]

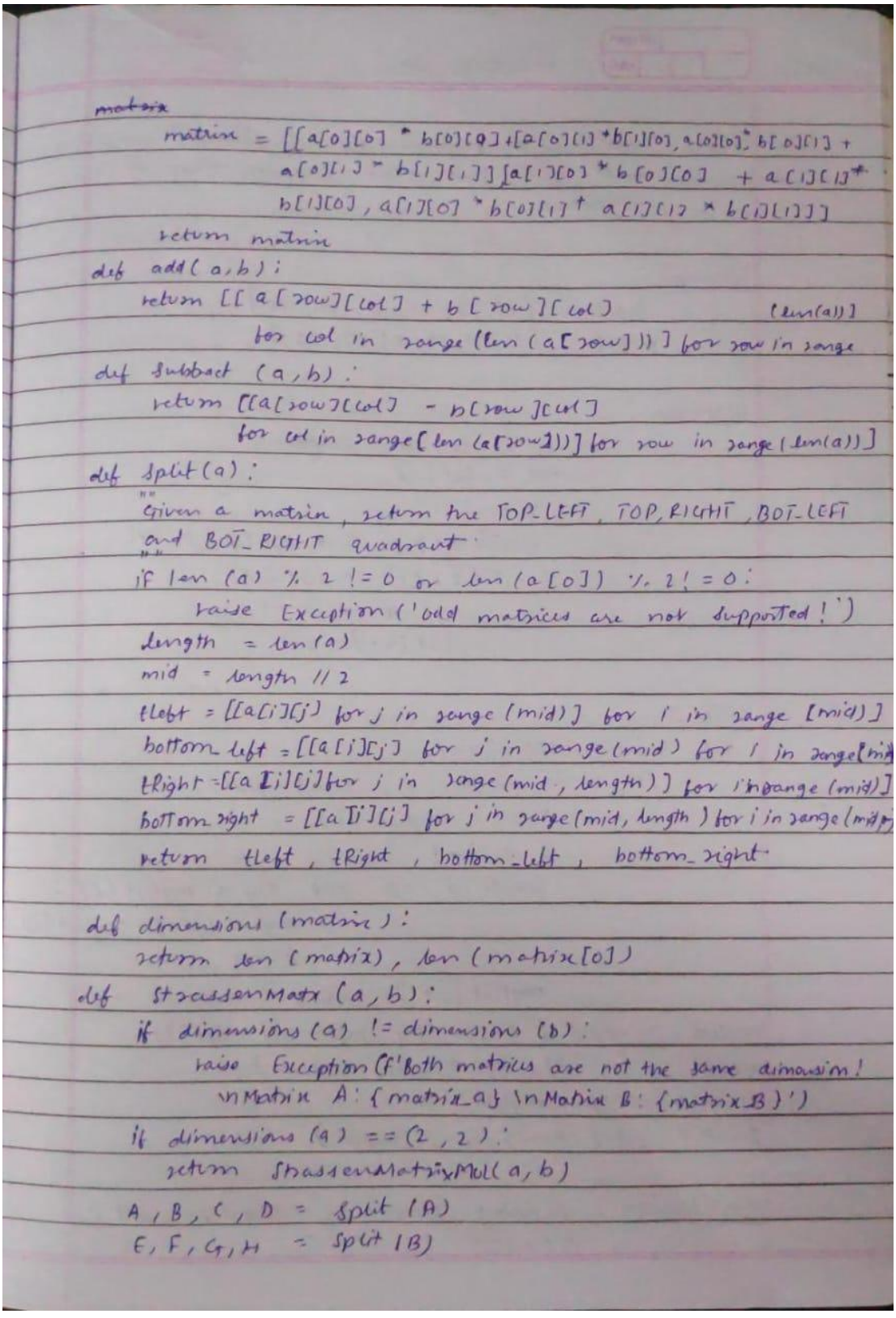
output: [5409 , 3942]
[876 , 664]
3242

Algorithm:

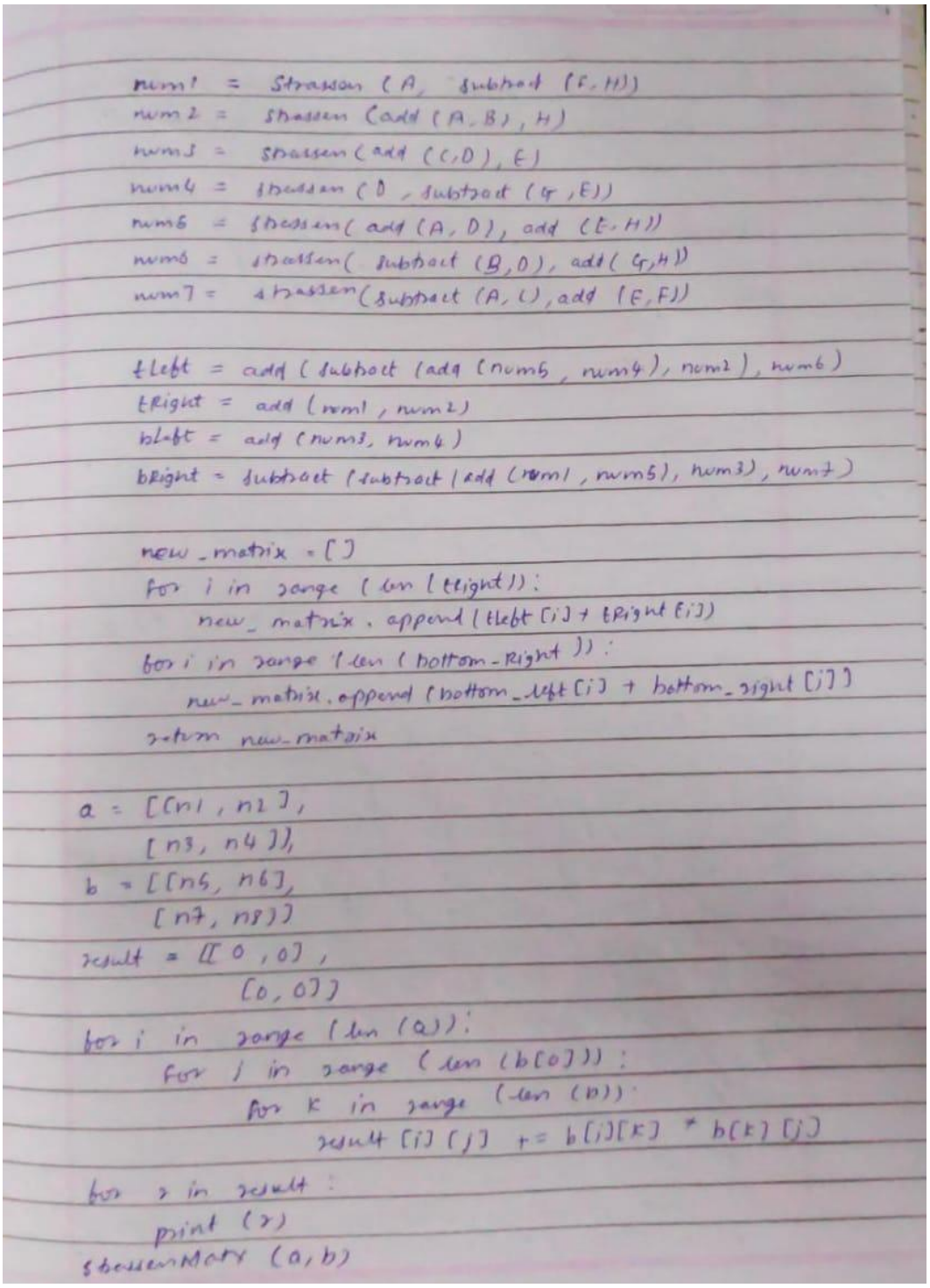
```
for i = 1 to a do
    for j = 1 to b do
        result [ i,j ] := 0
        for k = 1 to c do
            result [ i,j ] := result [i,j] + M1[i,k] x M2[k,j]
```

Code:

```
print ('Enter value for 2x2 matrix :')
n1 = int (input (' value for A :'))
n2 = int (input (' value for B :'))
n3 = int (input (' value for C :'))
n4 = int (input (' value for D :'))
n5 = int (input (' value for E :'))
n6 = int (input (' value for F :'))
n7 = int (input (' value for G :'))
n8 = int (input (' value for H :'))
def StrassenMatrixMul (a, b) :
    "only for 2x2 matrices
    " " "
    if len(a) != 2 or len(a[0]) != 2 or len(b[0])!=2:
        raise Exception ('matrices should be 2x2')
print (a[0][0] * b[0][1] + a[0][1] * b[1][1])
```



```
matrix
    matrix = [[a[0][0] * b[0][0]+[a[0][1]*b[1][0], a[0][0]* b[0][1] +
              a[0][1] * b[1][1]] [a[1][0] * b[0][0]  + a[1][1]*
              b[1][0], a[1][0] * b[0][1] + a[1][1] * b[1][1]]]
    return matrix
def add(a,b):
    return [[ a[row][col] + b[row][col]                    [len(a)]
            for col in range(len(a[row]))] for row in range
def subbact (a,b):
    return [[a[row][col]  - b[row][col]
            for col in range[len(a[row]))] for row in range(len(a))]
def split(a):
    ""
    Given a matrix, return the TOP-LEFT, TOP,RIGHIT ,BOT-LEFT
    and BOT_RIGHIT quadrant
    ""
    if len(a) % 2 != 0 or len(a[0]) % 2 != 0:
        raise Exception('odd matrices are not supported!')
    length = len(a)
    mid = length // 2
    tleft = [[a[i][j] for j in range(mid)] for i in range [mid)]
    bottom_left = [[a[i][j] for j in range(mid) for i in range[mi
    tRight =[[a[i][j]for j in range(mid, length)] for i in range(mid)]
    bottom_right = [[a[i][j] for j in range(mid, length) for i in range(mid
    return tleft, tRight, bottom_left, bottom_right

def dimensions(matrix):
    return len(matrix), len(matrix[0])
def StrassenMatrix(a,b):
    if dimensions(a) != dimensions(b):
        raise Exception(f'Both matrices are not the same dimension!
          \n Matrix A: {matrix_a} \n Matrix B: {matrix_B}')
    if dimensions(a) == (2,2):
        return StrassenMatrixMul(a,b)
A, B, C, D = split(A)
E, F, G, H = split(B)
```



```
num1 = Strassen (A, subtract (F, H))
num2 = Strassen (add (A, B), H)
num3 = Strassen (add (C, D), E)
num4 = Strassen (D, subtract (G, E))
num5 = Strassen (add (A, D), add (E, H))
num6 = Strassen (subtract (B, D), add (G, H))
num7 = Strassen (subtract (A, C), add (E, F))

tLeft = add (subtract (add (num5, num4), num2), num6)
tRight = add (num1, num2)
bLeft = add (num3, num4)
bRight = subtract (subtract (add (num1, num5), num3), num7)

new_matrix = []
for i in range (len (tRight)):
    new_matrix.append (tLeft[i] + tRight[i])
for i in range (len (bottom_Right)):
    new_matrix.append (bottom_left[i] + bottom_right[i])
return new_matrix

a = [[n1, n2],
     [n3, n4]],
b = [[n5, n6],
     [n7, n8]]
result = [[0, 0],
          [0, 0]]
for i in range (len (a)):
    for j in range (len (b[0])):
        for k in range (len (b)):
            result[i][j] += b[i][k] * b[k][j]

for r in result:
    print (r)
StrassenMatr (a, b)
```

Practical 3 - Strassen.py - C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practical 3 - Strassen.py (3.9.7)

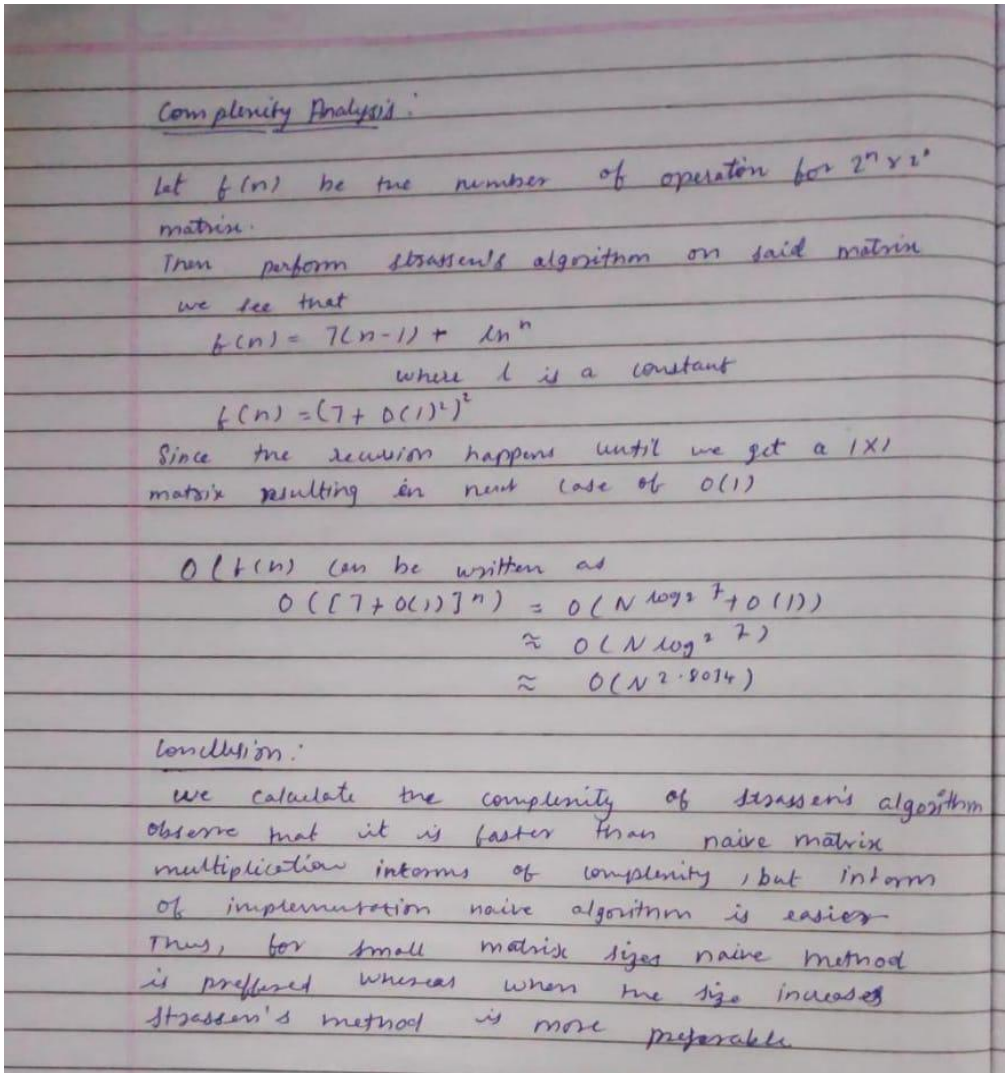
File Edit Format Run Options Window Help

```
print("Enter values for 2x2 matrix : ")
print("matrix a")
n1=int(input("Enter value for A : "))
n2=int(input("Enter value for B : "))
n3=int(input("Enter value for C : "))
n4=int(input("Enter value for D : "))
print("matrix b")
n5=int(input("Enter value for E : "))
n6=int(input("Enter value for F : "))
n7=int(input("Enter value for E : "))
n8=int(input("Enter value for F : "))
def StrassenMatrixM(a, b):
    """
    Only for 2x2 matrices
    """
    if len(a) != 2 or len(a[0]) != 2 or len(b) != 2 or len(b[0]) != 2:
        raise Exception('Matrices should be 2x2!')
    print(a[0][0] * b[0][1] + a[0][1] * b[1][1])
    matrix = [[a[0][0] * b[0][0] + a[0][1] * b[1][0], a[0][0] * b[0][1] + a[0][1] * b[1][1]],
                  [a[1][0] * b[0][0] + a[1][1] * b[1][0], a[1][0] * b[0][1] + a[1][1] * b[1][1]]]

    return matrix


def add(a, b):
    # print(matrix_a)
    return [[a[row][col] + b[row][col]
             for col in range(len(a[row]))] for row in range(len(a))]


def subtract(a, b):
    return [[a[row][col] - b[row][col]
             for col in range(len(a[row]))] for row in range(len(a))]


def split(a):
    """
    Given a matrix, return the TOP_LEFT, TOP_RIGHT, BOT_LEFT and BOT_RIGHT quadrant
    """
    if len(a) % 2 != 0 or len(a[0]) % 2 != 0:
        raise Exception('Odd matrices are not supported!')

    length = len(a)
    mid = length // 2
    tLeft = [[a[i][j] for j in range(mid)] for i in range(mid)]
    bottom_left = [[a[i][j] for j in range(mid)] for i in range(mid, length)]

    tRight = [[a[i][j] for j in range(mid, length)] for i in range(mid)]
    bottom_right = [[a[i][j] for j in range(mid, length)] for i in range(mid, length)]

    return tLeft, tRight, bottom_left, bottom_right


def dimensions(matrix):
    return len(matrix), len(matrix[0])
```

```
def strassenMatx(a, b):
    """
    Recursive function to calculate the product of two matrices, using the Strassen Algorithm.
    Currently only works for matrices of even length (2x2, 4x4, 6x6...etc)
    """
    if dimensions(a) != dimensions(b):
        raise Exception(f'Both matrices are not the same dimension! \nMatrix A:{matrix_a} \nMatrix B:{matrix_b}')
    if dimensions(a) == (2, 2):
        return StrassenMatrixM(a, b)

    A, B, C, D = split(a)
    E, F, G, H = split(b)

    num1 = strassen(A, subtract(F, H))
    num2 = strassen(add(A, B), H)
    num3 = strassen(add(C, D), E)
    num4 = strassen(D, subtract(G, E))
    num5 = strassen(add(A, D), add(E, H))
    num6 = strassen(subtract(B, D), add(G, H))
    num7 = strassen(subtract(A, C),add(E, F))

    tLeft = add(subtract(add(num5, num4), num2), num6)
    tRight = add(num1,num2)
    bLeft = add(num3, num4)
    bRight = subtract(subtract(add(num1, num5), num3), num7)

    # construct the new matrix from our 4 quadrants
    new_matrix = []
    for i in range(len(tRight)):
        new_matrix.append(tLeft[i] + tRight[i])
    for i in range(len(bottom_right)):
        new_matrix.append(bottom_left[i] + bottom_right[i])
    return new_matrix


a = [[n1,n2],
    [n3,n4]]
b = [[n5,n6],
    [n7,n8]]
result = [[0,0],
            [0,0]]
for i in range(len(a)):
       # iterate through columns of Y
       for j in range(len(b[0])):
           # iterate through rows of Y
           for k in range(len(b)):
               result[i][j] += b[i][k] * b[k][j]
for r in result:
    print(r)



strassenMatx(a,b)
```

OUTPUT :

```
>>>
= RESTART: C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals\Practical 3 - Stra
ssen.py
Enter values for 2x2 matrix :
matrix a
Enter value for A : 57
Enter value for B : 41
Enter value for C : 39
Enter value for D : 75
matrix b
Enter value for E : 69
Enter value for F : 54
Enter value for E : 12
Enter value for F : 4
[5409, 3942]
[876, 664]
3242
>>>
```



Complexity Analysis:

Let $f(n)$ be the number of operation for $2^n \times 2^n$ matrix.

Then perform Strassen's algorithm on said matrix we see that

$f(n) = 7(n-1) + 4^n$

where 1 is a constant

$f(n) = (7 + O(1)^2)^2$

Since the recursion happens until we get a 1X1 matrix resulting in next case of $O(1)$

$O(f(n)$ can be written as

$O([7 + O(1)]^n) = O(N^{\log_2 7} + O(1))$

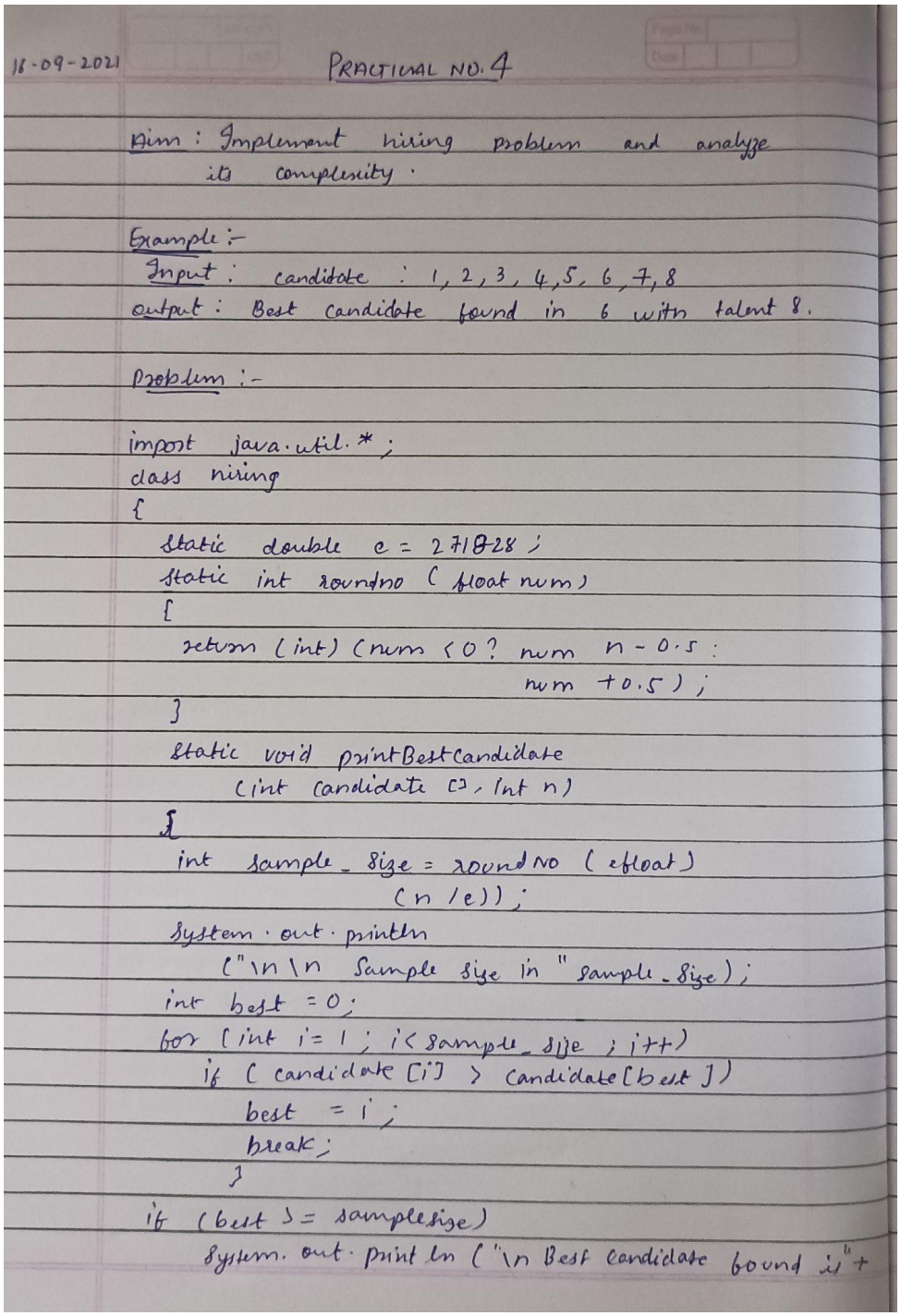
$\approx O(N \log^2 7)$

$\approx O(N^{2.8074})$

Conclusion:

We calculate the complexity of Strassen's algorithm observe that it is faster than naive matrix multiplication interms of complexity, but interm of implementation naive algorithm is easier. Thus, for small matrix sizes naive method is preffered whereas when the size increases Strassen's method is more preferable.

PRACTICAL NO: 4



18-09-2021

PRACTICAL NO. 4

Aim : Implement hiring problem and analyze its complexity.

Example :-

Input : candidate : 1, 2, 3, 4, 5, 6, 7, 8

output : Best candidate found in 6 with talent 8.

Problem :-

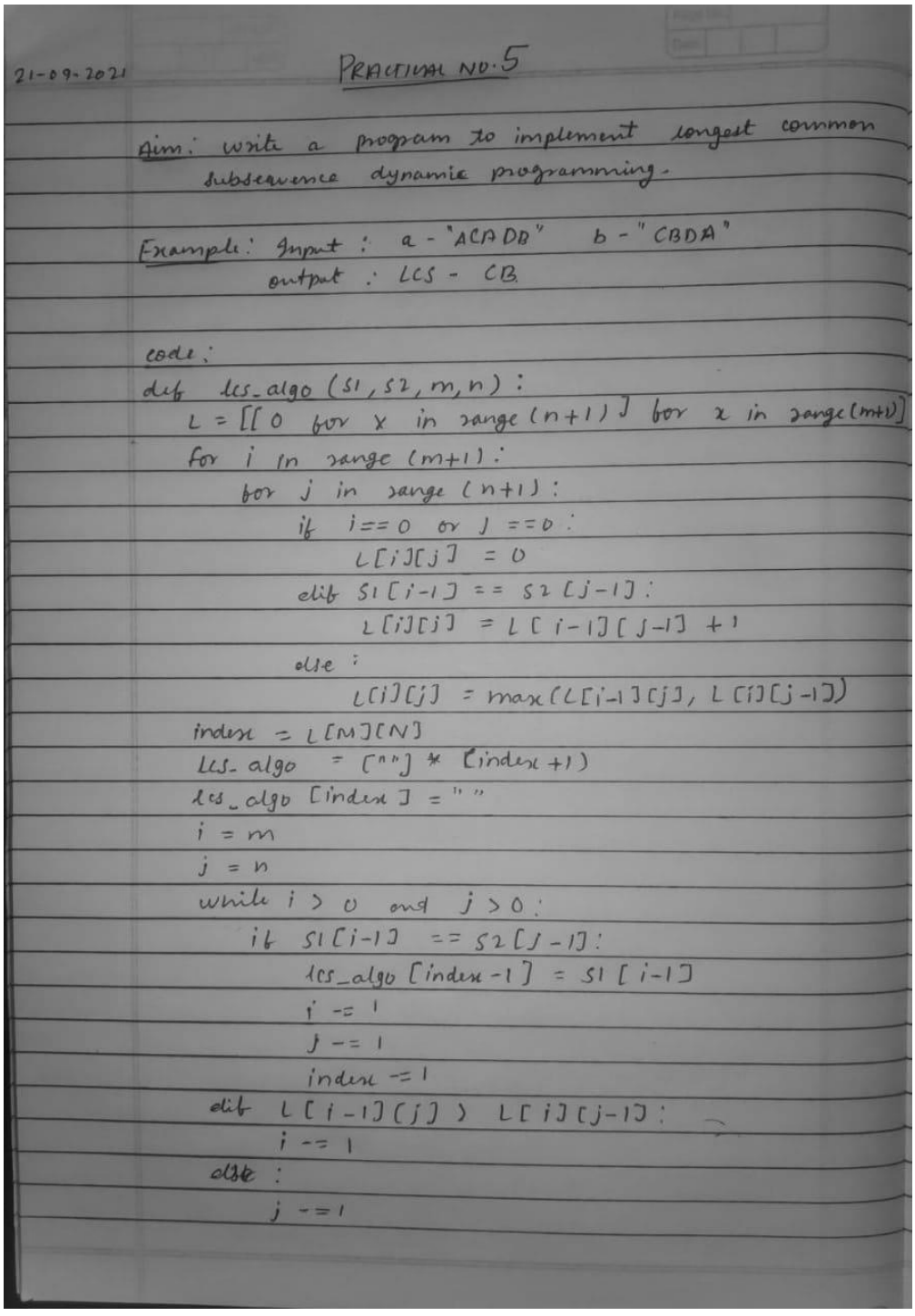
```
import java.util.*;
class hiring
{
  static double e = 2.71828;
  static int roundno (float num)
  {
    return (int) (num < 0 ? num - 0.5 :
                            num + 0.5);
  }
  static void printBestCandidate
      (int candidate [], int n)
  {
    int sample_size = roundNo (float)
                      (n / e));
    System.out.println
      ("\n\n Sample size is " sample_size);
    int best = 0;
    for (int i = 1; i < sample_size ; i++)
      if ( candidate [i] > candidate [best ])
        best = i;
        break;
      }
    if (best >= samplesize)
      System.out.println ("\n Best candidate bound is" +
```

```
        (best+1) + " with Talent" + candidate[best]);
else
    System.out.println("couldn't find a best
            candidate\n")
}
public static void main(String[] args)
{
    int n = 8;
    int[] candidate = new int[n];
    Random ra = new Random();
    for (int i=0; i<n; i++)
        candidate[i] = 1 + ra.nextInt((8-1)+1);
    System.out.print("candidate :");
    for (int i=0; i<n; i++)
        System.out.print(i+1+" ");
    System.out.println();
    System.out.print("Talents :");
    for (int i=0; i<n; i++))
        System.out.print(candidate[i]+"");
    printBestCandidate(candidate, n);
}
}
```

OUTPUT :

```
Candidate : 1 2 3 4 5 6 7 8
 Talents : 1 5 2 2 7 3 7 1

Sample size is 3

Best candidate found is 5 with talent 7


...Program finished with exit code 0
Press ENTER to exit console.
```

PRACTICAL NO: 5



21-09-2021

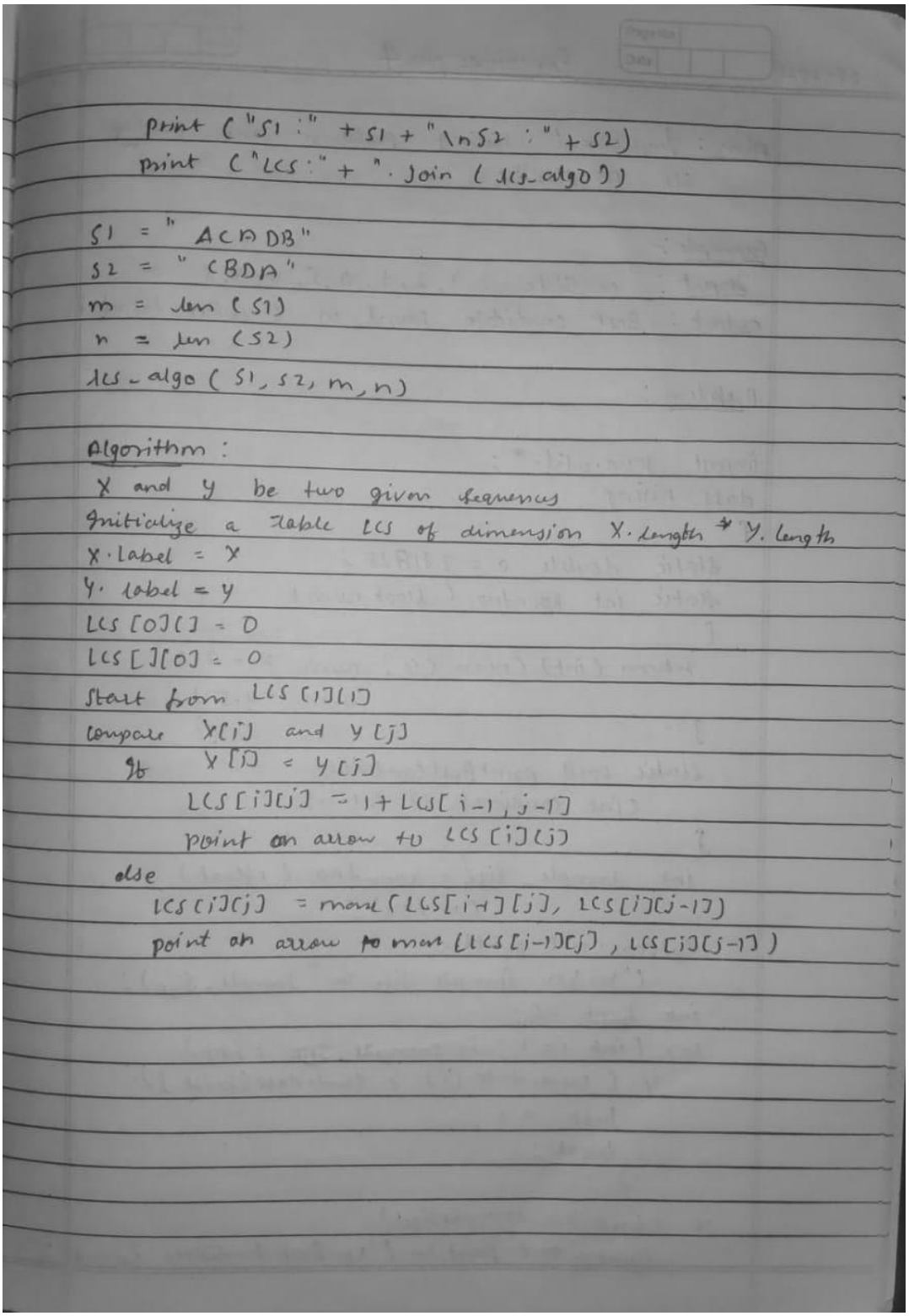
PRACTICAL NO. 5

Aim: write a program to implement longest common subsequence dynamic programming.

Example: Input : a - "ACADB" b - "CBDA"
output : LCS - CB

code:

```
def lcs_algo (S1, S2, m, n) :
    L = [[0 for x in range (n+1)] for x in range(m+1)]
    for i in range (m+1) :
        for j in range (n+1) :
            if i==0 or j ==0 :
                L[i][j] = 0
            elif S1[i-1] == S2[j-1] :
                L[i][j] = L[i-1][j-1] +1
            else :
                L[i][j] = max(L[i-1][j], L[i][j-1])
    index = L[M][N]
    lcs_algo = [""] * (index+1)
    lcs_algo [index] = ""
    i = m
    j = n
    while i > 0 and j > 0 :
        if S1[i-1] == S2[j-1] :
            lcs_algo [index-1] = S1[i-1]
            i -= 1
            j -= 1
            index -= 1
        elif L[i-1][j] > L[i][j-1] :
            i -= 1
        else :
            j -= 1
```



```
print ("S1 :" + S1 + "\nS2 :" + S2)
print ("LCS :" + "".join (lcs_algo))

S1 = "ACADB"
S2 = "CBDA"
m = len (S1)
n = len (S2)
lcs_algo (S1, S2, m, n)
```

Algorithm :

X and Y be two given sequences
Initialize a table LCS of dimension X.Length * Y.Length
X.label = X
Y.label = Y
LCS [0][] = 0
LCS [][0] = 0
Start from LCS [1][1]
Compare X[i] and Y[j]
If X[i] = Y[j]
    LCS[i][j] = 1 + LCS[i-1, j-1]
    Point an arrow to LCS[i][j]
else
    LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
    point an arrow to max(LCS[i-1][j], LCS[i][j-1])

Practical 5 - LCS.py - C:/Users/shakt/Documents/MSc-Part 1/DAA Practicals/Practical 5 - LCS...

File Edit Format Run Options Window Help

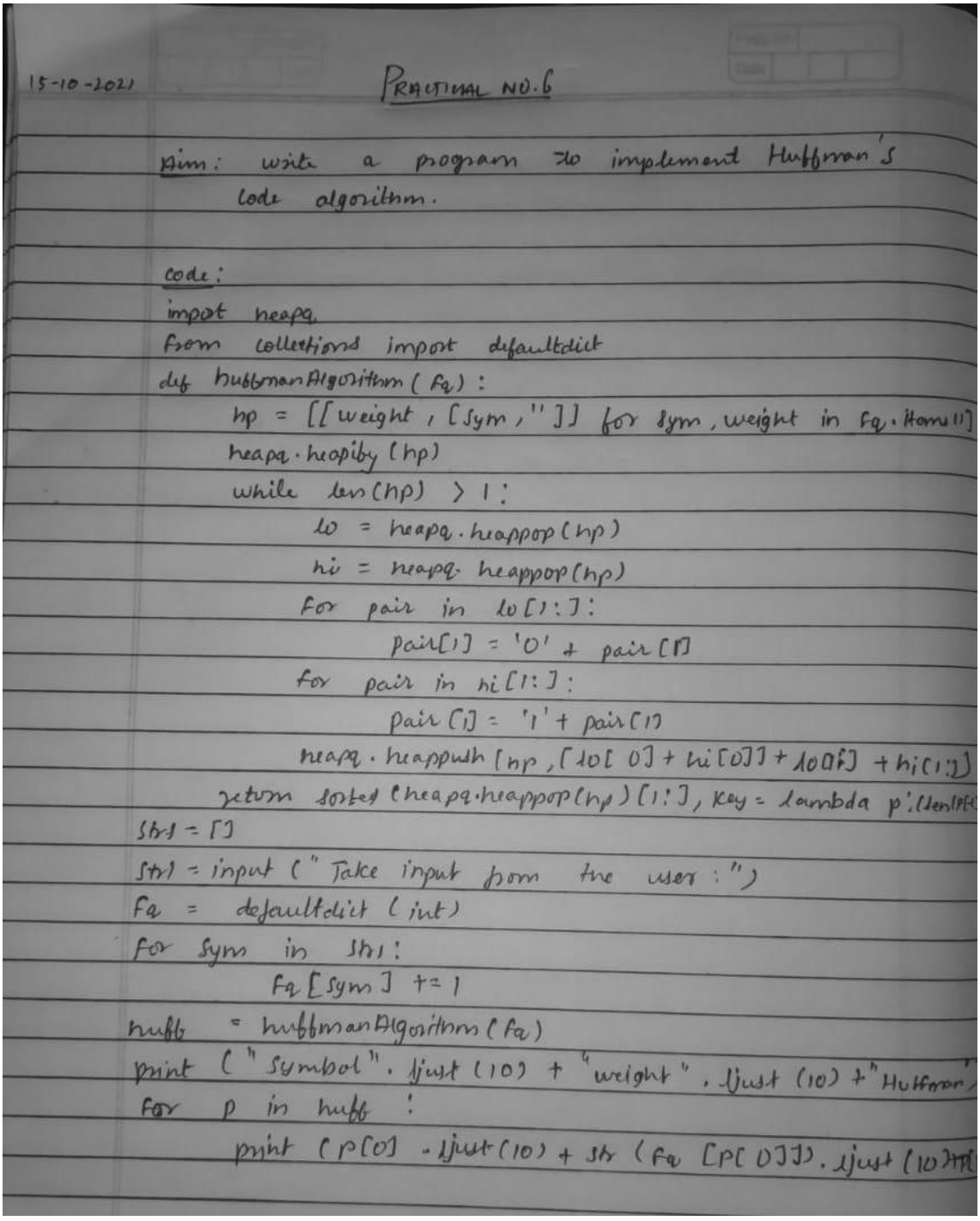
```
def lcs_algo(S1, S2, m, n):
    L = [[0 for x in range(n+1)] for x in range(m+1)]
    for i in range(m+1):
        for j in range(n+1):
            if i == 0 or j == 0:
                L[i][j] = 0
            elif S1[i-1] == S2[j-1]:
                L[i][j] = L[i-1][j-1] + 1
            else:
                L[i][j] = max(L[i-1][j], L[i][j-1])

    index = L[m][n]

    lcs_algo = [""] * (index+1)
    lcs_algo[index] = ""

    i = m
    j = n
    while i > 0 and j > 0:

        if S1[i-1] == S2[j-1]:
            lcs_algo[index-1] = S1[i-1]
            i -= 1
            j -= 1
            index -= 1

        elif L[i-1][j] > L[i][j-1]:
            i -= 1
        else:
            j -= 1

    print("S1 : " + S1 + "\nS2 : " + S2)
    print("LCS: " + "".join(lcs_algo))

S1 = "ACADB"
S2 = "CBDA"
m = len(S1)
n = len(S2)
lcs_algo(S1, S2, m, n)
```

Ln: 29 Col: 0

OUTPUT :

```
= RESTART: C:/Users/shakt/Documents/MSc-Part 1/DAA Practicals/Practical 5 - LCS.
py
S1 : ACADB
S2 : CBDA
LCS: CB
>>>
```

PRACTICAL NO: 6



15-10-2021

PRACTICAL NO.6

Aim: write a program to implement Huffman's code algorithm.

code:

```
import heapq
from collections import defaultdict
def huffmanAlgorithm(fq):
    hp = [[weight, [sym, ""]] for sym, weight in fq.items()]
    heapq.heapify(hp)
    while len(hp) > 1:
        lo = heapq.heappop(hp)
        hi = heapq.heappop(hp)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(hp, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    return sorted(heapq.heappop(hp)[1:], key = lambda p:(len(p)
str1 = []
str1 = input("Take input from the user:")
fq = defaultdict(int)
for sym in str1:
    fq[sym] += 1
huff = huffmanAlgorithm(fq)
print("Symbol".ljust(10) + "weight".ljust(10) + "Huffman
for p in huff:
    print(p[0].ljust(10) + str(fq[p[0]]).ljust(10)
```

Practical 6 - Huffman.py

```python
import heapq
from collections import defaultdict

def huffmanAlgorithm(fq):
    hp = [[weight, [sym, '']] for sym, weight in fq.items()]
    heapq.heapify(hp)
    while len(hp) > 1:
        lo = heapq.heappop(hp)
        hi = heapq.heappop(hp)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heapq.heappush(hp, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    return sorted(heapq.heappop(hp)[1:], key=lambda p: (len(p[-1]), p))

str1=[]
str1=input("Take input from the User: ")
fq = defaultdict(int)
for sym in str1:
    fq[sym] += 1

huff = huffmanAlgorithm(fq)
print ("Symbol".ljust(10) + "Weight".ljust(10) + "Huffman Code")
for p in huff:
    print (p[0].ljust(10) + str(fq[p[0]]).ljust(10) + p[1])

```

OUTPUT :

```
PS C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals>
Take input from the User: Heman Shakthi
Symbol    Weight    Huffman Code
a         2         101
h         2         110
m         1         000
n         1         001
t         1         010
          1         0110
H         1         0111
S         1         1000
e         1         1001
i         1         1110
k         1         1111
```

PRACTICAL NO: 7

20-10-2021

PRACTICAL NO.7.

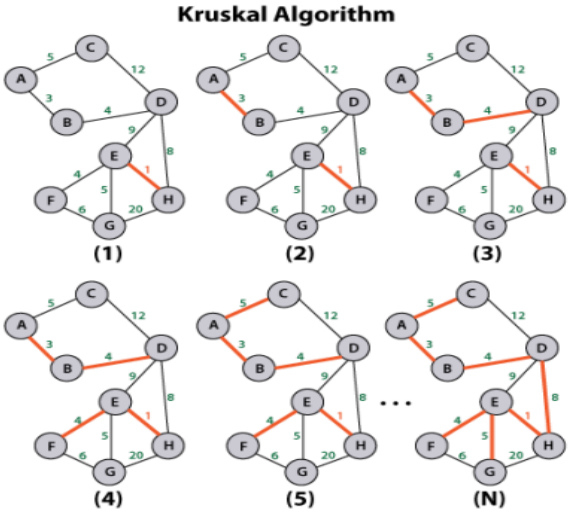
Aim: write a program to implement kruskal's Algorithm.

Algorithm:

- create the edge list of given graph, with their weights.
- sort the edge list according to their weights in ascending order.
- Draw all the nodes to create skeleton for spanning tree.
- Pick up the edges at the top of the edge list.
- remove the edge from the edge list
- connect the vertices with given edge. If by connecting the vertices, a cycle is created then discard this edge
- Repeat the following until n-1 edges are added or list of edges are over.

Time complexity:

Time complexity for kruskal's Algorithm is O(E log V)



Practical 7 - Kruskal.py

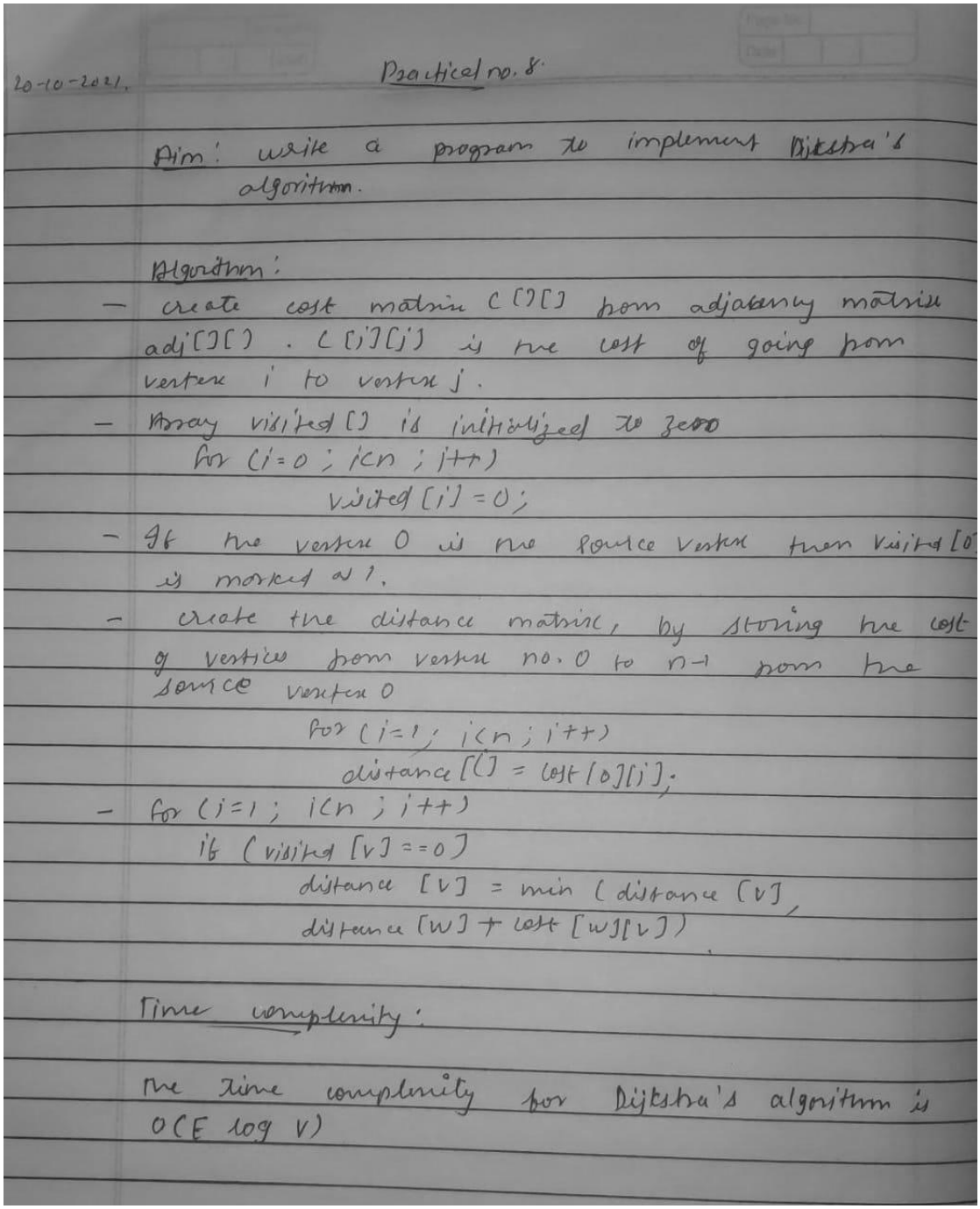
```python
from collections import defaultdict

class Graph:

    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)

        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot

        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    def KruskalMST(self):
        result = []
        i = 0
        e = 0
        self.graph = sorted(self.graph,
                            key=lambda item: item[2])
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
```

```
            rank.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)
        minimumCost = 0
        print ("Edges in the constructed MST")
        for u, v, weight in result:
            minimumCost += weight
            print("%d -- %d == %d" % (u, v, weight))
        print("Minimum Spanning Tree" , minimumCost)

g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

g.KruskalMST()
```

OUTPUT :

```
Edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Spanning Tree 19
```

PRACTICAL NO: 8



20-10-2021.

Practical no. 8.

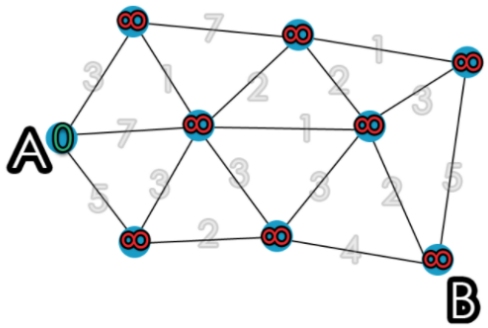
Aim: write a program to implement Dijkstra's algorithm.

Algorithm:

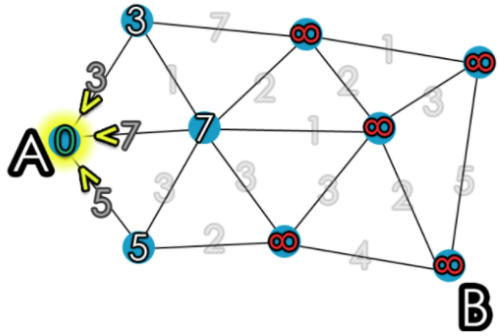
- create cost matrix C[][] from adjacency matrix adj[][]. C[i][j] is the cost of going from vertex i to vertex j.
- Array visited[] is initialized to zero
  for (i=0; i<n; i++)
  visited[i]=0;
- If the vertex 0 is the source vertex then visited[0] is marked as 1.
- create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0
  for (i=1; i<n; i++)
  distance[i] = cost[0][i];
- for (i=1; i<n; i++)
  if (visited[v]==0)
  distance[v] = min(distance[v], distance[w] + cost[w][v])

Time complexity:

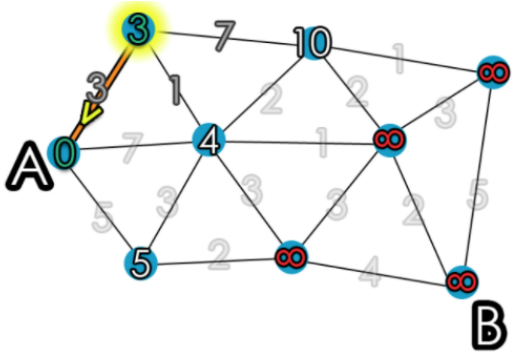
The time complexity for Dijkstra's algorithm is O(E log V)



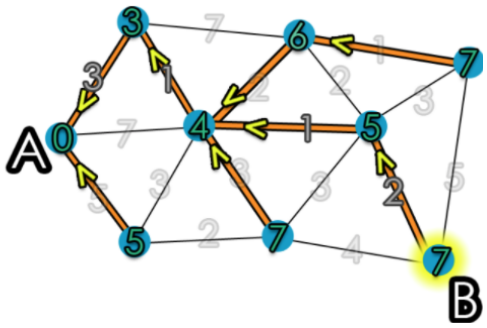
Initialize distances according to the algorithm.



Pick the first node and calculate distances to adjacent nodes.



Pick next node with minimal distance; repeat adjacent node distance calculations.



Final result of shortest-path tree

Practical 8 - Djsktra.py > ...

```python
import sys
vertices = [[0, 0, 1, 1, 0, 0, 0],
            [0, 0, 1, 0, 0, 1, 0],
            [1, 1, 0, 1, 1, 0, 0],
            [1, 0, 1, 0, 0, 0, 1],
            [0, 0, 1, 0, 0, 1, 0],
            [0, 1, 0, 0, 1, 0, 1],
            [0, 0, 0, 1, 0, 1, 0]]
edges = [[0, 0, 1, 2, 0, 0, 0],
         [0, 0, 2, 0, 0, 3, 0],
         [1, 2, 0, 1, 3, 0, 0],
         [2, 0, 1, 0, 0, 0, 1],
         [0, 0, 3, 0, 0, 2, 0],
         [0, 3, 0, 0, 2, 0, 1],
         [0, 0, 0, 1, 0, 1, 0]]
# Find which vertex is to be visited next
def to_be_visited():
    global visited_and_distance
    v = -10
    for index in range(num_of_vertices):
        if visited_and_distance[index][0] == 0 \
            and (v < 0 or visited_and_distance[index][1] <=
                 visited_and_distance[v][1]):
            v = index
    return v
num_of_vertices = len(vertices[0])
```

```
27
28  visited_and_distance = [[0, 0]]
29  for i in range(num_of_vertices-1):
30      visited_and_distance.append([0, sys.maxsize])
31
32  for vertex in range(num_of_vertices):
33      to_visit = to_be_visited()
34      for neighbor_index in range(num_of_vertices):
35
36          if vertices[to_visit][neighbor_index] == 1 and \
37                  visited_and_distance[neighbor_index][0] == 0:
38              new_distance = visited_and_distance[to_visit][1] \
39                  + edges[to_visit][neighbor_index]
40              if visited_and_distance[neighbor_index][1] > new_distance:
41                  visited_and_distance[neighbor_index][1] = new_distance
42
43          visited_and_distance[to_visit][0] = 1
44  i = 0
45  for distance in visited_and_distance:
46      print("Distance of ", chr(ord('a') + i),
47            " from source vertex: ", distance[1])
48      i = i + 1
```

OUTPUT :

```
Distance of  a  from source vertex:  0
Distance of  b  from source vertex:  3
Distance of  c  from source vertex:  1
Distance of  d  from source vertex:  2
Distance of  e  from source vertex:  4
Distance of  f  from source vertex:  4
Distance of  g  from source vertex:  3
```

PRACTICAL NO: 9

16-10-2021

PRACTICAL NO. 9.

Aim: write a program to implement multi threaded computation concepts in the generation of Fibonacci numbers.

code:

```
import java.io.*;
class Fibonacci extends thread
    public static void main (String [] args)
    {
      try {
          int a=0, b=1, c=0;
          BufferedReader br = new BufferedReader (new
                InputStreamReader (System.in));
          System.out.print ("Enter the limit for fibonacci:");
          int n = Integer.parseInt (br.readline());
          System.out.println ("Fibonacci series");
          while (n>0)
          {
            System.out.print (c + " ");
            a = b;
            b = c;
            c = a + b;
            n = n-1;
          }
      }
      catch (Exception ex)
      { ex.printStackTrace();
      }
    }
}
```

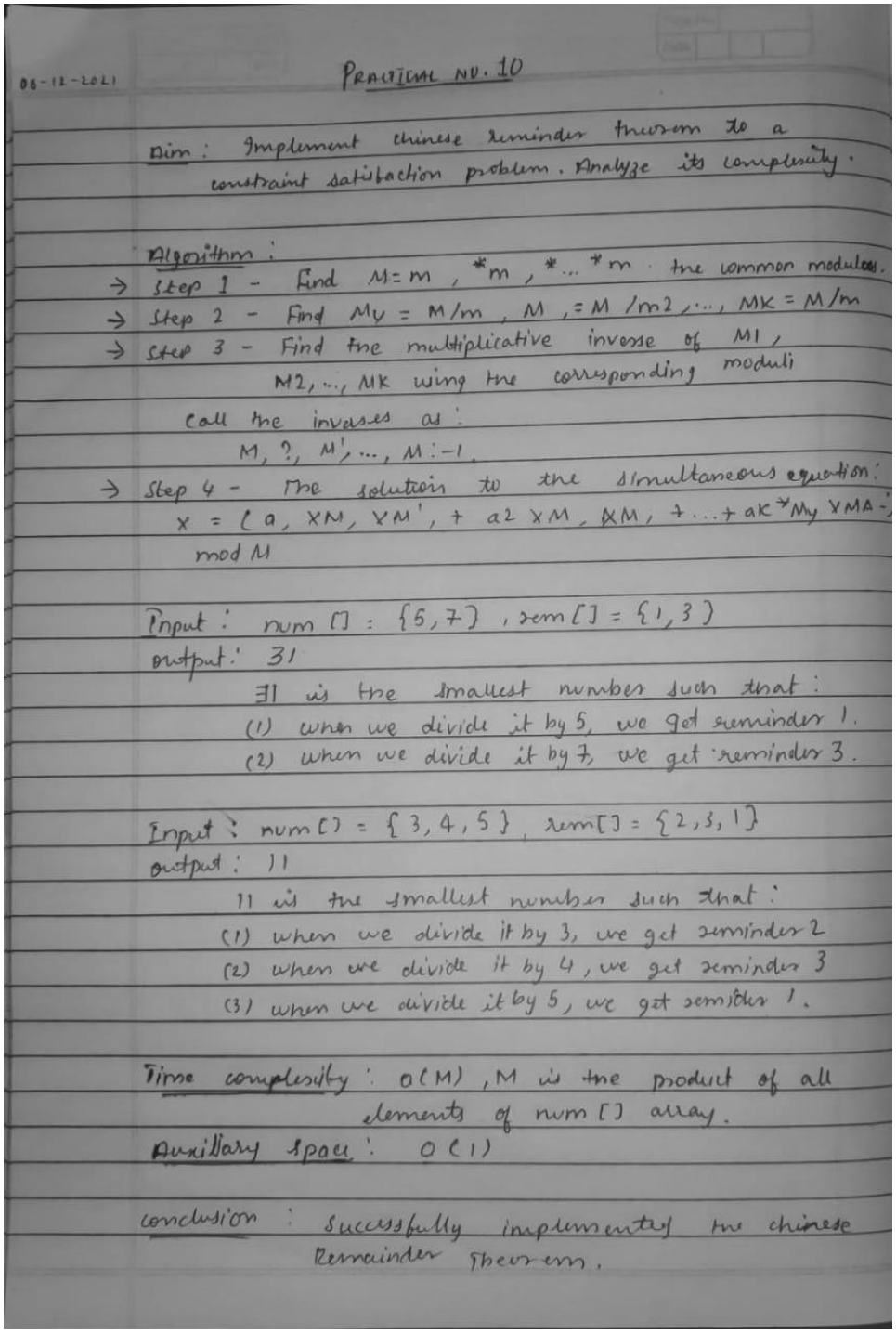
FibonacciThread.java > ...

```java
import java.io.*;
class Fibonacci extends Thread
{
    Run | Debug
    public static void main(String[] args)
    {
        try
        {
            int a=0, b=1, c=0;
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

            System.out.print("Enter the Limit for fabonacci: ");

            int n = Integer.parseInt(br.readLine());
            System.out.println("\n================================");
            System.out.println("Fibonacci series:");
            while (n>0)
            {
                System.out.print(c+" ");
                a=b;
                b=c;
                c=a+b;
                n=n-1;
            }
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

OUTPUT :

```
PS C:\Users\shakt\Documents\MSc-Part 1\DAA Practicals>
e.encoding=UTF-8' '-cp' 'C:\Users\shakt\AppData\Roaming\
Enter the Limit for fabonacci: 12

================================
Fibonacci series:
0 1 1 2 3 5 8 13 21 34 55 89
```

PRACTICAL NO: 10



06-12-2021

PRACTICAL NO. 10

Aim: Implement chinese reminder theorem to a constraint satisfaction problem. Analyze its complexity.

Algorithm:

→ Step 1 - Find M = m , *m , *... *m the common modulus.

→ Step 2 - Find My = M/m , M , = M /m2 ,..., MK = M/m

→ Step 3 - Find the multiplicative inverse of M1, M2, .., MK wing the corresponding moduli

call the inverses as:

M, ?, M', ..., M:-1.

→ Step 4 - The solution to the simultaneous equation:

x = (a, XM, YM', + a2 XM, XM, + ... + aK *My YMA-1 mod M

Input: num [] : {5, 7} , rem [] = {1, 3}

output: 31

31 is the smallest number such that:

(1) when we divide it by 5, we get reminder 1.

(2) when we divide it by 7, we get reminder 3.

Input: num [] = {3, 4, 5}, rem[] = {2, 3, 1}

output: 11

11 is the smallest number such that:

(1) when we divide it by 3, we get reminder 2

(2) when we divide it by 4, we get reminder 3

(3) when we divide it by 5, we get reminder 1.

Time complexity: O(M) , M is the product of all elements of num [] array.

Auxiliary space: O(1)

conclusion: Successfully implementing the chinese Remainder Theorem.

```
def findMinX(num, rem, k):
    x = 1;
    while(True):
        j = 0;
        while(j < k):
            if (x % num[j] != rem[j]):
                break;
            j += 1;
        if (j == k):
            return x;
    x += 1;
num = [3, 4, 5];
rem = [2, 3, 1];
k = len(num);
print("x is", findMinX(num, rem, k));
```

Output : X is 11

Time Complexity : O(M) , M is the product of all elements of num[] array

Auxiliary Space : O(1)

Conclusion : Successfully implemented the Chinese Remainder Theorem