Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East - 400098

A Practical Journal Submitted in fulfilment

of the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

YEAR 2022-2023

Part 1 - Semester 1

PSCSP101

"Analysis of Algorithms & Research Computing"

By

Ms. VISHWAKARMA ASHWANI HARILAL

Application ID:- 104620

Seat No:- 510177

Under the Guidance of

Prof. Kavita chowk

Institute of Distance and Open Learning

Vidya Nagari, kalina, Santacruz East – 400098

CERTIFICATE

This is to certify that, this practical journal entitled **"Analysis of Algorithms & Research Computing"** is a record of work carried out by **Ms. Vishwakarma Ashwani Harilal (Seat No:-510177)**, student of **Master of Science in Computer Science Part 1** class and is submitted to University of Mumbai, in partial fulfillment of the requirement for the award of the degree of **Master of Science in Computer Science.** The practical journal has been approved.

----------	----------	----------
Guide	External Examiner	Coordinator – M.Sc.CS

INDEX

Sr.No	Date	Practicals	Signature
1		Write a program to implement insertion sort and find the running time of the Algorithms.	
2		Write a program to implement a merge sort algorithm. Compare the time and memory complexity.	
3		Given an array of numbers of length l. Write a program to generate a random permutation of the array using (i) permute-by-sorting() and (ii) permute-by-cyclic().	
4		Write a program to implement the Longest Common Subsequence (LCS) algorithm.	
5		Write a program to implement Kruskal's algorithm.	
6		Write a program to implement Dijkstra's algorithm.	
7		Write a program to implement Euclid's algorithm to implement gcd of two non-negative integers a and b. Extend the algorithm to find x and y such that gcd(a,b) = ax+by. Compare the running time and recursive calls made in each case.	
8		Write a program to verify (i) Euclid's theorem (ii) Fermat's theorem.	

PRACTICAL - 01

Q). Write a program to implement insertion sort and find the running time of the Algorithms.

INPUT :-

```
def insertionSort(array):

   for step in range(1, len(array)):
       key = array[step]
       j = step - 1

       # Compare key with each element on the left of it until an
element smaller than it is found
       # For descending order, change key<array[j] to key>array[j].
       while j >= 0 and key < array[j]:
           array[j + 1] = array[j]
           j = j - 1

       # Place the key after the element just smaller than it.
       array[j + 1] = key


data = [9, 5, 1, 4, 3]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```

OUTPUT:-

```
Sorted Array in Ascending Order:
[1, 3, 4, 5, 9]

[Done] exited with code=0 in 0.032 seconds
```

PRACTICAL - 02

Q). Write a program to implement a merge sort algorithm. Compare the time and memory complexity.

INPUT :-

```
def mergeSort(arr):
       if len(arr) > 1:
           # Create sub_array2 ← A[start..mid] and sub_array2 ←
A[mid+1..end]
           mid = len(arr)//2
           sub_array1 = arr[:mid]
           sub_array2 = arr[mid:]
           # Sort the two halves
           mergeSort(sub_array1)
           mergeSort(sub_array2)

           # Initial values for pointers that we use to keep track of
where we are in each array
           i = j = k = 0
       # Until we reach the end of either start or end, pick larger
among
       # elements start and end and place them in the correct position
in the sorted array
           while i < len(sub_array1) and j < len(sub_array2):
               if sub_array1[i] < sub_array2[j]:
                   arr[k] = sub_array1[i]
                   i += 1
               else:
                   arr[k] = sub_array2[j]
                   j += 1
               k += 1
       # When all elements are traversed in either arr1 or arr2,
       # pick up the remaining elements and put in sorted array
           while i < len(sub_array1):
               arr[k] = sub_array1[i]
               i += 1
               k += 1
           while j < len(sub_array2):
               arr[k] = sub_array2[j]
               j += 1
               k += 1
```

```
arr = [10, 9, 2, 4, 6, 13]
mergeSort(arr)
print("Insertion sort")
print(arr)
```

OUTPUT : -

```
Insertion sort
[2, 4, 6, 9, 10, 13]

[Done] exited with code=0 in 0.064 seconds
```

PRACTICAL - 03

Q). Given an array of numbers of length l. Write a program to generate a random permutation of the array using (i) permute-by-sorting() and(ii) permute-by-cyclic().

INPUT : -

```
from itertools import permutations
list1=list(permutations(range(0,2)))
print(list1)
import itertools
cnt=0
x=itertools.cycle([1,2])
for i in x:
  print(i)
cnt=cnt+1
if cnt>10:
  break
```

OUTPUT : -

```
[(0, 1), (1, 0)]
1
2
1
2
1
2
1
2
1
2
1

[Done] exited with code=0 in 0.05 seconds
```

PRACTICAL - 04

Q). Write a program to implement the Longest Common Subsequence (LCS) algorithm.

INPUT : -

```
def lcs(X, Y, m, n):


  if m == 0 or n == 0:

     return 0
  elif X[m-1] == Y[n-1]:

     return 1 + lcs(X, Y, m-1, n-1)
  else:

     return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n))


# Driver program to test the above function
X = "AGGTAB"
Y = "GXTXAYB"
print("Length of LCS is ", lcs(X, Y, len(X), len(Y)))
```

OUTPUT : -

```
Length of LCS is  4

[Done] exited with code=0 in 0.024 seconds
```

PRACTICAL - 05

Q). Write a program to implement Kruskal's algorithm.

INPUT : -

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

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

  # Search function

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

  def apply_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

  #  Applying Kruskal algorithm
  def kruskal_algo(self):
      result = []
      i, e = 0, 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)
```

```
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.apply_union(parent, rank, x, y)
      for u, v, weight in result:
          print("%d - %d: %d" % (u, v, weight))


g = Graph(6)
g.add_edge(0, 1, 4)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 2)
g.add_edge(1, 0, 4)
g.add_edge(2, 0, 4)
g.add_edge(2, 1, 2)
g.add_edge(2, 3, 3)
g.add_edge(2, 5, 2)
g.add_edge(2, 4, 4)
g.add_edge(3, 2, 3)
g.add_edge(3, 4, 3)
g.add_edge(4, 2, 4)
g.add_edge(4, 3, 3)
g.add_edge(5, 2, 2)
g.add_edge(5, 4, 3)
g.kruskal_algo()
```

OUTPUT : -

```
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4
[Done] exited with code=0 in 0.029 seconds
```

PRACTICAL - 06

Q). Write a program to implement Dijkstrass's algorithm.

INPUT : -

```
import sys


class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \tDistance from Source")
        for node in range(self.V):
            print(node, "\t", dist[node])

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minDistance(self, dist, sptSet):

        # Initialise minimum distance for next node
        min = sys.maxsize

        # Search not nearest vertex not in the
        # shortest path tree
        for u in range(self.V):
            if dist[u] < min and sptSet[u] == False:
                min = dist[u]
                min_index = u

        return min_index

    # Function that implements Dijkstra's single source
    # shortest path algorithm for a graph represented
    # using adjacency matrix representation
    def dijkstra(self, src):
```

```
        dist = [sys.maxsize] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):

            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # x is always equal to src in first iteration
            x = self.minDistance(dist, sptSet)

            # Put the minimum distance vertex in the
            # shortest path tree
            sptSet[x] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shortest path tree
            for y in range(self.V):
                if self.graph[x][y] > 0 and sptSet[y] == False and \
                        dist[y] > dist[x] + self.graph[x][y]:
                    dist[y] = dist[x] + self.graph[x][y]

        self.printSolution(dist)


if__name__== "__main__":
    g = Graph(9)
    g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
               [4, 0, 8, 0, 0, 0, 0, 11, 0],
               [0, 8, 0, 7, 0, 4, 0, 0, 2],
               [0, 0, 7, 0, 9, 14, 0, 0, 0],
               [0, 0, 0, 9, 0, 10, 0, 0, 0],
               [0, 0, 4, 14, 10, 0, 2, 0, 0],
               [0, 0, 0, 0, 0, 2, 0, 1, 6],
               [8, 11, 0, 0, 0, 0, 1, 0, 7],
               [0, 0, 2, 0, 0, 0, 6, 7, 0]
               ]

    g.dijkstra(0)
```

OUTPUT : -

```
Vertex  Distance from Source
0    0
1    4
2    12
3    19
4    21
5    11
6    9
7    8
8    14

[Done] e ited with code=0 in 0.038 seconds
```

PRACTICAL - 07

Q). Write a program to implement Euclid's algorithm to implement gcd of two non negative integers a and b. Extend the algorithm to find x and y such that gcd(a,b) = ax+by. Compare the running time and recursive calls made in each case.

INPUT : -

```
a=int(input("enter the first number : "))
b=int(input("enter the second number : "))

i=1
if a>b:
small=a
else:
small=b
for i in range(1,small+1):
if(a%i==0) and (b%i==0):

  out=i
print("gcd of number",a,b,out)
```

OUTPUT : -

```
enter the first number 15
enter the second number 10
gcd of number 15 10 is 5
```

PRACTICAL - 08

Q). Write a program to verify (i) Euclid's theorem (ii) Fermat's theorem.

(i) Euclid's theorem

INPUT : -

```
print("euclid algo")
def egcd(a,b):
if(a == 0):
 return b
return egcd(b%a,a)
a=15
b=10
print("gcd of ",egcd(a,b))
```

OUTPUT : -

```
euclid algo
gcd of  5
```

(ii) Fermat's theorem.

INPUT : -

```
def power(x, y):
    res = 1
    while (y > 0):
        if (y & 1):
            res = res * x
        y = y >> 1
        x = x * x
    return res
def Fermat(i):
    power2_i = power(2, i)
    power2_2_i = power(2, power2_i)
    return power2_2_i + 1

def Fermat_Number(n):
     for i in range(n):
         print(Fermat(i), end = "")
         if(i != n - 1):
             print(end = ", ")
```

```
n = 4
Fermat_Number(n)
```

OUTPUT : -

```
3, 5, 17, 257
[Done] exited with code=0 in 0.047 seconds
```