SVR ENGINEERING COLLEGE

AYYALURUMETTA (V), NANDYAL, KURNOOL
DT.ANDHRA PRADESH – 518502

2021 – 2022

LABORATORY MANUAL

OF

Advanced Data Structures and Algorithms Lab

(20A05301P)

(R-20 REGULATION)

Prepared by

Dr. Rajesh Chandra
ProfessorFor

B.Tech II YEAR – I SEM. (CSE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SVR ENGINEERING COLLEGE

(AFFILIATED TO JNTUA ANANTHAPURAM- AICITE-INDIA)
AYYALURUMETTA (V), NANDYAL, KURNOOL
DT.ANDHRA PRADESH – 518502

LAB MANUAL CONTENT

Advanced Data Structures and Algorithms Lab

(20A05301P)

1. Institute Vision & Mission, Department Vision & Mission
2. PO, PEO& PSO Statements.
3. List of Experiments
4. CO-PO Attainment
5. Experiment Code and Outputs

1. Institute Vision & Mission, Department Vision & Mission

Institute Vision:

To produce Competent Engineering Graduates & Managers with a strongbase of Technical & Managerial Knowledge and the Complementary Skills needed to be Successful Professional Engineers & Managers.

Institute Mission:

To fulfill the vision by imparting Quality Technical & Management Education to the Aspiring Students, by creating Effective Teaching/Learning Environment and providing State – of the – Art Infrastructure and Resources.

Department Vision:

To produce Industry ready Software Engineers to meet the challenges of 21st Century.

Department Mission:

- Impart core knowledge and necessary skills in Computer Science and Engineering through innovative teaching and learning methodology.
- Inculcate critical thinking, ethics, lifelong learning and creativity needed for industry and society.
- Cultivate the students with all-round competencies, for career, higher education and self-employability.

2. PO, PEO& PSO Statements

PROGRAMME OUTCOMES (POs)

PO-1: Engineering knowledge - Apply the knowledge of mathematics, science, engineering fundamentals of Computer Science& Engineering to solve complex real-life engineering problems related to CSE.

PO-2: Problem analysis - Identify, formulate, review research literature, and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO-3: Design/development of solutions - Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, cultural, societal and environmental considerations.

PO-4: Conduct investigations of complex problems - Use research-based knowledge and research methods, including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

PO-5: Modern tool usage - Select/Create and apply appropriate techniques, resources andmodern engineering and IT tools and technologies for rapidly changing computing needs, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

PO-6: The engineer and society - Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the CSE professional engineering practice.

PO-7: Environment and Sustainability - Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO-8: Ethics - Apply ethical principles and commit to professional ethics and responsibilities and norms of the relevant engineering practices.

PO-9: Individual and team work - Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO-10: Communication - Communicate effectively on complex engineering activities with the engineering community and with the society-at-large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, give and receive clear instructions.

PO-11: Project management and finance - Demonstrate knowledge and understanding ofthe engineering and management principles and apply these to one's own work, as amember and leader in a team, to manage projects and in multidisciplinary environments. **PO-12: Life-long learning -** Recognize the need for and have the preparation and ability toengage in independent and life-long learning in the broadcast context of technologicalchanges.

Program Educational Objectives (PEOs):

PEO 1:Graduates will be prepared for analyzing, designing, developing and testing the software solutions and products with creativity and sustainability.

PEO 2: Graduates will be skilled in the use of modern tools for critical problem solvingandanalyzing industrial and societal requirements.

PEO 3:Graduates will be prepared with managerial and leadership skills for career andstarting up own firms.

Program Specific Outcomes (PSOs):

PSO 1: Develop creative solutions by adapting emerging technologies / tools for real timeapplications.

PSO 2: Apply the acquired knowledge to develop software solutions and innovative mobileapps for various automation applications

2.1 Subject Time Table

SVR ENGINEERING COLLEGE::NANDYAL DEPARTMENT OF CSE								
Dr. Rajesh Chandra						II-I		
Day/ Time	9:30 AM	10:20 AM	11:30 AM	12:20 PM-	LUNCH BREAK	02:00 PM	02:50 PM	03:40 PM
	10:20 AM	11:10AM	12:20 PM	01:10 PM		02:50 PM	03:40 PM	04:30 PM
MON								
TUE								
WED						ADSA		
THU								
FRI								
SAT								

LIST OF EXPERIMENTS (SYLLABUS)
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

B.Tech – II-I Sem

(20A05301P) Advanced Data Structures and Algorithms Lab

Course Objectives:

- Learn data structures for various applications.
- Implement different operations of data structures by optimizing the performance.
- Develop applications using Greedy, Divide and Conquer, dynamic programming.
- Implement applications for backtracking algorithms using relevant data structures.

Course Outcomes(CO):

- Understand and apply data structure operations.
- Understand and apply non-linear data structure operations.
- Apply Greedy, divide and conquer algorithms.
- Develop dynamic programming algorithms for various real-time applications.
- Illustrate and apply backtracking algorithms, further able to understand non-deterministic algorithms.

List of Experiments:

1. Write a program to implement the following operations on Binary Search Tree:
   a) Insert b) Delete c) Search d) Display
2. Write a program to perform a Binary Search for a given set of integer values.
3. Write a program to implement Splay trees.
4. Write a program to implement Merge sort for the given list of integer values.
5. Write a program to implement Quicksort for the given list of integer values.
6. Write a program to find the solution for the knapsack problem using the greedy method.
7. Write a program to find minimum cost spanning tree using Prim's algorithm
8. Write a program to find minimum cost spanning tree using Kruskal's algorithm
9. Write a program to find a single source shortest path for a given graph.
10. Write a program to find the solution for job sequencing with deadlines problems.

11. Write a program to find the solution for a 0-1 knapsack problem using dynamic programming.
12. Write a program to solve Sum of subsets problem for a given set of distinct numbers using backtracking.
13. ImplementNQueen'sproblemusingBackTrackin

1. **Write a program to implement the following operations on Binary Search Tree:**
   a) Insert b) Delete c) Search d) Display

```
# Binary Search Tree operations in Python
# Create a node
class Node:
        def__init_(self, key):
                self.key = key
                self.left = None
                self.right = None
                # Inorder traversal
        def  inorder(root):
                If root is not None:
                        # Traverse left
                        inorder(root.left)
                        # Traverse root
                        print(str(root.key) + "->", end=' ')
                        # Traverse right
                        inorder(root.right)
        # Insert a node
        def  insert(node, key):
                # Return a new node if the tree is empty
                if node is None:
                        return Node(key)
                # Traverse to the right place and insert the node
                if key < node.key:
                        node.left = insert(node.left, key)
                else:
                        node.right = insert(node.right, key)
                return node
        # Find the inorder successor
        def  minValueNode(node):
                current = node
                # Find the leftmost leaf
                while(current.left is not None):
                        current = current.left
                return current
        # Deleting a node
        def deleteNode(root, key):
                # Return if the tree is empty
                if root is None:


                                return
                                root# Find
        the node to be deleted
         if key < root.key:
```

```
            root.left = deleteNode(root.left, key)
        elif(key > root.key):
            root.right = deleteNode(root.right, key)
        else:
            # If the node is 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
            # If the node has two children,
            # place the inorder successor in position of the node to be deleted
            temp = minValueNode(root.right)
            root.key = temp.key
            # Delete the inorder successor
            root.right = deleteNode(root.right, temp.key)
        return root
root = None
root = insert(root, 8)
root = insert(root, 3)
root = insert(root, 1)
root = insert(root, 6)
root = insert(root, 7)
root = insert(root, 10)
root = insert(root, 14)
root = insert(root, 4)
print("Inorder traversal: ", end=' ')
inorder(root)
print("\nDelete 10")
root = deleteNode(root, 10)
print("Inorder traversal: ", end=' ')
inorder(root)
```

OUTPUT

```
Inorder traversal:  1-> 3-> 4-> 6-> 7-> 8-> 10-> 14->
Delete 10
Inorder traversal:  1-> 3-> 4-> 6-> 7-> 8-> 14->

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

2. Write a program to perform a Binary Search for a given set of integer values.

```
# Binary Search in python
def binarySearch(array, x, low, high):
        # Repeat until the pointers low and high meet each other
        while low <= high:
                mid = low + (high - low)//2
                if array[mid] == x:
                        return mid
                elif array[mid] < x:
                        low = mid + 1
                else:
                        high = mid - 1
        return -1
array = [3, 4, 5, 6, 7, 8, 9]
x = 4
result = binarySearch(array, x, 0, len(array)-1)
if result != -1:
        print("Element is present at index " + str(result))
else:
        print("Not found")
```

OUTPUT

```
Element is present at index 1


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

3. Write a program to implement Splay trees.

```
class Node:
        def__init_(self, data):
                self.data = data
                self.parent = None
                self.left = None
                self.right = None
class SplayTree:
         def__init_(self):
                 self.root = None
def maximum(self, x):
        while x.right != None:
                 x = x.right
                 return x
def left_rotate(self, x):
        y = x.right
        x.right = y.left
        if y.left != None:
                y.left.parent = x
                y.parent = x.parent
        if x.parent == None: #x is root
                self.root = y
        elif x == x.parent.left: #x is left child
                 x.parent.left = y
        else: #x is right child
                 x.parent.right = y
                y.left = x
                x.parent = y
def right_rotate(self, x):
        y = x.left
        x.left = y.right
         if y.right != None:
                y.right.parent = x
                y.parent = x.parent
        if x.parent == None: #x is root
                self.root = y
        elif x == x.parent.right: #x is right child
                x.parent.right = y
        else: #x is left child
                x.parent.left = y
                y.right = x
                x.parent = y
def splay(self, n):
        while n.parent != None: #node is not root
                if n.parent == self.root: #node is child of root, one rotation
```

```
                if n == n.parent.left:
                        self.right_rotate(n.parent)
                else:
                        self.left_rotate(n.parent)
                else:
                        p = n.parent
                        g = p.parent #grandparent
                if n.parent.left == n and p.parent.left == p: #both are left children
                        self.right_rotate(g)
                        self.right_rotate(p)
                elif n.parent.right == n and p.parent.right == p: #both are right children
                        self.left_rotate(g)
                        self.left_rotate(p)
                elif n.parent.right == n and p.parent.left == p:
                        self.left_rotate(p)
                        self.right_rotate(g)
                elif n.parent.left == n and p.parent.right == p:
                        self.right_rotate(p)
                        self.left_rotate(g)
def insert(self, n):
        y = None
        temp = self.root
        while temp != None:
                y = temp
                if n.data < temp.data:
                        temp = temp.left
                else:
                        temp = temp.right
                        n.parent = y
                if y == None: #newly added node is root
                        self.root = n
                elif n.data < y.data:
                        y.left = n
                else:
                         y.right = n
                        self.splay(n)
def search(self, n, x):
         if x == n.data:
                self.splay(n)
                return n
        elif x < n.data:
                return self.search(n.left, x)
        elif x > n.data:
                return self.search(n.right, x)
        else:
                return None
```

```
def delete(self, n):
        self.splay(n)
        left_subtree = SplayTree()
        left_subtree.root = self.root.left
        if left_subtree.root != None:
                left_subtree.root.parent = None
                right_subtree = SplayTree()
                right_subtree.root = self.root.right
        if right_subtree.root != None:
                right_subtree.root.parent = None
        if left_subtree.root != None:
                m = left_subtree.maximum(left_subtree.root)
                left_subtree.splay(m)
                left_subtree.root.right = right_subtree.root
                self.root = left_subtree.root
        else:
                self.root = right_subtree.root
def inorder(self, n):
        if n != None:
                self.inorder(n.left)
                print(n.data)
                self.inorder(n.right)
        if __name_   == '__main__':
                 t = SplayTree()
                 a = Node(10)
                 b = Node(20)
                 c = Node(30)
                 d = Node(100)
                 e =  Node(90)
                 f =  Node(40)
                 g = Node(50)
                 h =  Node(60)
                 i = Node(70)
                 j =   Node(80)
                 k = Node(150)
                 l = Node(110)
                 m = Node(120)
                 t.insert(a)
                 t.insert(b)
                 t.insert(c)
                 t.insert(d)
                 t.insert(e)
                 t.insert(f)
                 t.insert(g)
                 t.insert(h)
                 t.insert(i)
```

```
t.insert(j)
t.insert(k)
t.insert(l)
t.insert(m)
t.delete(a)
t.delete(m)
t.inorder(t.root)
```

4. Write a program to implement Merge sort for the given list of integer values.

```
# Python program for implementation of MergeSort
# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]

def merge(arr, l, m, r):
        n1 = m - l + 1
        n2 = r - m
        # create temp arrays
        L = [0] * (n1)
        R = [0] * (n2)
        # Copy data to temp arrays L[] and R[]
        for i in range(0, n1):
                L[i] = arr[l + i]
        for  j in range(0, n2):
                R[j] = arr[m + 1 + j]
        # Merge the temp arrays back into arr[l..r]
        i = 0 # Initial index of first subarray
        j = 0       # Initial index of second subarray
        k = l # Initial index of merged subarray
        while i < n1 and j < n2:
                if L[i] <= R[j]:
                        arr[k] = L[i]
                        i += 1
                else:
                        arr[k] = R[j]
                        j += 1
                k += 1
        # Copy the remaining elements of L[], if there # are any
        while i < n1:
                arr[k] = L[i]
                i += 1
                k += 1
        # Copy the remaining elements of R[], if there are any
        while j < n2:
                arr[k] = R[j]
                j += 1
                k += 1
# l is for left index and r is right index of the
# sub-array of arr to be sorted
def mergeSort(arr, l, r):
```

```
        if l < r:
                # Same as (l+r)//2, but avoids overflow for large l and h
                m = l+(r-l)//2
                # Sort first and second halves
                mergeSort(arr, l, m)
                mergeSort(arr, m+1, r)
                merge(arr, l, m, r)
# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print("Given array is")
for i in range(n):
        print("%d" % arr[i]),
mergeSort(arr, 0, n-1)
print("\n\nSorted array is")
for i in range(n):
        print("%d" % arr[i]),
```

OUTPUT

```
Given array  is
12  11  13  5  6  7

Sorted array is
5  6  7  11  12  13
```

5. Write a program to implement Quick Sort for the given list of integer values.

```
# Python program for implementation of Quicksort Sort
def partition(arr, low, high):
    i = (low-1)             # index of smaller element
    pivot = arr[high]       # pivot
    for j in range(low, high):
        # If current element is smaller than or
        # equal to pivot
        if arr[j] <= pivot:
            # increment index of smaller element
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)
# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low --> Starting index,
# high --> Ending index
# Function to do Quick sort
def quickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:
        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr, low, high)
        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)
# Driver code to test above
arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr, 0, n-1)
print("Sorted array is:")
for i in range(n):
    print("%d" % arr[i])
```

OUTPUT

```
Sorted array is:
1
5
7
8
9
10


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

6. Write a program to find the solution for the knapsack problem using the greedy method.

```
# Python3 program to solve
fractional# Knapsack Problem
class ItemValue:

    """Item Value DataClass"""
    def __init_(self, wt, val, ind):
        self.wt = wt
        self.val = val
        self.ind = ind
        self.cost = val //
        wt

    def __lt_(self, other):
        return self.cost < other.cost

# Greedy Approach
class FractionalKnapSack:

    """Time Complexity O(n log
    n)"""@staticmethod
    def getMaxValue(wt, val, capacity):

        """function to get maximum value
        """iVal = [ ]
        for i in range(len(wt)):
            iVal.append(ItemValue(wt[i], val[i],
            i))

        # sorting items by value
        iVal.sort(reverse =
        True)totalValue = 0
        for i in iVal:
            curWt = int(i.wt)
            curVal =
            int(i.val)
            if capacity - curWt >=
                0:capacity -= curWt
                totalValue += curVal
            else:
                fraction = capacity / curWt
                totalValue += curVal *
                fraction
                capacity = int(capacity - (curWt *
                fraction))break
```

```
        return totalValue



# Driver Code
If___name___== "_main_":
    wt = [10, 40, 20, 30]
    val = [60, 40, 100, 120]
    capacity = 50

  maxValue = FractionalKnapSack.getMaxValue(wt, val,
  capacity)print("Maximum value in Knapsack =", maxValue)
```

OUTPUT:

Maximum value in Knapsack = 240.0

7. Write a program to find minimum cost spanning tree using Prim's Algorithm

```
    # Prim's Algorithm in Python

    INF = 9999999
    # number of vertices in graphN = 5
    #creating graph    by adjacency matrix    method
    G = [[0, 19, 5,    0, 0],
         [19, 0, 5,    9, 2],
          [5, 5, 0, 1, 6],
          [0, 9, 1, 0, 1],
          [0, 2, 6, 1, 0]]
selected_node = [0, 0, 0, 0, 0]

no_edge = 0
selected_node[0] = True

# printing for edge and weight
print("Edge : Weight\n")
while (no_edge < N - 1):
minimum = INF
    a = 0
    b = 0
    for m in range(N):
        if selected_node[m]:
            for n in range(N):
                if ((not selected_node[n]) and G[m][n]):
                    # not in selected and there is an edge
                    if
                        mini
                        mu
                        m >
                        G[m
                        ][n]:
                        mini
                        mu
                        m =
                        G[m
                        ][n]
                        a =
                        m
                        b = n
    print(str(a) + "-" + str(b) + ":" + str(G[a][b]))
    selected_node[b] = True
    no_edge += 1
```

8. Write a program to find minimum cost spanning tree using Kruskal's algorithm.

```
# Python program for Kruskal's algorithm to
find # Minimum Spanning Tree of a given
connected,# undirected and weighted graph
from collections import
defaultdict# Class to represent a
graph
class Graph:
        def__init__(self, vertices):
    self.V = vertices # No. of vertices self.graph = [] #
    default dictionary# to store graph
        # function to add an edge to
        graphdef addEdge(self, u, v, w):
    self.graph.append([u, v, w])
        # A utility function to find set of an
        element i# (uses path compression
        technique)
       def find(self, parent, i):
            if parent[i] == i:
                return i
    return self.find(parent, parent[i])
        # A function that does union of two sets of x
        and y# (uses union by rank)
def union(self, parent, rank, x, y): xroot =
    self.find(parent, x)yroot =
    self.find(parent, y)
    # Attach smaller rank tree under root of# high rank
    tree (Union by Rank)
                if rank[xroot] < rank[yroot]:
                    parent[xroot] = yroot
                elif rank[xroot] >
                    rank[yroot]:
                    parent[yroot] =
                    xroot
    # If ranks are same, then make one as root# and
    increment its rank by one
    else:
                    parent[yroot] =
                    xrootrank[xroot] +=
```

```
                                    1
# The main function to construct MST using Kruskal's# algorithm
                        def KruskalMST(self):
            result = [] # This will store the resultant MST
```

```
# An index variable, used for sorted edgesi = 0
# An index variable, used for result[ ]e = 0
# Step 1: Sort all the edges in
# non-decreasing order of their
# weight. If we are not allowed to change the# given graph,
we can create a copy of graph
self.graph = sorted(self.graph, key=lambda item: item[2])parent = []
rank = []
# Create V subsets with single elementsfor node in
range(self.V):
                        parent.append(node
                        )rank.append(0)
# Number of edges to be taken is equal to V-1while e < self.V
- 1:
                        # Step 2: Pick the smallest edge and
                        increment# the index for next iteration
                        u, v, w =
                        self.graph[i]i = i + 1
                        x = self.find(parent,
                        u)y =
                        self.find(parent, v)
                        # If including this edge does't
                        # cause cycle, include it in result
                        # and increment the indexof
                        result# for next edge
                        if x != y:
                                e = e + 1
                                result.append([u, v,
                                w])
                                self.union(parent, rank, x,
                        y)# Else discard the edge
                minimumCost = 0
                print ("Edges in the constructed
                MST")for u, v, weight in result:
                        minimumCost += weight
                        print("%d -- %d == %d" % (u, v,
# Driver          weight)) print("Minimum Spanning Tree" ,
codeg =         minimumCost)
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)# Function call
g.KruskalMST()
```

OUOTPUT:

```
Edges in the constructed
MST2 - - 3 = = 4
0 - - 3 = = 5
0 - - 1 = = 10
Minimum Spanning Tree 19
```

9. Write a program to find a single source shortest path for a given graph.

```
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]
        def minDistance(self, dist, sptSet):
                min = sys.maxint
                for u in range(self.V):
                        if dist[u] < min and sptSet[u] == False:
                                min = dist[u]
                                min_index = u
                return min_index
        def dijkstra(self, src):
                dist = [sys.maxint] * self.V
                dist[src] = 0
                sptSet = [False] * self.V
                for cout in range(self.V):
                        x = self.minDistance(dist, sptSet)
                        sptSet[x] = True
                        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)
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

Program 10:

Title:

Write a program to find the solution for job sequencing with deadlines problems.

Program:

```
def printJobScheduling(arr, t):n =
        len(arr)
        for i in range(n):
                        for j in range(n - 1 - i):
                                if arr[j][2] < arr[j + 1][2]:
                                        arr[j], arr[j + 1] = arr[j + 1], arr[j]
        result = [False] * tjob = ['-
        1'] * t
        for i in range(len(arr)):
                        for j in range(min(t - 1, arr[i][1] - 1), -1, -
                                1):if result[j] is False:
                                        result[j] = True
                                        job[j] =
                                        arr[i][0]break
                print(job)
         arr = [['a', 2, 100],
        ['b', 1, 19],
        ['c', 2, 27],
        ['d', 1, 25],
        ['e', 3, 15]]
         print("Following is maximum profit sequence of jobs")
         printJobScheduling(arr, 3)
```

OUTPUT:

Following is maximum profit sequence of
jobs['c', 'a', 'e']