Ramniranjan Jhunjhunwala College of Arts, Science and Commerce (Autonomous).

Department Of Data Science and Artificial Intelligence.

CERTIFICATE

This is to certify to Krish Kevin Kounder of Class: SY -Bsc (DSAI).

Roll No: 10132 has successfully completed the practical Design and Analysis of Algorithm during the Academic Year 2025-2026.

Date:

Prof:

Prof-In-Charge

INDEX

SR NO.	PRACTICAL	DATE	SIGN
1	Write a Program that Implements Following Sorting.	15-11-2025	
2	Write a Program to Implement the 8-Queen's problem using Backtracking.	20-11-2025	
3	Program to Implement Minimum and Maximum using Divide and Conquer.	29-11-2025	
4	Program to Implement Merge Sort using Divide and Conquer.	4-12-2025	
5	Program to Implement Knapsack Algorithm using Greedy Method.	13-12-2025	
6	Program to Implement Prim's algorithm using Greedy Method.	18-12-2025	
7	Program to Implement Kruskal's Algorithm using Greedy Method.	20-12-2025	
8	Program to Implement the Problems using Backtracking.	3-1-2026	
9	Write a Program to Perform Binary Search for a given Set of Integer Values Recursively and Nonrecursively.	8-1-2026	
10	Program to Implement All Pairs Shortest Path Using Dynamic Programming.	13-1-2026	
11	Write a Program to Implement Dijkstra Algorithm.	17-1-2026	

PRACTICAL 1:

1)Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0,n-i-1):
            if arr[j] > arr[j+1]:
                arr[j],arr[j+1] = arr[j+1],arr[j]
                swapped = True
        if not swapped:
            break
    return arr

arr = [3,7,8,5,10,6]
sorted_arr = bubble_sort(arr)
print("Sorted list:",sorted_arr)
```

2)Selection Sort

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i+1,n):
            if arr[j] < arr[min_index]:
                min_index = j
            arr[i],arr[min_index] = arr[min_index],arr[i]
    return arr

arr = [-2,45,0,11,-9,88,-97,-202,747]
```

```
selection_sort(arr)
print(arr)
```

3)Quick Sort

```
def partition(arr,low,high):
    pivot  = arr[high]
    i = low-1
    for j in range(low,high):
        if arr[i] <= pivot:
            i +=1
            arr[i],arr[j] = arr[j],arr[i]
        arr[i +1],arr[high] = arr[high],arr[i +1]
        return i+1
def quick_sort(arr,low,high):
    if low<high:
        p = partition(arr,low,high)
        quick_sort(arr,low,p-1)
        quick_sort(arr,p+1,high)
arr = [3,6,4,10,1,5]
quick_sort(arr,0,len(arr)-1)
print(arr)
```

PRACTICAL 2:

1)N-Queens Algorithm

```
N = 4  # Chessboard size

def is_safe(board, row, col):
    # Check previous rows for a queen in the same column
    for i in range(row):
        if board[i] == col:
            return False

    # Check upper-left diagonal
    i, j = row - 1, col - 1
    while i >= 0 and j >= 0:
        if board[i] == j:
            return False
        i -= 1
        j -= 1

    # Check upper-right diagonal
    i, j = row - 1, col + 1
    while i >= 0 and j < N:
        if board[i] == j:
            return False
        i -= 1
        j += 1

    return True


def solve_queens(board, row):
    if row == N:
```

```
            print_solution(board)
            return True

        found_solution = False
        for col in range(N):
            if is_safe(board, row, col):
                board[row] = col
                if solve_queens(board, row + 1):
                    found_solution = True
                board[row] = -1

        return found_solution
    def print_solution(board):
        print("\nSolution:")
        for row in range(N):
            line = ""
            for col in range(N):
                if board[row] == col:
                    line += " Q "
                else:
                    line += " . "
            print(line)
    # Driver Code
    board = [-1] * N
    solve_queens(board, 0)
```

PRACTICAL 3: Divide And Conquere

1)Max and Min

```
#Max and Min using Divide and Conquere
def findMinMax(arr,low,high):
```

```
    min_val = arr[low]
    max_val = arr[high]
    # If there is only one element
    if low == high:
        return(min_val,max_val)
    # If there is only two elements
    elif high == low+1:
        if arr[low] > arr[high]:
            min_val = arr[high]
            max_val = arr[low]
        else:
            min_val = arr[low]
            max_val = arr[high]
        return(min_val,max_val)
    else:
        mid = (low +high) //2
        min_val_left , max_val_left = findMinMax(arr,low,mid)
        min_val_right , max_val_right = findMinMax(arr, mid + 1,high)
        if min_val_left < min_val_right:
            min_val = min_val_left
        else:
            min_val = min_val_right
        if max_val_left > max_val_right:
            max_val = max_val_left
        else:
            max_val = max_val_right
        return (min_val,max_val)

if __name__ == '__main__':
```

```
    my_list = []
    n = int(input("Enter the number of Elements:"))
    for i in range(n):
        user_input = int(input("Enter elements that you want in your array:"))
        my_list.append(user_input)
    print("Given array is:",my_list)
    min_val,max_val = findMinMax(my_list,0,n-1)
    print("Minimum Value is:",min_val)
    print("Maximum Value is:",max_val)
```

2)Merge Sort

```
# Function to merge two sorted arrays
def merge(arr1,arr2):
    i = 0
    j = 0
    #Array to storethe merged sorted array
    result = []
    while(i < len(arr1) and j< len(arr2)):
        #if arr1[i] is smaller than arr2[j],push arr1[i] into the result
        #and move to the next element in arr1
        if arr2[j] <= arr1[i]:
            result.append(arr1[i])
            i +=1
        #if arr2[j] is smaller than arr1[i],push arr2[j] into the result and move to
        else:
            result.append(arr2[j])
            j +=1
    while(i < len(arr1)):
        result.append(arr1[i])
        i +=1
```

```
    while(j < len(arr2)):
        result.append(arr2[j])
        j +=1
    return result

def mergeSort(arr):
    if len(arr) <=1:
        return arr
    mid = len(arr)//2
    #Sort the Left Half
    left=  mergeSort(arr[:mid])
    #Sort the right Half
    right =  mergeSort(arr[mid:])
    return merge(left,right)

if __name__ == '__main__':
    my_arr = []
    n = int(input("Enter the total number of elements you wan't in your array:"))
    for i in range(n):
        user_input = int(input("Enter the numbers you wan't in array"))
        my_arr.append(user_input)
    print("Given array is:",my_arr)
    print(*my_arr)
    arr = mergeSort(my_arr)
    print("\nSorted array is ")
    print(*arr)
```

PRACTICAL 4:

```
import heapq

def prim(graph):
    if not graph:
        return [], 0

    # Start from any node
```

```
start_node = list(graph.keys())[0]
mst_nodes = {start_node}      # FIXED name
mst_edges = []
total_weight = 0

# Min-heap for edges
edges = []
for weight, neighbor in graph[start_node]:
    heapq.heappush(edges, (weight, start_node, neighbor))

# Main loop
while edges and len(mst_nodes) < len(graph):
    weight, u, v = heapq.heappop(edges)

    # If v already in MST, skip
    if v in mst_nodes:
        continue

    # Otherwise add to MST
    mst_nodes.add(v)
    mst_edges.append((u, v, weight))
    total_weight += weight

    # Add edges from v
    for w, neighbor in graph[v]:
        if neighbor not in mst_nodes:
            heapq.heappush(edges, (w, v, neighbor))

return mst_edges, total_weight
```

```
graph = {
    'A': [(1, 'B'), (3, 'C')],
    'B': [(1, 'A'), (4, 'C'), (2, 'D')],
    'C': [(3, 'A'), (4, 'B'), (5, 'D')],
    'D': [(2, 'B'), (5, 'C')]
}
pip install networkx

import networkx as nx
import matplotlib.pyplot as plt

def draw_graph(graph):
    G = nx.Graph()

    # Add edges
    for u in graph:
        for w, v in graph[u]:
            G.add_edge(u, v, weight=w)

    pos = nx.spring_layout(G)  # positions for nodes

    # Draw nodes and edges
    nx.draw(G, pos, with_labels=True, node_size=1500, node_color="skyblue",
font_size=12)

    # Draw edge labels (weights)
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
```

```
    plt.show()
```

PRACTICAL 6: Knapsack Algorithm

```
# Knapsack Algorithm
  def compare(a,b):
      a1 = (1.0*a[0])/a[1]
      b1 = (1.0*b[0])/b[1]
      return b1 - a1
  def fract_kan(val,wt,cap):
      n = len(val)
      item = [[val[i],wt[i]] for i in range(n)]
      item.sort(key= lambda x: x[0]/x[1],reverse = True)
      res = 0.0
```

```
    curCap= cap
    for i in range(n):
        if item[i][1] <=curCap:
            res += item[i][0]
            curCap -= item[i][1]
        else:
            res += (1.0*item[i][0]/item[i][1])*curCap
            break
    return res
if  __name__ == "__main__":
  val= [60,100,120]
  wt = [10,20,30]
  cap = 50
  print(fract_kan(val,wt,cap))
```

PRACTICAL 7: Kruskal Wali Test(For Directed Graph)

```
#Kruskal Algorithm(Used For Directed Graph) ex GPS,Routers,Messages sending
class Graph:
  def __init__(self,vertices):
      self.V = vertices
      self.graph = []

  #Function to add an edge to graph
  def addEdge(self,u,v,w):
      self.graph.append([u,v,w])

  def find(self,parent,i):
      if parent[i] !=i:
```

```
        parent[i] = self.find(parent,parent[i])
    return parent[i]
def union(self,parent,rank,x,y):
    if rank[x] < rank[y]:
        parent[x] = y
    elif rank[x] > rank[y]:
        parent[y] = x
    else:
        parent[y] = x
        rank[x] +=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)
    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(0,1,20)
g.addEdge(0,2,8)
g.addEdge(0,3,10)
g.addEdge(1,3,15)
g.addEdge(2,3,4),g.kruskalMST()
```

PRACTICAL 8:

1)Binary Search

```
        def Binary_search(arr,x):
            left = 0
            right = len(arr) -1
            while left<= right:
                mid = (left+right) //2
                if arr[mid] == x:
                    return mid
                elif arr[mid] < x:
                    left = mid+1
                else:
                    right = mid-1
            return -1
        if __name__ == '__main__':
            arr = [2,5,8,10,15,20]
            x = int(input("Enter the number you wan't to find: "))
            print(f"You entered:{x}")
            result = Binary_search(arr,x)
            if result !=-1:
                print("Element is present at index",result)
            else:
                print("Element is not present in array")
```

2)Binary search recursive

```
def Binary_search(arr,left,right,x):
    print(arr.sort())
    while left<= right:
        mid = (left+right) //2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return Binary_search(arr,left,mid-1,x)
        else:
            return Binary_search(arr,mid+1,right,x)
    else:
      return -1
if __name__ == '__main__':
    arr = [2,5,8,10,15,11,14,22,32,4,20]
    x = int(input("Enter the number you wan't to find: "))
    print(f"You entered:{x}")
    result = Binary_search(arr,0,len(arr)-1,x)
    if result !=-1:
        print("Element is present at index",result)
    else:
        print("Element is not present in array")
```

PRACTICAL 9:

```
def print_board(board):
    for row in board:
        print(" ".join(str(num) for num in row))



def is_safe(board, row, col, num):
    # Check row
    for x in range(4):
        if board[row][x] == num:
            return False

    # Check column
    for x in range(4):
        if board[x][col] == num:
            return False

    # Check 2x2 subgrid
```

```
    start_row = row - row % 2
    start_col = col - col % 2
    for i in range(2):
        for j in range(2):
            if board[start_row + i][start_col + j] == num:
                return False

    return True



def find_empty_location(board):
    for row in range(4):
        for col in range(4):
            if board[row][col] == 0:
                return row, col
    return None



def solve_sudoku(board):
    empty = find_empty_location(board)
    if not empty:
        return True

    row, col = empty

    for num in range(1, 5):
        if is_safe(board, row, col, num):
            board[row][col] = num
```

```
            if solve_sudoku(board):
                return True

            board[row][col] = 0  # Backtracking

    return False



board = [
    [1, 0, 3, 0],
    [2, 0, 1, 0],
    [4, 0, 2, 0],
    [3, 0, 4, 0]
]

print("Original Sudoku:")
print_board(board)

if solve_sudoku(board):
    print("\nSolved Sudoku:")
    print_board(board)
else:
    print("No solution exists")
```

PRACTICAL 10: Floyd Warshall Algorithm

```
# Floyd Warshall
def floyd_warshall(graph):
    n = len(graph)
    dist =  [row[:] for row in graph]
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]
    return dist

INF = float('inf')
graph = [[0,6,INF,5],[2,0,INF,4],[INF,1,0,INF],[INF,INF,INF,0]]
shortest_paths = floyd_warshall(graph)
for row in shortest_paths:
    print(row)
```

PRACTICAL 11:Dijkstra Algorithm

```
import heapq
import sys

def dijkstra(adj,src):
    V = len(adj)
    #Min-heap (priority Queue) storing pairs of (distance,node)
    pq = []
    dist = [sys.maxsize] *V
    #Distance from source itself is 0
    dist[src] = 0
    heapq.heappush(pq,(0,src))
    # Process the queue until all reachable vertices are finalized
    while pq:
        d,u = heapq.heappop(pq)
        #If this distance not the latest
        if d> dist[u]:
            continue
        #Explore neighbors of the current verttex
        for v,w in adj[u]:
            #If we found a shorter path to v through u,update it
```

```
            if dist[u] + w<dist[v]:
                dist[v] = dist[u] + w
                heapq.heappush(pq,(dist[v],v))
    return dist


if __name__ == "__main__":
    src = 0
    adj = [[(1,4),(2,8)],[(0,4),(4,6)],[(0,8),(1,3)],[(2,2),(4,10)],[(1,6),(3,10)]]
    result = dijkstra(adj,src)
    print(result)
```