PART-B

1. Write a program to sort a given set of **n integer elements** using **Merge Sort method** and compute its time complexity. Run the program for varied values of *n > 5000*, and record the time taken to sort.
2. Write a program to sort a given set of **n integer elements** using **Quick Sort method** and compute its time complexity. Run the program for varied values of *n > 5000* and record the time taken to sort.
3. Write a **C program** that accepts the vertices and edges for a graph and stores it as an **adjacency matrix**.
4. Write a program to implement a function to **print In-Degree, Out-Degree** and to display that **adjacency matrix**.
5. Write a program to implement **backtracking algorithm** for solving problems like **N-Queens**.
6. Write a program to implement the **backtracking algorithm** for the **Sum of Subsets problem**.
7. Write a program to implement a **Greedy algorithm** for **Job Sequencing with Deadlines**.
8. Write a program to implement **Dynamic Programming algorithm** for the **Optimal Binary Search Tree Problem**.
9. Write a program that implements **Prim's algorithm** to generate a **Minimum Cost Spanning Tree**.
10. Write a program that implements **Kruskal's algorithm** to generate a **Minimum Cost Spanning Tree**.

PART-B

1. **Write a C program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n > 5000, and record the time taken to sort.**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void merge(int a[], int l, int m, int r)
{
   int n1=m-l+1,n2=r-m,L[n1],R[n2],i,j,k;
   for(i=0;i<n1;i++) L[i]=a[l+i];
   for(j=0;j<n2;j++) R[j]=a[m+1+j];
   for(i=0,j=0,k=l;i<n1 && j<n2;) a[k++]=(L[i]<=R[j])?L[i++]:R[j++];
   while(i<n1) a[k++]=L[i++];
   while(j<n2) a[k++]=R[j++];
}
void mergeSort(int a[], int l, int r)
 {
   if(l<r){int m=(l+r)/2;
  mergeSort(a,l,m);
   mergeSort(a,m+1,r);
  merge(a,l,m,r);
}
}
int main()
{
   int n;
  printf("Enter n (>5000): ");
  scanf("%d",&n);
   int *a=malloc(n*sizeof(int));
   for(int i=0;i<n;i++)
 a[i]=rand()%10000;
   clock_t t=clock();
   mergeSort(a,0,n-1);
   t=clock()-t;
   printf("Time taken: %f sec\n",(double)t/CLOCKS_PER_SEC);
   free(a);
   return 0;
}
```

Output:

```
Enter n (>5000): 6000
Time taken: 0.000882 sec
```

PART-B

2. **Write a program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of *n > 5000* and record the time taken to sort.**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void quick(int a[], int l, int h)
{
   if (l >= h) return;
   int i = l, j = h, p = a[(l + h) / 2];
   while (i <= j)
{
      while (a[i] < p) i++;
      while (a[j] > p) j--;
      if (i <= j)
      {
         int t = a[i];
         a[i] = a[j];
         a[j] = t;
         i++;
         j--;
      }
   }
   quick(a, l, j);
   quick(a, i, h);
}
int main()
{
   int n;
   printf("Enter n (>5000): ");
   scanf("%d", &n);
   int *a = malloc(n * sizeof(int));
   for (int i = 0; i < n; i++)
   a[i] = rand() % 100000;
   clock_t s = clock();
   quick(a, 0, n - 1);
   clock_t e = clock();
   printf("Time: %.6f sec\n", (double)(e - s) / CLOCKS_PER_SEC);
   free(a);
   return 0;
}
```

OUTPUT:

```
Enter n (>5000): 5800
Time: 0.000560 sec
```

PART-B

3. Write a C program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.

```
#include <stdio.h>
int main()
{
   int v, e, i, j, src, dest;
   printf("Enter number of vertices: ");
   scanf("%d", &v);
   int adj[v][v];
   // Initialize matrix to 0
   for(i = 0; i < v; i++)
      for(j = 0; j < v; j++)
         adj[i][j] = 0;
   printf("Enter number of edges: ");
   scanf("%d", &e);
   // Read edges
   for(i = 0; i < e; i++)
 {
      printf("Enter edge (src dest): ");
      scanf("%d%d", &src, &dest);
      adj[src][dest] = 1;       // For directed graph
      adj[dest][src] = 1;        // For undirected graph, keep this line
   }
   // Display adjacency matrix
   printf("Adjacency Matrix:\n");
   for(i = 0; i < v; i++)
   {
      for(j = 0; j < v; j++)
         printf("%d ", adj[i][j]);
      printf("\n");
   }
   return 0;
}
```

Output:

```
Enter number of vertices: 3
Enter number of edges: 2
Enter edge (src dest): 0 1,1 2
Enter edge (src dest): Adjacency Matrix:
                       0 1 0
                       1 0 0
                       0 0 0
```

PART-B

4. Write a program to implement a function to print In-Degree, Out-Degree and to display that adjacency matrix.

```
#include <stdio.h>
#define MAX 10
void displayAdjMatrix(int adj[MAX][MAX], int v)
{
   printf("\nAdjacency Matrix:\n");
   for(int i = 0; i < v; i++) {
      for(int j = 0; j < v; j++) {
         printf("%d ", adj[i][j]);
      }
      printf("\n");
   }
}
void computeDegrees(int adj[MAX][MAX], int v)
{
   int indegree, outdegree;
   printf("\nVertex\tIn-Degree\tOut-Degree\n");
   for(int i = 0; i < v; i++)
   {
      indegree = outdegree = 0;
      // Calculate Out-Degree (row-wise)
      for(int j = 0; j < v; j++)
         outdegree += adj[i][j];
      // Calculate In-Degree (column-wise)
      for(int j = 0; j < v; j++)
         indegree += adj[j][i];
      printf("%d\t\t%d\t\t%d\n", i, indegree, outdegree);
   }
}
int main()
{
   int v, e, src, dest;
   int adj[MAX][MAX] = {0};
   printf("Enter number of vertices: ");
   scanf("%d", &v);
   printf("Enter number of edges: ");
   scanf("%d", &e);
   printf("Enter edges (source destination):\n");
   for(int i = 0; i < e; i++) {
      scanf("%d %d", &src, &dest);
      adj[src][dest] = 1; // directed edge
   }
   displayAdjMatrix(adj, v);
   computeDegrees(adj, v);
   return 0;
```

```
}
```

OUTPUT:

```
Enter number of vertices: 4
Enter number of edges: 4
Enter edges (source destination):
0 1
0 2
1 2
2 3
Adjacency Matrix:
0 1 1 0
0 0 1 0
0 0 0 1
0 0 0 0
```

Vertex	In-Degree	Out-Degree
0	0	2
1	1	1
2	2	1
3	1	0

5. Write a program to implement backtracking algorithm for solving problems like N-Queens.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
int board[MAX], n;
int isSafe(int row, int col)
{
   for (int i = 0; i < row; i++)
      if (board[i] == col || abs(board[i] - col) == abs(i - row))
         return 0;
   return 1;
}
void printSolution()
{
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)
         printf("%c ", board[i] == j ? 'Q' : '.');
      printf("\n");
   }
   printf("\n");
}
void solveNQueens(int row)
{
   if (row == n) { printSolution(); return;
   }
   for (int col = 0; col < n; col++) {
```

```
        if (isSafe(row, col)) {
            board[row] = col;
            solveNQueens(row + 1);
        }
    }
}
int main()
{
    printf("Enter number of Queens: ");
    scanf("%d", &n);
    solveNQueens(0);
    return 0;
}
```

Output:

```
Enter number of Queens: 4
. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. . . Q
. Q . .
```

6. Write a program to implement the backtracking algorithm for the Sum of Subsets problem.

```
#include <stdio.h>
#define MAX 20
int n, target, set[MAX], subset[MAX];
void sumOfSubsets(int idx, int currSum, int start)
{
    if (currSum == target)
    {       // Found a valid subset
        printf("{ ");
        for (int i = 0; i < idx; i++)
            printf("%d ", subset[i]);
        printf("}\n");
        return;      // Continue to find other subsets
    }
    for (int i = start; i < n; i++)
    {
        if (currSum + set[i] <= target)
        { // Choose the element
            subset[idx] = set[i];
            sumOfSubsets(idx + 1, currSum + set[i], i + 1);
        }
    }
```

```
}
int main()
{
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter elements of the set: ");
   for (int i = 0; i < n; i++)
      scanf("%d", &set[i]);
   printf("Enter target sum: ");
   scanf("%d", &target);
   printf("Subsets with sum %d:\n", target);
   sumOfSubsets(0, 0, 0);
   return 0;
}
```

Output:

Enter number of elements: 4

Enter elements of the set: 3 4 5 2

Enter target sum: 7

Subsets with sum 7:

{ 3 4 }

{ 5 2 }

7. Write a program to implement a Greedy algorithm for Job Sequencing with Deadlines.

```
#include <stdio.h>
#define MAX 20
typedef struct
{
   char id;
   int deadline;
   int profit;
} Job;
void jobSequencing(Job jobs[], int n)
{
   int i, j, maxDeadline = 0;
   int slot[MAX];
   char result[MAX];
   // Find maximum deadline
   for (i = 0; i < n; i++)
      if (jobs[i].deadline > maxDeadline)
         maxDeadline = jobs[i].deadline;
   // Sort jobs by decreasing profit (Greedy choice)
   for (i = 0; i < n - 1; i++)
      for (j = i + 1; j < n; j++)
         if (jobs[i].profit < jobs[j].profit)
         {
            Job temp = jobs[i];
            jobs[i] = jobs[j];
            jobs[j] = temp;
         }
   // Initialize all slots as free
   for (i = 0; i < maxDeadline; i++)
      slot[i] = -1;
```

```
    // Assign jobs to slots
    for (i = 0; i < n; i++)
    {
        for (j = jobs[i].deadline - 1; j >= 0; j--)
        {
            if (slot[j] == -1)
            {       // If slot is free
                slot[j] = i;          // Assign job index
                break;
            }
        }
    }
    // Print the scheduled jobs
    printf("\nScheduled Jobs: ");
    for (i = 0; i < maxDeadline; i++)
        if (slot[i] != -1)
            printf("%c ", jobs[slot[i]].id);
}
int main() {
    int n, i;
    Job jobs[MAX];
    printf("Enter number of jobs: ");
    scanf("%d", &n);
    printf("Enter Job ID, Deadline, and Profit for each job:\n");
    for (i = 0; i < n; i++)
 scanf(" %c %d %d", &jobs[i].id, &jobs[i].deadline, &jobs[i].profit);
    jobSequencing(jobs, n);
    return 0;
}
```

Output:

Enter number of jobs: 5
Enter Job ID, Deadline, and Profit for each job:
A 2 100
B 1 19
C 2 27
D 1 25
E 3 15
Scheduled Jobs: C A E

8. Write a program to implement Dynamic Programming algorithm for the Optimal Binary Search Tree Problem.

```
#include <stdio.h>
#define MAX 20
#define INF 999999
int main()
{
    int n, i, j, k, l, r;
    int freq[MAX], cost[MAX][MAX], weight[MAX][MAX], root[MAX][MAX];
    printf("Enter number of keys: ");
```

```
    scanf("%d", &n);
    printf("Enter frequency of %d keys:\n", n);
    for (i = 1; i <= n; i++)
        scanf("%d", &freq[i]);
    // Initialize cost and weight for single keys
    for (i = 1; i <= n; i++)
    {
        cost[i][i] = freq[i];
        weight[i][i] = freq[i];
        root[i][i] = i;
    }
    // For chains of length 2 to n
    for (l = 2; l <= n; l++)
    {
        for (i = 1; i <= n - l + 1; i++)
        {
            j = i + l - 1;
            cost[i][j] = INF;
            weight[i][j] = weight[i][j - 1] + freq[j];
            for (r = i; r <= j; r++) {
                int c = ((r > i) ? cost[i][r - 1] : 0) +
                        ((r < j) ? cost[r + 1][j] : 0) +
                        weight[i][j];
                if (c < cost[i][j])
                {
                    cost[i][j] = c;
                    root[i][j] = r;
                }
            }
        }
    }
    printf("\nOptimal Cost of OBST: %d\n", cost[1][n]);
    printf("Root of OBST: Key %d\n", root[1][n]);
    return 0;
}
```

Output:

```
Enter number of keys: 4
Enter frequency of 4 keys:
4 2 6 3
Optimal Cost of OBST: 26
Root of OBST: Key 3
```

PART-B

9. Write a program that implements Prim's algorithm to generate a Minimum Cost Spanning Tree.

```
#include <stdio.h>
#include <limits.h>
#define MAX 20
int main()
{
   int n, i, j, k;
   int cost[MAX][MAX], selected[MAX] = {0}, min, u, v, edge_count = 0, total_cost = 0;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter the adjacency matrix (0 if no edge):\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
      {
         scanf("%d", &cost[i][j]);
         if (cost[i][j] == 0)
            cost[i][j] = INT_MAX;  // Treat 0 as no edge
      }
   selected[0] = 1;  // Start from vertex 0
   printf("\nEdges in Minimum Spanning Tree:\n");
   while (edge_count < n - 1)
   {
      min = INT_MAX;
      u = v = -1;
      for (i = 0; i < n; i++)
      {
         if (selected[i]) {
            for (j = 0; j < n; j++)
            {
               if (!selected[j] && cost[i][j] < min)
               {
                  min = cost[i][j];
                  u = i;
                  v = j;
               }
            }
         }
      }
      if (u != -1 && v != -1)
      {
         printf("%d - %d : %d\n", u, v, cost[u][v]);
         total_cost += cost[u][v];
         selected[v] = 1;
         edge_count++;
      }
   }
   printf("\nTotal cost of MST: %d\n", total_cost);
   return 0;
}
```

Output:

Enter number of vertices: 4

```
Enter the adjacency matrix (0 if no edge):
0 2 0 6
2 0 3 8
0 3 0 0
6 8 0 0

Edges in Minimum Spanning Tree:
0 - 1 : 2
1 - 2 : 3
0 - 3 : 6

Total cost of MST: 11
```

10. Write a program that implements Kruskal's algorithm to generate a Minimum Cost Spanning Tree.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
typedef struct
{
   int u, v, weight;
} Edge;
int parent[MAX];
int find(int i) {
   while (parent[i] != i)
      i = parent[i];
   return i;
}
void union_sets(int i, int j)
{
   int a = find(i);
   int b = find(j);
   parent[a] = b;
}
int compare(const void *a, const void *b)
{
   return ((Edge *)a)->weight - ((Edge *)b)->weight;
}
int main()
{
   int n, e, i, count = 0, total_cost = 0;
   Edge edges[MAX];
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter number of edges: ");
   scanf("%d", &e);
   printf("Enter edges (u v weight):\n");
   for (i = 0; i < e; i++) {
   scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].weight);
   }
   // Initialize parent for union-find
   for (i = 0; i < n; i++)
      parent[i] = i;
```

```
    // Sort edges by weight
    qsort(edges, e, sizeof(Edge), compare);
    printf("\nEdges in Minimum Spanning Tree:\n");
    for (i = 0; i < e && count < n - 1; i++)
    {
        int u_set = find(edges[i].u);
        int v_set = find(edges[i].v);
        if (u_set != v_set) {
            printf("%d - %d : %d\n", edges[i].u, edges[i].v, edges[i].weight);
            total_cost += edges[i].weight;
            union_sets(u_set, v_set);
            count++;
        }
    }
    printf("\nTotal cost of MST: %d\n", total_cost);
    return 0;
}
```

Output:

```
Enter number of vertices: 4
Enter number of edges: 5
Enter edges (u v weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4

Edges in Minimum Spanning Tree:
2 - 3 : 4
0 - 3 : 5
0 - 1 : 10

Total cost of MST: 19
```