Experiment No :01 a) Selection Sort:

```
#include <stdio.h>

void selectionSort(int arr[], int n) {
   int i, j, minIndex, temp;
   for (i = 0; i < n - 1; i++) {
      minIndex = i;
      for (j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex])
            minIndex = j;
      }

      // Swap the found minimum with the first element
      temp = arr[minIndex];
      arr[minIndex] = arr[i];
      arr[i] = temp;

      // Print array after each step
      printf("Step %d: ", i + 1);
      for (int k = 0; k < n; k++)
         printf("%d ", arr[k]);
      printf("\n");
   }
}

void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}

int main() {
   int arr[] = {64, 25, 12, 22, 11};
   int n = sizeof(arr) / sizeof(arr[0]);

   // Print unsorted array
   printf("Unsorted array:\n");
   printArray(arr, n);

   // Perform selection sort with step-by-step display
   selectionSort(arr, n);

   // Print sorted array
   printf("Sorted array:\n");
   printArray(arr, n);
  return 0;
}
```

B)Insertion sort:

```
#include <stdio.h>

void insertionSort(int arr[], int n) {
   int i, key, j;
   for (i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;

      // Move elements greater than key to one position ahead
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;

      // Print array after each step
      printf("Step %d: ", i);
      for (int k = 0; k < n; k++)
         printf("%d ", arr[k]);
      printf("\n");
   }
}

void printArray(int arr[], int n) {
   for (int i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}

int main() {
   int arr[] = {64, 25, 12, 22, 11};
   int n = sizeof(arr) / sizeof(arr[0]);

   // Print unsorted array
   printf("Unsorted array:\n");
   printArray(arr, n);

   // Perform insertion sort with step-by-step display
   insertionSort(arr, n);

   // Print sorted array
   printf("Sorted array:\n");
   printArray(arr, n);

   return 0;
}
```

Experiment No 02: a) Merge sort

```
#include <stdio.h>

void merge(int a[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;

   int left[50], right[50];  // assuming size < 50

   for (i = 0; i < n1; i++)
      left[i] = a[l + i];
   for (j = 0; j < n2; j++)
      right[j] = a[m + 1 + j];

   i = j = 0;
   k = l;

   while (i < n1 && j < n2) {
      if (left[i] <= right[j])
         a[k++] = left[i++];
      else
         a[k++] = right[j++];
   }

   while (i < n1)
      a[k++] = left[i++];
   while (j < n2)
      a[k++] = right[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);

      printf("Step (l=%d, r=%d): ", l, r);
      for (int i = l; i <= r; i++)
         printf("%d ", a[i]);
      printf("\n");
   }
}

void print(int a[], int n) {
   for (int i = 0; i < n; i++)
```

```
        printf("%d ", a[i]);
    printf("\n");
}

int main() {
    int a[] = {38, 27, 43, 3, 9, 82, 10};
    int n = sizeof(a) / sizeof(a[0]);

    printf("Unsorted array:\n");
    print(a, n);

    printf("\nSorting steps:\n");
    mergeSort(a, 0, n - 1);

    printf("\nSorted array:\n");
    print(a, n);

    return 0;
}
```

B)Quick sort:

```
#include <stdio.h>

// Swap two numbers
void swap(int *a, int *b) {
   int t = *a;
   *a = *b;
   *b = t;
}

// Partition function
int partition(int a[], int low, int high) {
   int pivot = a[high];
   int i = low - 1;

   for (int j = low; j < high; j++) {
      if (a[j] < pivot) {
         i++;
         swap(&a[i], &a[j]);
      }
   }
   swap(&a[i + 1], &a[high]);
   return i + 1;
}

// Recursive Quick Sort
void quickSort(int a[], int low, int high) {
   if (low < high) {
      int pi = partition(a, low, high);

      // Print step-by-step progress
      printf("Step (low=%d, high=%d): ", low, high);
      for (int i = low; i <= high; i++)
         printf("%d ", a[i]);
      printf("\n");

      quickSort(a, low, pi - 1);
      quickSort(a, pi + 1, high);
   }
}

// Print array
void print(int a[], int n) {
   for (int i = 0; i < n; i++)
      printf("%d ", a[i]);
   printf("\n");
```

```
}

// Main
int main() {
   int a[] = {10, 7, 8, 9, 1, 5};
   int n = sizeof(a) / sizeof(a[0]);

   printf("Unsorted array:\n");
   print(a, n);

   printf("\nSorting steps:\n");
   quickSort(a, 0, n - 1);

   printf("\nSorted array:\n");
   print(a, n);

   return 0;
}
```

Exp :- 3

a). Dijkstra's Algorithm

```
#include <stdio.h>
#include <limits.h>
#define MAXN 100
#define INF INT_MAX/2

int n;  // number of vertices
int graph[MAXN][MAXN];
int distArr[MAXN];
int used[MAXN];

void dijkstra(int src) {
    for (int i = 0; i < n; i++) {
        distArr[i] = INF;
        used[i] = 0;
    }
    distArr[src] = 0;
    for (int i = 0; i < n; i++) {
        int u = -1;
        for (int j = 0; j < n; j++) {
            if (!used[j] && (u == -1 || distArr[j] < distArr[u])) {
                u = j;
            }
        }
        used[u] = 1;
        for (int v = 0; v < n; v++) {
            if (distArr[v] > distArr[u] + graph[u][v]) {
                distArr[v] = distArr[u] + graph[u][v];
            }
        }
    }
}

int main() {
    // Example
    n = 4;
    // initialize graph adjacency matrix
    for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) graph[i][j] = INF;
    // edges
    graph[0][1] = 1;
    graph[0][2] = 4;
    graph[1][2] = 2;
    graph[1][3] = 5;
     graph[2][3] = 1;

    dijkstra(0);

    printf("Distances from node 0:\n");
```

```
    for (int i = 0; i < n; i++) {
        printf("To %d = %d\n", i, distArr[i]);
    }

    return 0;
}
```

B):. Kruskal's Minimum Spanning Tree in C

```
#include <stdio.h>
#include <stdlib.h>

#define MAXV 100
#define MAXE 1000

typedef struct {
   int u, v, w;
} Edge;

Edge edges[MAXE];
int parent[MAXV];
int rankArr[MAXV];

int compareEdges(const void *a, const void *b) {
   Edge *ea = (Edge*)a;
   Edge *eb = (Edge*)b;
   return ea->w - eb->w;
}

void makeSet(int n) {
   for (int i = 0; i < n; i++) {
      parent[i] = i;
      rankArr[i] = 0;
   }
}

int findSet(int x) {
   if (parent[x] != x)
      parent[x] = findSet(parent[x]);
   return parent[x];
}

void unionSet(int a, int b) {
   a = findSet(a);
   b = findSet(b);
   if (a != b) {
      if (rankArr[a] < rankArr[b]) {
         parent[a] = b;
      } else if (rankArr[b] < rankArr[a]) {
         parent[b] = a;
      } else {
         parent[b] = a;
         rankArr[a]++;
      }
   }
}
```

```
void kruskal(Edge edges[], int V, int E) {
    qsort(edges, E, sizeof(Edge), compareEdges);
    makeSet(V);

    printf("Edges in the MST:\n");
    for (int i = 0; i < E; i++) {
        int u = edges[i].u;
        int v = edges[i].v;
        if (findSet(u) != findSet(v)) {
            printf("%d -- %d == %d\n", u, v, edges[i].w);
            unionSet(u, v);
        }
    }
}

int main() {
    int V = 4;  // number of vertices
    int E = 5;  // number of edges
    // define edges (u, v, weight)
    edges[0] = (Edge){0, 1, 1};
    edges[1] = (Edge){0, 2, 3};
    edges[2] = (Edge){1, 2, 1};
    edges[3] = (Edge){1, 3, 4};
    edges[4] = (Edge){2, 3, 1};

    kruskal(edges, V, E);

    return 0;
}
```

Exp 4:-
Dynamic Programming
a) Floyd Warshall Algorithm

```
#include <stdio.h>
#define INF 99999
#define V 4

void floydWarshall(int graph[V][V]) {
   int dist[V][V], i, j, k;

   for (i = 0; i < V; i++)
      for (j = 0; j < V; j++)
         dist[i][j] = graph[i][j];

   for (k = 0; k < V; k++)
      for (i = 0; i < V; i++)
         for (j = 0; j < V; j++)
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];

   printf("Shortest distances:\n");
   for (i = 0; i < V; i++) {
      for (j = 0; j < V; j++) {
         if (dist[i][j] == INF)
            printf("INF ");
         else
            printf("%3d ", dist[i][j]);
      }
      printf("\n");
   }
}

int main() {
   int graph[V][V] = {
      {0, 3, INF, 5},
      {2, 0, INF, 4},
      {INF, 1, 0, INF},
      {INF, INF, 2, 0}
   };
   floydWarshall(graph);
   return 0;
}
```

b) 0/1 Knapsack

```
#include <stdio.h>

int max(int a, int b) { return (a > b) ? a : b; }

int knapsack(int W, int wt[], int val[], int n) {
    int dp[n+1][W+1];
    for (int i = 0; i <= n; i++)
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (wt[i-1] <= w)
                dp[i][w] = max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]);
            else
                dp[i][w] = dp[i-1][w];
        }
    return dp[n][W];
}

int main() {
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("Max value = %d\n", knapsack(W, wt, val, n));
    return 0;
}
```

Experiment No 05.: Longest Common Subsequence (Used in TSP and DP)

```
#include <stdio.h>
#include <string.h>

int max(int a, int b) { return (a > b) ? a : b; }

int lcs(char *X, char *Y, int m, int n) {
   int L[m+1][n+1];
   for (int i = 0; i <= m; i++)
      for (int j = 0; j <= n; j++)
         if (i == 0 || j == 0)
            L[i][j] = 0;
         else if (X[i-1] == Y[j-1])
            L[i][j] = 1 + L[i-1][j-1];
         else
            L[i][j] = max(L[i-1][j], L[i][j-1]);
   return L[m][n];
}

int main() {
   char X[] = "AGGTAB";
   char Y[] = "GXTXAYB";
   printf("Length of LCS is %d\n", lcs(X, Y, strlen(X), strlen(Y)));
   return 0;
}
```

Experitment No 06:. Graph Algorithms (BFS and DFS)

```
#include <stdio.h>

int visited[100], queue[100], front = -1, rear = -1;

void bfs(int adj[10][10], int n, int start) {
    visited[start] = 1;
    queue[++rear] = start;
    while (front != rear) {
        int current = queue[++front];
        printf("%d ", current);
        for (int i = 0; i < n; i++) {
            if (adj[current][i] && !visited[i]) {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
}

int main() {
    int n = 4;
    int adj[10][10] = {
        {0, 1, 1, 0},
        {1, 0, 1, 1},
        {1, 1, 0, 0},
        {0, 1, 0, 0}
    };
    bfs(adj, n, 0);
    return 0;
}
```

DFS

```
#include <stdio.h>

int visited[10];

void dfs(int adj[10][10], int n, int start) {
    visited[start] = 1;
    printf("%d ", start);
    for (int i = 0; i < n; i++)
        if (adj[start][i] && !visited[i])
            dfs(adj, n, i);
}

int main() {
    int n = 4;
    int adj[10][10] = {
        {0, 1, 1, 0},
        {1, 0, 1, 1},
        {1, 1, 0, 0},
        {0, 1, 0, 0}
    };
    dfs(adj, n, 0);
    return 0;
}
```

Experiment No 07:. Backtracking

```
#include <stdio.h>
#define N 4

int board[N][N];

int isSafe(int row, int col) {
   for (int i = 0; i < col; i++)
      if (board[row][i]) return 0;
   for (int i=row, j=col; i>=0 && j>=0; i--, j--)
      if (board[i][j]) return 0;
   for (int i=row, j=col; i<N && j>=0; i++, j--)
      if (board[i][j]) return 0;
   return 1;
}

int solve(int col) {
   if (col >= N) return 1;
   for (int i = 0; i < N; i++) {
      if (isSafe(i, col)) {
         board[i][col] = 1;
         if (solve(col + 1)) return 1;
         board[i][col] = 0;
      }
   }
   return 0;
}

void printBoard() {
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++)
         printf("%d ", board[i][j]);
      printf("\n");
   }
}

int main() {
   if (solve(0))
      printBoard();
   else
      printf("Solution does not exist");
   return 0;
}
```

Experiment No 09:. String Matching

```
#include <stdio.h>
#include <string.h>

void search(char* txt, char* pat) {
    int n = strlen(txt);
    int m = strlen(pat);
    for (int i = 0; i <= n - m; i++) {
        int j;
        for (j = 0; j < m; j++)
            if (txt[i + j] != pat[j]) break;
        if (j == m) printf("Pattern found at index %d\n", i);
    }
}

int main() {
    char txt[] = "AABAACAADAABAABA";
    char pat[] = "AABA";
    search(txt, pat);
    return 0;
}
```

Experiment 10 : Min-Max Algorithm (used in game trees)

```
#include <stdio.h>
#include<math.h>
#define MAX 1000
#define MIN -1000

int minimax(int depth, int nodeIndex, int isMax, int scores[], int h) {
   if (depth == h)
      return scores[nodeIndex];
   if (isMax)
      return fmax(minimax(depth+1, nodeIndex*2, 0, scores, h),
               minimax(depth+1, nodeIndex*2 + 1, 0, scores, h));
   else
      return fmin(minimax(depth+1, nodeIndex*2, 1, scores, h),
               minimax(depth+1, nodeIndex*2 + 1, 1, scores, h));
}

int main() {
   int scores[] = {3, 5, 2, 9, 12, 5, 23, 23};
   int h = 3;
   printf("The optimal value is : %d\n", minimax(0, 0, 1, scores, h));
   return 0;
}
```

Experiment No 11: Job Sequencing with Deadlines

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    char id;
    int deadline, profit;
} Job;

int cmp(Job a, Job b) {
    return b.profit - a.profit;
}

void jobSequencing(Job jobs[], int n) {
    char result[n];
    int slot[n];
    for (int i = 0; i < n; i++) slot[i] = 0;

    for (int i = 0; i < n; i++) {
        for (int j = jobs[i].deadline - 1; j >= 0; j--) {
            if (!slot[j]) {
                result[j] = jobs[i].id;
                slot[j] = 1;
                break;
            }
        }
    }

    printf("Scheduled Jobs: ");
    for (int i = 0; i < n; i++)
        if (slot[i]) printf("%c ", result[i]);
}

int main() {
    Job jobs[] = {{'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27}, {'d', 1, 25}, {'e', 3, 15}};
    int n = sizeof(jobs)/sizeof(jobs[0]);
    jobSequencing(jobs, n);
    return 0;
}
```

Experiment No 12 :

```
#include <stdio.h>

#define V 5
#define E 8
#define INF 99999

typedef struct {
   int u, v, w;
} Edge;

Edge edges[E] = {
   {0, 1, -1}, {0, 2, 4}, {1, 2, 3}, {1, 3, 2},
   {1, 4, 2}, {3, 2, 5}, {3, 1, 1}, {4, 3, -3}
};

void bellmanFord(int src) {
   int dist[V];
   for (int i = 0; i < V; i++) dist[i] = INF;
   dist[src] = 0;

   for (int i = 1; i <= V - 1; i++)
      for (int j = 0; j < E; j++) {
         int u = edges[j].u;
         int v = edges[j].v;
         int w = edges[j].w;
         if (dist[u] != INF && dist[u] + w < dist[v])
            dist[v] = dist[u] + w;
      }

   printf("Vertex Distance from Source:\n");
   for (int i = 0; i < V; i++)
      printf("%d \t %d\n", i, dist[i]);
}

int main() {
   bellmanFord(0);
   return 0;
}
```