DEWAN VS INSTITUTE OF ENGINEERING AND TECHNOLOGY, MEERUT

(Affiliated to Dr APJ Abdul Kalam Technical University, Lucknow)

NH58Bypass, Partapur, Meerut, UP, India

(DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING)

DEWAN VS GROUP OF INSTITUTIONS INDIA

DAA Lab File

(BCS-553)

B.TECH–3rdYEAR (ODD SEM, 2025-2026)

Name	
RollNo.	
Section-Batch	

Submitted to
MS.

(Assistant Professor)

Index Page

Program Name	Date	Teacher's Signature
1. Linear Search		
2. Binary Search		
3. Quick Sort		
4. Merge Sort		
5. Heap Sort		
6. Knapsack Problem using Greedy Method		
7. Prim’s Algorithm		
8. Dijkstra’s Algorithm		
9. Floyd-Warshall Algorithm		
10. N-Queens Problem		

1. Write a program in C of Linear search by using recursion

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key, int idx) {
   if (idx >= n) return -1;
   if (arr[idx] == key) return idx;
   return linearSearch(arr, n, key, idx + 1);
}

int main() {
   int n;
   printf("Enter number of elements: ");
   if (scanf("%d", &n)!=1) return 0;
   int arr[n];
   printf("Enter %d integers:\n", n);
   for (int i=0;i<n;i++) scanf("%d", &arr[i]);
   int key;
   printf("Enter key to search: ");
   scanf("%d", &key);
   int pos = linearSearch(arr, n, key, 0);
   if (pos==-1) printf("Element not found.\n");
   else printf("Element found at index %d (0-based).\n", pos);
   return 0;
}
```

2. Write a Program in C to perform Binary Search for a given set of integer values recursively

```
#include <stdio.h>

int binarySearch(int arr[], int l, int r, int key) {
   if (l > r) return -1;
   int mid = l + (r - l) / 2;
   if (arr[mid] == key) return mid;
   else if (arr[mid] > key) return binarySearch(arr, l, mid - 1, key);
   else return binarySearch(arr, mid + 1, r, key);
}

int main() {
   int n;
   printf("Enter number of elements (sorted): ");
   if (scanf("%d", &n)!=1) return 0;
   int arr[n];
   printf("Enter %d sorted integers:\n", n);
   for (int i=0;i<n;i++) scanf("%d", &arr[i]);
   int key;
   printf("Enter key to search: ");
   scanf("%d", &key);
   int pos = binarySearch(arr, 0, n-1, key);
   if (pos==-1) printf("Element not found.\n");
   else printf("Element found at index %d (0-based).\n", pos);
   return 0;
}
```

3. Write a program in C to perform Quick Sort for the given list of integer values

```
#include <stdio.h>

void swap(int *a, int *b) { int t = *a; *a = *b; *b = t; }

int partition(int arr[], int low, int high) {
   int pivot = arr[high];
   int i = low - 1;
   for (int j = low; j <= high - 1; j++) {
      if (arr[j] <= pivot) {
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i+1], &arr[high]);
   return i+1;
}
void quickSort(int arr[], int low, int high) {
   if (low < high) {
      int pi = partition(arr, low, high);
      quickSort(arr, low, pi-1);
      quickSort(arr, pi+1, high);
   }
}
int main() {
   int n;
   printf("Enter number of elements: ");
   if (scanf("%d", &n)!=1) return 0;
   int arr[n];
   printf("Enter %d integers:\n", n);
   for (int i=0;i<n;i++) scanf("%d", &arr[i]);
   quickSort(arr, 0, n-1);
   printf("Sorted array:\n");
   for (int i=0;i<n;i++) printf("%d ", arr[i]);
   printf("\n");
   return 0;}
```

4. Write a program in C to perform Merge sort for any given list of numbers.

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r) {
   int n1 = m - l + 1;
   int n2 = r - m;
   int *L = malloc(n1 * sizeof(int));
   int *R = malloc(n2 * sizeof(int));
   for (int i = 0; i < n1; i++) L[i] = arr[l + i];
   for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];
   int i = 0, j = 0, k = l;
   while (i < n1 && j < n2) {
      if (L[i] <= R[j]) arr[k++] = L[i++];
      else arr[k++] = R[j++];
   }
   while (i < n1) arr[k++] = L[i++];
   while (j < n2) arr[k++] = R[j++];
   free(L); free(R);
}
void mergeSort(int arr[], int l, int r) {
   if (l < r) {
      int m = l + (r - l) / 2;
      mergeSort(arr, l, m);
      mergeSort(arr, m+1, r);
      merge(arr, l, m, r);}}
int main() {
   int n;
   printf("Enter number of elements: ");
   if (scanf("%d", &n)!=1) return 0;
   int arr[n];
   printf("Enter %d integers:\n", n);
   for (int i=0;i<n;i++) scanf("%d", &arr[i]);
   mergeSort(arr, 0, n-1);
   printf("Sorted array:\n");
   for (int i=0;i<n;i++) printf("%d ", arr[i])
   printf("\n");
   return 0;}
```

5. Write a program in C to perform Heap sort for any given list of numbers.

```
#include <stdio.h>

void swap(int *a, int *b) { int t = *a; *a = *b; *b = t; }

void heapify(int arr[], int n, int i) {
   int largest = i;
   int l = 2*i + 1;
   int r = 2*i + 2;
   if (l < n && arr[l] > arr[largest]) largest = l;
   if (r < n && arr[r] > arr[largest]) largest = r;
   if (largest != i) {
      swap(&arr[i], &arr[largest]);
      heapify(arr, n, largest);
   }
}

void heapSort(int arr[], int n) {
   for (int i = n/2 - 1; i >= 0; i--) heapify(arr, n, i);
   for (int i = n-1; i >= 0; i--) {
      swap(&arr[0], &arr[i]);
      heapify(arr, i, 0);
   }
}

int main() {
   int n;
   printf("Enter number of elements: ");
   if (scanf("%d", &n)!=1) return 0;
   int arr[n];
   printf("Enter %d integers:\n", n);
   for (int i=0;i<n;i++) scanf("%d", &arr[i]);
   heapSort(arr, n);
   printf("Sorted array:\n");
   for (int i=0;i<n;i++) printf("%d ", arr[i]);
   printf("\n");
   return 0;}
```

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

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

typedef struct {
   int id;
   double value;
   double weight;
   double ratio;
} Item;

int cmp(const void *a, const void *b) {
   double r1 = ((Item*)a)->ratio;
   double r2 = ((Item*)b)->ratio;
   if (r1 < r2) return 1;
   if (r1 > r2) return -1;
   return 0;
}

int main() {
   int n;
   printf("Enter number of items: ");
   if (scanf("%d", &n)!=1) return 0;
   Item items[n];
   for (int i=0;i<n;i++) {
      items[i].id = i+1;
      printf("Item %d value and weight: ", i+1);
      scanf("%lf %lf", &items[i].value, &items[i].weight);
      items[i].ratio = items[i].value / items[i].weight;
   }
   double capacity;
   printf("Enter knapsack capacity: ");
   scanf("%lf", &capacity);
   qsort(items, n, sizeof(Item), cmp);
   double totalValue = 0.0;
   double remaining = capacity;
```

```
    for (int i=0;i<n && remaining>0;i++) {
      if (items[i].weight <= remaining) {
         remaining -= items[i].weight;
          totalValue += items[i].value;
          printf("Taken whole item %d\n", items[i].id);
       } else {
          double fraction = remaining / items[i].weight;
          totalValue += items[i].value * fraction;
          printf("Taken %.2f fraction of item %d\n", fraction, items[i].id);
          remaining = 0;
       }
    }
    printf("Maximum value in knapsack = %.2f\n", totalValue);
    return 0;
}
```

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

```
#include  <stdio.h>
#include <limits.h>
#define V 100

int minKey(int key[], int mstSet[], int n) {
   int min = INT_MAX, min_index = -1;
   for (int v = 0; v < n; v++)
      if (mstSet[v] == 0 && key[v] < min) {
         min = key[v];
         min_index = v;
      }
   return min_index;
}

void primMST(int graph[][V], int n) {
   int parent[n];
   int key[n];
   int mstSet[n];
   for (int i = 0; i < n; i++) { key[i] = INT_MAX; mstSet[i] = 0; }
   key[0] = 0; parent[0] = -1;
   for (int count = 0; count < n-1; count++) {
      int u = minKey(key, mstSet, n);
      mstSet[u] = 1;
      for (int v = 0; v < n; v++)
         if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
            parent[v] = u;
            key[v] = graph[u][v];
         }
   }
   printf("Edge \tWeight\n");
   for (int i = 1; i < n; i++) printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}

int main() {
   int n;
   printf("Enter number of vertices: ");
```

```
    if (scanf("%d", &n)!=1) return 0;
    int graph[V][V];
    printf("Enter adjacency matrix (use 0 for no-edge):\n");
    for (int i=0;i<n;i++) for (int j=0;j<n;j++) scanf("%d", &graph[i][j]);
    primMST(graph, n);
    return 0;
}
```

8. Write a C program implementing Dijkstra's Algorithm

```
#include  <stdio.h>
#include <limits.h>
#define V 100

int minDistance(int dist[], int sptSet[], int n) {
   int min = INT_MAX, min_index = -1;
   for (int v = 0; v < n; v++)
      if (sptSet[v] == 0 && dist[v] <= min) {
         min = dist[v];
         min_index = v;
      }
   return min_index;
}

void dijkstra(int graph[][V], int src, int n) {
   int dist[n];
   int sptSet[n];
   for (int i=0;i<n;i++) { dist[i] = INT_MAX; sptSet[i]=0; }
   dist[src] = 0;
   for (int count=0; count<n-1; count++) {
      int u = minDistance(dist, sptSet, n);
      sptSet[u] = 1;
      for (int v=0; v<n; v++)
         if (!sptSet[v] && graph[u][v] && dist[u]!=INT_MAX && dist[u]+graph[u][v] <
dist[v])
            dist[v] = dist[u] + graph[u][v];
   }
   printf("Vertex Distance from Source\n");
   for (int i=0;i<n;i++) printf("%d \t %d\n", i, dist[i]);
}

int main() {
   int n;
   printf("Enter number of vertices: ");
   if (scanf("%d", &n)!=1) return 0;
   int graph[V][V];
```

```
    printf("Enter adjacency matrix (use 0 for no-edge):\n");
    for (int i=0;i<n;i++) for (int j=0;j<n;j++) scanf("%d", &graph[i][j]);
    int src;
    printf("Enter source vertex: ");
    scanf("%d", &src);
    dijkstra(graph, src, n);
    return 0;
}
```

9. Write a program for Floyd-Warshall Algorithm in C

```
#include  <stdio.h>
#include <limits.h>
#define V 100
#define INF 1000000000

void floydWarshall(int graph[][V], int n) {
   int dist[V][V];
   for (int i=0;i<n;i++)
      for (int j=0;j<n;j++)
         dist[i][j] = (graph[i][j]==0 && i!=j) ? INF : graph[i][j];

   for (int k=0;k<n;k++)
      for (int i=0;i<n;i++)
         for (int j=0;j<n;j++)
            if (dist[i][k] + dist[k][j] < dist[i][j])
               dist[i][j] = dist[i][k] + dist[k][j];
   printf("Shortest distances matrix:\n");
   for (int i=0;i<n;i++) {
      for (int j=0;j<n;j++) {
         if (dist[i][j] == INF) printf("INF ");
         else printf("%d ", dist[i][j]);
      }
      printf("\n");
   }
}
int main() {
   int n;
   printf("Enter number of vertices: ");
   if (scanf("%d", &n)!=1) return 0;
   int graph[V][V];
   printf("Enter adjacency matrix (use 0 for no-edge):\n");
   for (int i=0;i<n;i++) for (int j=0;j<n;j++) scanf("%d", &graph[i][j]);
   floydWarshall(graph, n);
   return 0;
}
```

10. Write a program in C to solve N-QUEENS problem.

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

int isSafe(int board[], int row, int col, int n) {
   for (int i=0;i<col;i++) {
      if (board[i]==row) return 0;
      if (abs(board[i]-row) == abs(i - col)) return 0;
   }
   return 1;
}

void printSolution(int board[], int n) {
   for (int i=0;i<n;i++) {
      for (int j=0;j<n;j++) {
         if (board[j] == i) printf(" Q ");
         else printf(" . ");
      }
      printf("\n");
   }
   printf("\n");
}

void solveNQUtil(int board[], int col, int n, int *count) {
   if (col == n) {
      (*count)++;
      printf("Solution %d:\n", *count);
      printSolution(board, n);
      return;
   }
   for (int i=0;i<n;i++) {
      if (isSafe(board, i, col, n)) {
         board[col] = i;
         solveNQUtil(board, col+1, n, count);
      }
   }
```

```
}

int main() {
    int n;
    printf("Enter value of N (e.g., 8): ");
    if (scanf("%d", &n)!=1) return 0;
    int *board = malloc(n * sizeof(int));
    int count = 0;
    solveNQUtil(board, 0, n, &count);
    if (count==0) printf("No solutions found.\n");
    free(board);
    return 0;
}
```