EXPERIMENT NO. 01

Title: Binary search techniques using array and recursion.

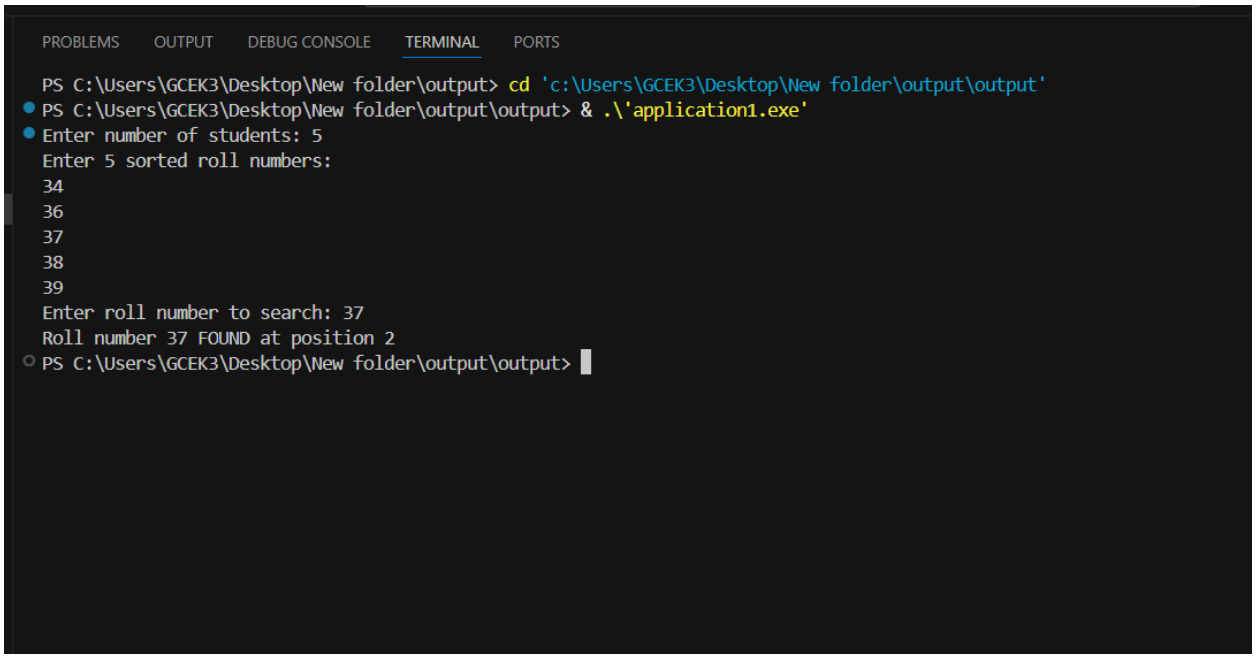
Application name: Student Record Search System using Recursive Binary Search

Program:

```
#include <stdio.h>

// Iterative Binary Search Function
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key)
            return mid;

        else if (key < arr[mid])
            high = mid - 1;

        else
            low = mid + 1;
    }

    return -1;  // Not found
}

int main() {
    int n, key, i;

    printf("Enter number of students: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d sorted roll numbers:\n", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
```

```
    printf("Enter roll number to search: ");
    scanf("%d", &key);

    int result = binarySearch(arr, n, key);

    if (result == -1)
        printf("Roll number %d NOT FOUND.\n", key);
    else
        printf("Roll number %d FOUND at position %d\n", key, result);

    return 0;
}
```

OUTPUT:



Application list:

1. Searching Student Records

Used to quickly find roll numbers, IDs, or marks from a sorted list.

2. Searching in Telephone/Contact Directories

Phone books or contact lists are sorted, so binary search retrieves entries fast.

3. Searching Product IDs in Online Shopping Sites

E-commerce websites store sorted product codes, enabling fast search.

4. Searching for Words in a Dictionary

Digital dictionaries use binary search to find words in sorted word lists.

5. Searching Book IDs in a Library Management System

Library catalogs store books in sorted order (ISBN), allowing fast lookup.

6. Searching Employee IDs in Companies

HR systems use binary search to find employee details from sorted employee IDs.

7. Auto-Suggestion Systems

Binary search helps to find the closest match in sorted suggestions.

8. Competitive Coding and Problem Solving

Binary search is widely used to optimize search-based problems.

9. In Computer Networks

Routing tables are often sorted; binary search helps in fast lookup.

10. Searching in Databases

Indexing in databases uses binary search to find records efficiently.

EXPERIMENT NO. 02

Title: Quick sort, merge sort using array as a data structure.

Application name: Student Marks Sorting System Using Quick Sort and Merge Sort.

Program:

```
#include <stdio.h>


// ---------------- QUICK SORT FUNCTIONS ----------------
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1, temp;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
```

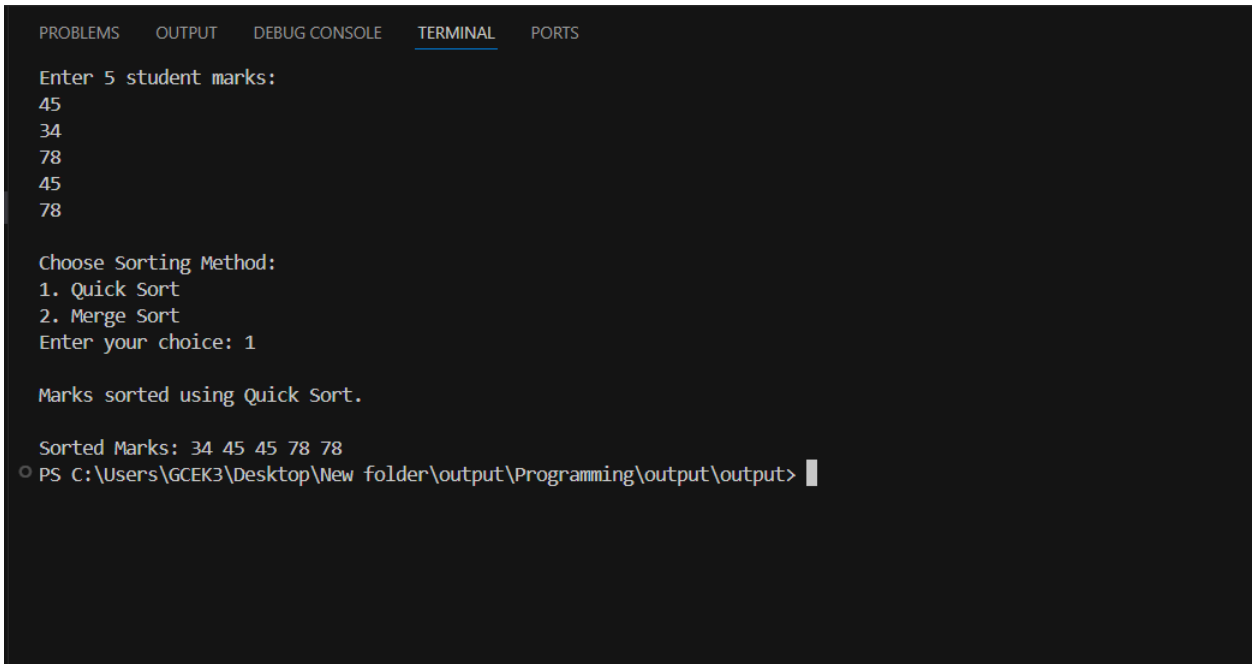
```
    return i + 1;
}

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);
    }
}

// ---------- MERGE SORT FUNCTIONS ----------
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
```

```
    int i = 0, j = 0, k = left;

    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++];
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

```
// ---------------- UTILITY ----------------
void display(int arr[], int n) {
    printf("\nSorted Marks: ");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main() {
    int n, choice;

    printf("Enter number of students: ");
    scanf("%d", &n);

    int marks[n];
    printf("Enter %d student marks:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &marks[i]);

    printf("\nChoose Sorting Method:\n");
    printf("1. Quick Sort\n");
    printf("2. Merge Sort\n");
    printf("Enter your choice: ");
```

```
    scanf("%d", &choice);

    if (choice == 1) {
        quickSort(marks, 0, n - 1);
        printf("\nMarks sorted using Quick Sort.\n");
    }
    else if (choice == 2) {
        mergeSort(marks, 0, n - 1);
        printf("\nMarks sorted using Merge Sort.\n");
    }
    else {
        printf("\nInvalid choice!\n");
        return 0;
    }
    display(marks, n);
    return 0;
}
```

OUTPUT:



Application list:

1. Sorting student marks
2. Sorting employee salaries
3. Sorting product prices
4. Sorting IDs
5. Sorting files/data entries

EXPERIMENT NO. 03

Title: Knapsack problem using Greedy method.

Application name: Optimal Resource Selection System.

Program:

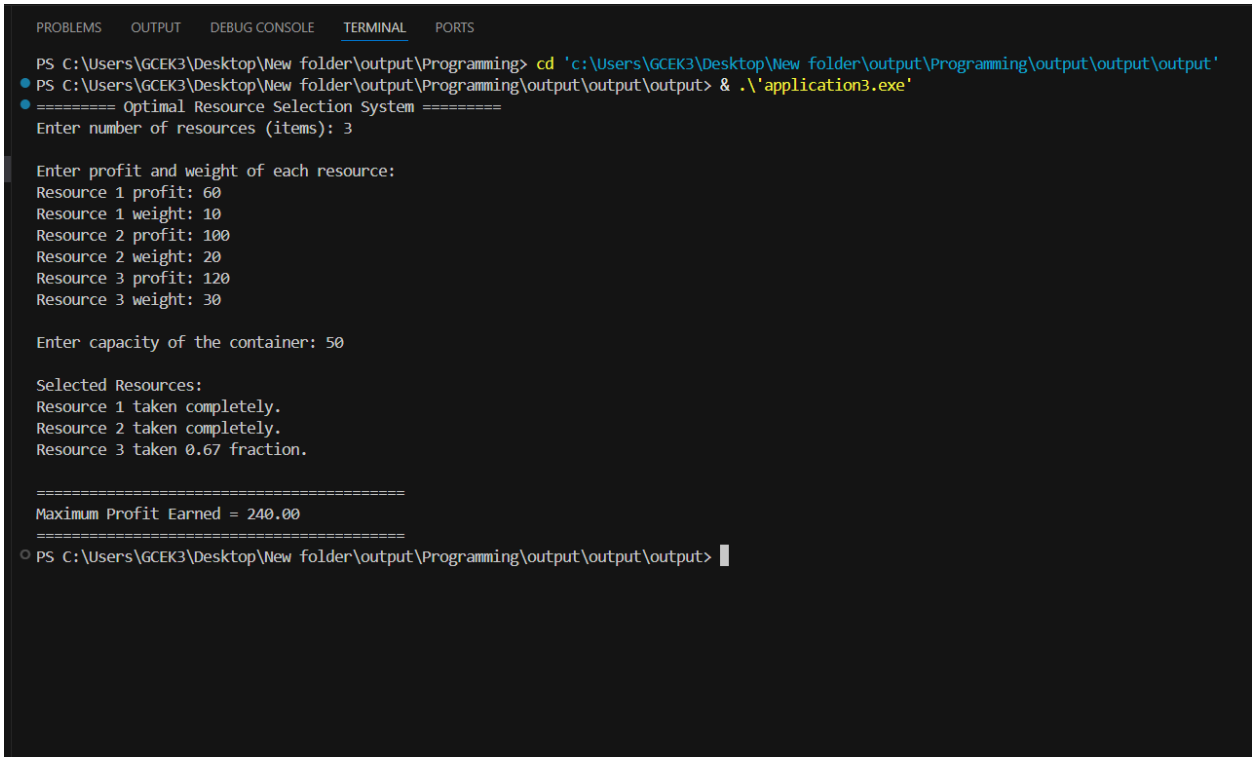
```
#include <stdio.h>


int main() {
    int n, i, j;
    float capacity;


    printf("========= Optimal Resource Selection System =========\n");


    printf("Enter number of resources (items): ");
    scanf("%d", &n);


    float profit[n], weight[n], ratio[n];


    // Input resources
    printf("\nEnter profit and weight of each resource:\n");
    for (i = 0; i < n; i++) {
        printf("Resource %d profit: ", i + 1);
        scanf("%f", &profit[i]);


        printf("Resource %d weight: ", i + 1);
        scanf("%f", &weight[i]);
```

```
    ratio[i] = profit[i] / weight[i];  // Greedy ratio
}

printf("\nEnter capacity of the container: ");
scanf("%f", &capacity);

// Sorting by highest ratio
for (i = 0; i < n - 1; i++) {
    for (j = i + 1; j < n; j++) {
        if (ratio[i] < ratio[j]) {
            float temp;

            temp = ratio[i];
            ratio[i] = ratio[j];
            ratio[j] = temp;

            temp = profit[i];
            profit[i] = profit[j];
            profit[j] = temp;

            temp = weight[i];
            weight[i] = weight[j];
            weight[j] = temp;
        }
```

```
    }
}

float totalProfit = 0.0f;

// Greedy selection
printf("\nSelected Resources:\n");
for (i = 0; i < n; i++) {

    if (capacity == 0)
        break;

    if (weight[i] <= capacity) {
        // Take full resource
        printf("Resource %d taken completely.\n", i + 1);
        totalProfit += profit[i];
        capacity -= weight[i];
    } else {
        // Take fractional resource
        float fraction = capacity / weight[i];
        printf("Resource %d taken %.2f fraction.\n", i + 1, fraction);
        totalProfit += profit[i] * fraction;
        capacity = 0;
    }
}
```

```
    printf("\n=========================================\n");
    printf("Maximum Profit Earned = %.2f\n", totalProfit);
    printf("=========================================\n");

    return 0;
}
```

OUTPUT:



Application list:

1. Optimal Resource Selection System
2. Maximum Profit Selection Application

3. Greedy-Based Knapsack Optimization System

4. Cargo Loading Optimization Applications

5. Smart Inventory Filling System

6. Profit Maximization Using Greedy Method

7. Fractional Knapsack Decision Support System

8. Efficient Storage and Profit Optimization App

9. Resource Allocation Using Greedy Algorithm

10. Greedy Knapsack Profit Calculator

EXPERIMENT NO. 04

Title: Greedy method to solve problems of Optimal Merge Pattern.

Application name: Document Merge Cost Optimization System

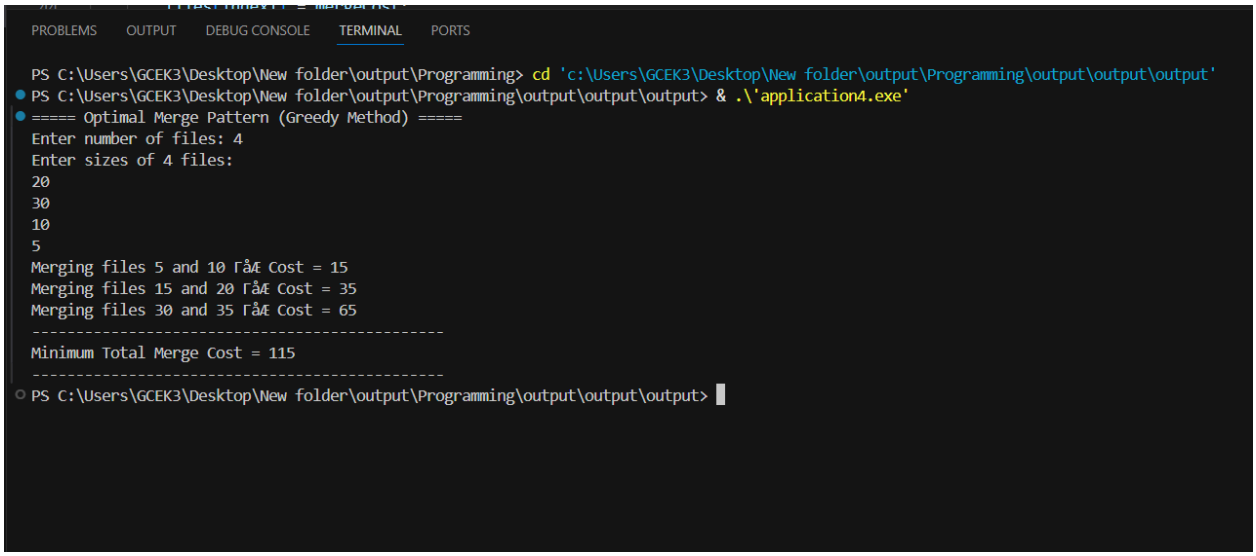
Program:

```
#include <stdio.h>


int main() {
    int n;


    printf("===== Optimal Merge Pattern (Greedy Method) =====\n");
    printf("Enter number of files: ");
    scanf("%d", &n);


    int files[n];


    printf("Enter sizes of %d files:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &files[i]);


    int totalCost = 0;


    // Repeat merging until only one file remains
    for (int k = 1; k < n; k++) {
```

```
int min1 = 999999, min2 = 999999;
int index1 = -1, index2 = -1;

// Find smallest two files
for (int i = 0; i < n; i++) {
    if (files[i] < min1 && files[i] != -1) {
        min2 = min1;
        index2 = index1;

        min1 = files[i];
        index1 = i;
    }
    else if (files[i] < min2 && files[i] != -1) {
        min2 = files[i];
        index2 = i;
    }
}

int mergeCost = min1 + min2;
totalCost += mergeCost;

printf("Merging files %d and %d → Cost = %d\n", min1, min2, mergeCost);
```

```
        // Replace first file with merged size and mark second as used
        files[index1] = mergeCost;
        files[index2] = -1;
    }

    printf("-----------------------------------------------\n");
    printf("Minimum Total Merge Cost = %d\n", totalCost);
    printf("-----------------------------------------------\n");

    return 0;
}
```

OUTPUT:



Application list:

1. File compression
2. Database merging
3. Combining sorted lists
4. Tape drive operations
5. Cloud storage merging
6. Log file merging
7. Disk scheduling
8. Data archival systems
9. Document zip merging
10. Network data consolidation

EXPERIMENT NO. 05

Title: Minimum cost spanning tree of a given undirected graph using Prim's algorithm and Kruskal's algorithm and compare.

Application name: Minimum Cost Network Construction System

Program:

```
#include <stdio.h>


#define INF 999999


// Structure for Kruskal
struct Edge {
   int u, v, w;
};


// Find operation
int find(int parent[], int i) {
   while (parent[i] != i)
      i = parent[i];
   return i;
}


// Union operation
void unionSet(int parent[], int a, int b) {
   int x = find(parent, a);
   int y = find(parent, b);
```

```
    parent[x] = y;
}

// -------------------- PRIM'S ALGORITHM --------------------
int prim(int graph[20][20], int n) {
    int selected[20] = {0};
    selected[0] = 1;

    int edges = 0, minCost = 0;

    printf("\n--- Minimum Spanning Tree using Prim's Algorithm ---\n");

    while (edges < n - 1) {
        int min = INF, x = -1, y = -1;

        for (int i = 0; i < n; i++) {
            if (selected[i]) {
                for (int j = 0; j < n; j++) {
                    if (!selected[j] && graph[i][j] && graph[i][j] < min) {
                        min = graph[i][j];
                        x = i;
                        y = j;
                    }
                }
            }
```

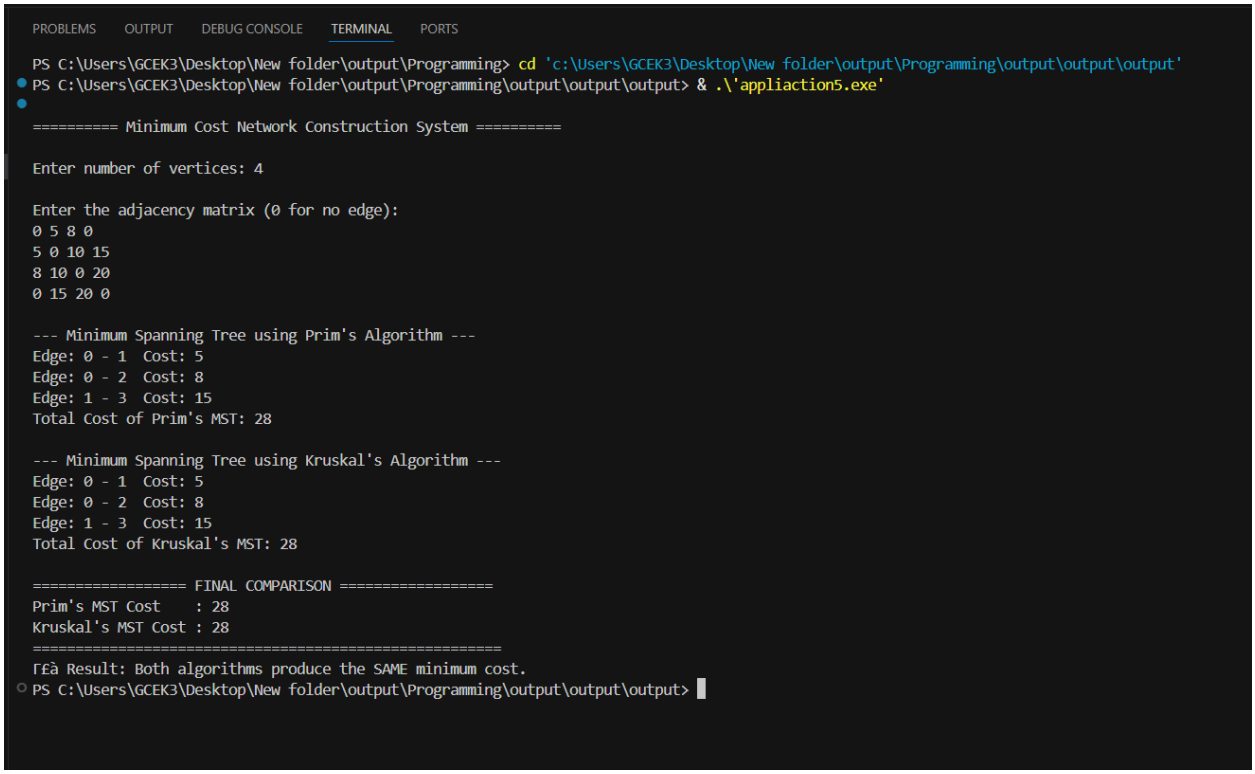
```
        }

        printf("Edge: %d - %d  Cost: %d\n", x, y, min);
        minCost += min;
        selected[y] = 1;
        edges++;
    }

    printf("Total Cost of Prim's MST: %d\n", minCost);
    return minCost;
}

// -------------------- KRUSKAL'S ALGORITHM --------------------
int kruskal(struct Edge edges[], int n, int e) {
    int parent[20];

    for (int i = 0; i < n; i++)
        parent[i] = i;

    // Bubble sort edges by weight
    for (int i = 0; i < e; i++) {
        for (int j = 0; j < e - i - 1; j++) {
            if (edges[j].w > edges[j + 1].w) {
                struct Edge temp = edges[j];
                edges[j] = edges[j+1];
```

```
            edges[j+1] = temp;
        }
    }
}

int minCost = 0, count = 0;

printf("\n--- Minimum Spanning Tree using Kruskal's Algorithm ---\n");

for (int i = 0; i < e && count < n - 1; i++) {
    int u = edges[i].u;
    int v = edges[i].v;

    if (find(parent, u) != find(parent, v)) {
        printf("Edge: %d - %d  Cost: %d\n", u, v, edges[i].w);
        minCost += edges[i].w;
        unionSet(parent, u, v);
        count++;
    }
}

printf("Total Cost of Kruskal's MST: %d\n", minCost);
return minCost;
}
```

```
// -------------------- MAIN APPLICATION --------------------
int main() {
    int n, graph[20][20];
    struct Edge edges[50];
    int e = 0;

    printf("\n========== Minimum Cost Network Construction System ==========\n");
    printf("\nEnter number of vertices: ");
    scanf("%d", &n);

    printf("\nEnter the adjacency matrix (0 for no edge):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);

            if (graph[i][j] != 0 && i < j) {
                edges[e].u = i;
                edges[e].v = j;
                edges[e].w = graph[i][j];
                e++;
            }
        }
    }

    int primCost = prim(graph, n);
```

```
    int kruskalCost = kruskal(edges, n, e);

    printf("\n================== FINAL COMPARISON
==================\n");
    printf("Prim's MST Cost    : %d\n", primCost);
    printf("Kruskal's MST Cost : %d\n", kruskalCost);

printf("=========================================================\n");

    if (primCost == kruskalCost)
        printf(“Result: Both algorithms produce the SAME minimum cost.\n");
    else
        printf("Result: MST costs differ due to graph structure.\n");

    return 0;
}
```

OUTPUT:



Application list:

1. Telephone line connection planning
2. Electrical grid optimization
3. Cost-effective road/railway network design
4. Water/irrigation pipelines
5. Computer networks (LAN cabling)
6. Network clustering
7. Transport and logistics route planning
8. Sensor network deployment
9. Power distribution systems
10. Fiber-optic path optimization

EXPERIMENT NO. 06

Title: Shortest paths to other vertices using Dijkstra's algorithm from a given vertex in a weighted connected graph.

Application name: Single Source Shortest Path Finder Using Dijkstra's Algorithm

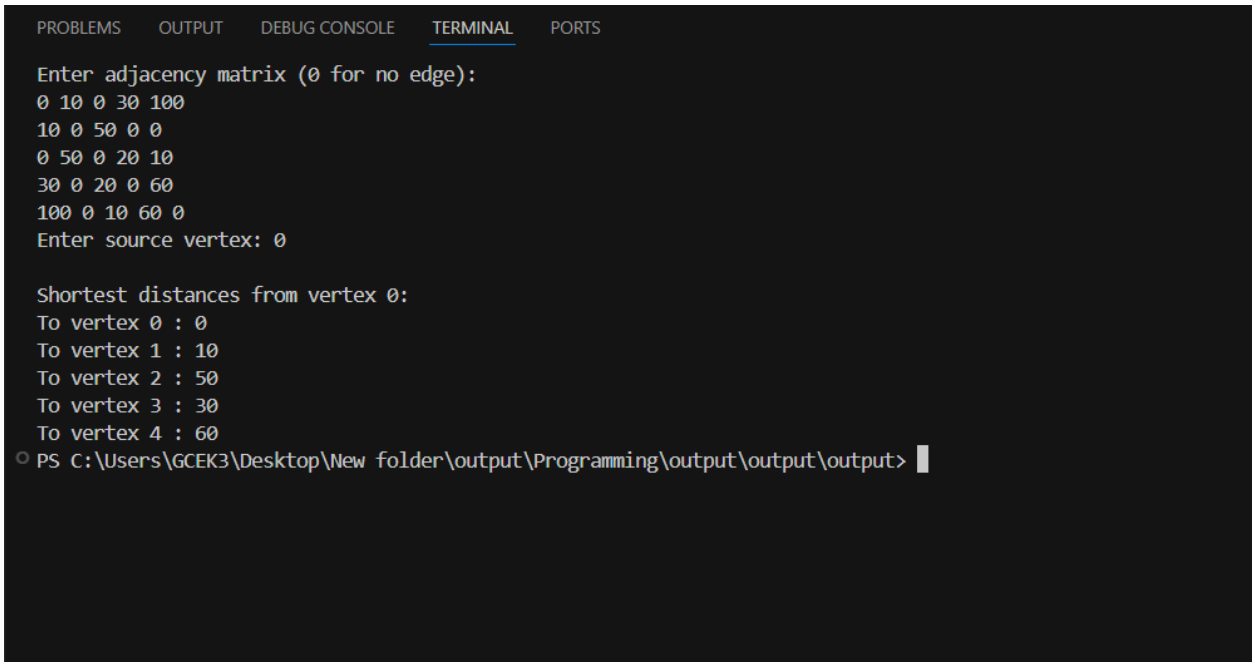
Program:

```
#include <stdio.h>


#define INF 99999


void dijkstra(int graph[20][20], int n, int src) {
   int dist[20], visited[20];


   // Initialize distances
   for (int i = 0; i < n; i++) {
      dist[i] = INF;
      visited[i] = 0;
   }


   dist[src] = 0;  // Distance of source = 0


   for (int c = 0; c < n - 1; c++) {
      int min = INF, u = -1;


      // Pick unvisited vertex with minimum distance
      for (int i = 0; i < n; i++) {
```

```
            if (!visited[i] && dist[i] < min) {
                min = dist[i];
                u = i;
            }
        }

        visited[u] = 1;

        // Update neighbors
        for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] &&
                dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Print result
    printf("\nShortest distances from vertex %d:\n", src);
    for (int i = 0; i < n; i++) {
        printf("To vertex %d : %d\n", i, dist[i]);
    }
}

int main() {
```

```
    int n, graph[20][20], src;

    printf("\n===== Single Source Shortest Path Application (Dijkstra) =====\n");

    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix (0 for no edge):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    printf("Enter source vertex: ");
    scanf("%d", &src);

    dijkstra(graph, n, src);

    return 0;
}
```

OUTPUT:



Application list:

1. GPS navigation
2. Network routing
3. Data packet transmission
4. Transport planning
5. Road maps
6. Robot path planning