```c
#include <stdio.h>

int linearSearch(int arr[], int size, int key, int index) {
    if (index >= size)
        return -1;

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

    return linearSearch(arr, size, key, index + 1);
}

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

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

    int arr[n];
    printf("Enter %d elements: ", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Enter element to search: ");
    scanf("%d", &key);

    result = linearSearch(arr, n, key, 0);

    if (result != -1)
        printf("Element found at index %d\n", result);
    else
        printf("Element not found\n");

    return 0;
}
```

```
>_pwsh  ⇒ Exp_01 - LInear and Binary Search (Recursive)  main ≡  2s 811ms          18,13:00
└─>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_01 - LInear and Binary Search (Recursive)\" ; gcc "LinearSe
arch.c" -o "LinearSearch.exe" ; if ($?) { .\"LinearSearch.exe" ; del "LinearSearch.exe" }
Enter number of elements: 5
Enter 5 elements: 89
67
18
33
45
Enter element to search: 18
Element found at index 2
```

```c
#include <stdio.h>

int binarySearch(int arr[], int left, int right, int x)
{
    if (right >= left)
    {
        int mid = left + (right - left) / 2;

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

        if (arr[mid] > x)
            return binarySearch(arr, left, mid - 1, x);

        return binarySearch(arr, mid + 1, right, x);
    }

    return -1;
}

int main()
{
    int arr[] = {2, 3, 4, 10, 40, 50, 70, 80};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x;

    printf("Enter element to search: ");
    scanf("%d", &x);

    int result = binarySearch(arr, 0, n - 1, x);

    if (result == -1)
        printf("Element is not present in array\n");
    else
        printf("Element is present at index %d\n", result);

    return 0;
}
```

```
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_01 - LInear and Binary Search (Recursive)\" ; gcc "binSearch.c" -o "binSearch.exe" ; if ($?) { .\"binSearch.exe" ; del "binSearch.exe" }
Enter element to search: 10
Element is present at index 3
```

```c
#include <stdio.h>

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

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    // Build max heap
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // Extract elements from heap
    for (int i = n - 1; i > 0; i--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}

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

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter elements: ");
    for (int i = 0; i < n; i++)
```

```c
        scanf("%d", &arr[i]);

    printf("Original array: ");
    printArray(arr, n);

    heapSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

```
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_02_Heap_Sort\" ; gcc "Heapsort.c" -o "Heapsort.exe" ; if ($
?) { .\"Heapsort.exe" ; del "Heapsort.exe" }
Enter number of elements: 7
Enter elements: 56
35
78
91
46
100
78
Original array: 56 35 78 91 46 100 78
Sorted array: 35 46 56 78 78 91 100
```

```c
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void merge(int arr[], int left, int mid, int right) {
 5      int n1 = mid - left + 1;
 6      int n2 = right - mid;
 7
 8      int *L = (int*)malloc(n1 * sizeof(int));
 9      int *R = (int*)malloc(n2 * sizeof(int));
10
11      for (int i = 0; i < n1; i++)
12          L[i] = arr[left + i];
13      for (int j = 0; j < n2; j++)
14          R[j] = arr[mid + 1 + j];
15
16      int i = 0, j = 0, k = left;
17
18      while (i < n1 && j < n2) {
19          if (L[i] <= R[j]) {
20              arr[k] = L[i];
21              i++;
22          } else {
23              arr[k] = R[j];
24              j++;
25          }
26          k++;
27      }
28
29      while (i < n1) {
30          arr[k] = L[i];
31          i++;
32          k++;
33      }
34
35      while (j < n2) {
36          arr[k] = R[j];
37          j++;
38          k++;
39      }
40
41      free(L);
42      free(R);
43  }
44
45  void mergeSort(int arr[], int left, int right) {
46      if (left < right) {
47          int mid = left + (right - left) / 2;
48
49          mergeSort(arr, left, mid);
50          mergeSort(arr, mid + 1, right);
51          merge(arr, left, mid, right);
```

```c
        }
}

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

int main() {
    int n;

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

    int *arr = (int*)malloc(n * sizeof(int));

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

    printf("Original array: ");
    printArray(arr, n);

    mergeSort(arr, 0, n - 1);

    printf("Sorted array: ");
    printArray(arr, n);

    free(arr);
    return 0;
}
```

```
pwsh    Exp_02_Heap_Sort    main ≡    1ms    18,13:03

>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_03_Merge_Sort\" ; gcc "MergeSort.c" -o "MergeSort.exe" ; if
($?) { .\"MergeSort.exe" ; del "MergeSort.exe" }
Enter number of elements: 7
Enter elements: 45
32
15
87
694
12
35
Original array: 45 32 15 87 694 12 35
Sorted array: 12 15 32 35 45 87 694
```

```c
#include <stdio.h>

void selectionSort(int arr[], int n) {
    int i, j, minIdx, temp;

    for (i = 0; i < n - 1; i++) {
        minIdx = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIdx]) {
                minIdx = j;
            }
        }
        if (minIdx != i) {
            temp = arr[i];
            arr[i] = arr[minIdx];
            arr[minIdx] = temp;
        }
    }
}

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

int main() {
    int n;

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

    int arr[n];
    printf("Enter elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Original array: ");
    printArray(arr, n);

    selectionSort(arr, n);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

```
>_pwsh    Exp_03_Merge_Sort    main ≡    1ms                                    18,13:04

>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_04_Selection Sort\" ; gcc "selectionsort.c" -o "selectionso
rt.exe" ; if ($?) { .\"selectionsort.exe" ; del "selectionsort.exe" }
Enter number of elements: 5
Enter elements: 241
254
78
45
87
Original array: 241 254 78 45 87
Sorted array: 45 78 87 241 254
```

```c
 1  #include <stdio.h>
 2
 3  void insertionSort(int arr[], int n) {
 4      for (int i = 1; i < n; i++) {
 5          int key = arr[i];
 6          int j = i - 1;
 7
 8          while (j >= 0 && arr[j] > key) {
 9              arr[j + 1] = arr[j];
10              j--;
11          }
12          arr[j + 1] = key;
13      }
14  }
15
16  void printArray(int arr[], int n) {
17      for (int i = 0; i < n; i++) {
18          printf("%d ", arr[i]);
19      }
20      printf("\n");
21  }
22
23  int main() {
24      int n;
25
26      printf("Enter number of elements: ");
27      scanf("%d", &n);
28
29      int arr[n];
30      printf("Enter %d elements: ", n);
31      for (int i = 0; i < n; i++) {
32          scanf("%d", &arr[i]);
33      }
34
35      printf("Original array: ");
36      printArray(arr, n);
37
38      insertionSort(arr, n);
39
40      printf("Sorted array: ");
41      printArray(arr, n);
42
43      return 0;
44  }
```

```
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_05_Insertion_Sort\" ; gcc "InsertionSort.c" -o "InsertionSort.exe" ; if ($?) { .\"InsertionSort.exe" ; del "InsertionSort.exe" }
Enter number of elements: 6
Enter 6 elements: 74
424
78
45
85
23
Original array: 74 424 78 45 85 23
Sorted array: 23 45 74 78 85 424
```

```c
 1  #include <stdio.h>
 2
 3  void swap(int* a, int* b) {
 4      int temp = *a;
 5      *a = *b;
 6      *b = temp;
 7  }
 8
 9  int partition(int arr[], int low, int high) {
10      int pivot = arr[high];
11      int i = low - 1;
12
13      for (int j = low; j < high; j++) {
14          if (arr[j] < pivot) {
15              i++;
16              swap(&arr[i], &arr[j]);
17          }
18      }
19      swap(&arr[i + 1], &arr[high]);
20      return i + 1;
21  }
22
23  void quickSort(int arr[], int low, int high) {
24      if (low < high) {
25          int pi = partition(arr, low, high);
26          quickSort(arr, low, pi - 1);
27          quickSort(arr, pi + 1, high);
28      }
29  }
30
31  void printArray(int arr[], int size) {
32      for (int i = 0; i < size; i++) {
33          printf("%d ", arr[i]);
34      }
35      printf("\n");
36  }
37
38  int main() {
39      int n;
40
41      printf("Enter number of elements: ");
42      scanf("%d", &n);
43
44      int arr[n];
45      printf("Enter %d elements: ", n);
46      for (int i = 0; i < n; i++) {
47          scanf("%d", &arr[i]);
48      }
49
50      printf("Original array: ");
51      printArray(arr, n);
```

```c
    quickSort(arr, 0, n - 1);

    printf("Sorted array: ");
    printArray(arr, n);

    return 0;
}
```

```
pwsh    Exp_05_Insertion_Sort    main ≡    Oms    18,13:05
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_06_Quick_Sort\" ; gcc "QuickSort.c" -o "QuickSort.exe" ; if
($?) { .\"QuickSort.exe" ; del "QuickSort.exe" }
Enter number of elements: 5
Enter 5 elements: 152
466
458
244
378
Original array: 152 466 458 244 378
Sorted array: 152 244 378 458 466
```

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5      int weight;
6      int value;
7      double ratio;
8      int index;
9  } Item;
10
11 int compare(const void *a, const void *b) {
12     Item *itemA = (Item *)a;
13     Item *itemB = (Item *)b;
14     if (itemB->ratio > itemA->ratio) return 1;
15     if (itemB->ratio < itemA->ratio) return -1;
16     return 0;
17 }
18
19 void knapsackGreedy(int weights[], int values[], int n, int capacity) {
20     Item items[n];
21
22     // Calculate value-to-weight ratio
23     for (int i = 0; i < n; i++) {
24         items[i].weight = weights[i];
25         items[i].value = values[i];
26         items[i].ratio = (double)values[i] / weights[i];
27         items[i].index = i + 1;
28     }
29
30     // Sort by ratio in descending order
31     qsort(items, n, sizeof(Item), compare);
32
33     double totalValue = 0.0;
34     int remainingCapacity = capacity;
35     double fractions[n];
36
37     for (int i = 0; i < n; i++) {
38         fractions[i] = 0.0;
39     }
40
41     printf("\nItem\tWeight\tValue\tRatio\t\tFraction\n");
42     printf("-----------------------------------------------------------\n");
43
44     for (int i = 0; i < n; i++) {
45         if (remainingCapacity >= items[i].weight) {
46             // Take full item
47             fractions[items[i].index - 1] = 1.0;
48             totalValue += items[i].value;
49             remainingCapacity -= items[i].weight;
50             printf("%d\t%d\t%d\t%.2f\t\t%.2f\n", items[i].index, items[i].weight,
51                     items[i].value, items[i].ratio, 1.0);
```

```c
        } else if (remainingCapacity > 0) {
            // Take fractional item
            double fraction = (double)remainingCapacity / items[i].weight;
            fractions[items[i].index - 1] = fraction;
            totalValue += items[i].value * fraction;
            printf("%d\t%d\t%d\t%.2f\t\t%.2f\n", items[i].index, items[i].weight,
                    items[i].value, items[i].ratio, fraction);
            remainingCapacity = 0;
            break;
        }
    }

    printf("-----------------------------------------------------------\n");
    printf("Maximum value: %.2f\n", totalValue);
}

int main() {
    int n, capacity;

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

    int weights[n], values[n];

    printf("Enter weights: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &weights[i]);
    }

    printf("Enter values: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &values[i]);
    }

    printf("Enter knapsack capacity: ");
    scanf("%d", &capacity);

    knapsackGreedy(weights, values, n, capacity);

    return 0;
}
```

```
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_07\" ; gc
($?) { .\"KnapsackGreedy.exe" ; del "KnapsackGreedy.exe" }
```

```
4
4
4
7
8
10
Enter values: 1
2
1
6
5
5
5
10
12
15
Enter knapsack capacity: 20

Item    Weight  Value   Ratio           Fraction
------------------------------------------------------------
9       8       12      1.50            1.00
10      10      15      1.50            1.00
8       7       10      1.43            0.29
------------------------------------------------------------
Maximum value: 29.86
```

```c
1  #include <stdio.h>
2  #include <limits.h>
3  #include <stdbool.h>
4
5  #define MAX 20
6
7  int n;
8  int graph[MAX][MAX];
9  int dp[MAX][1 << MAX];
10
11 int min(int a, int b) {
12     return (a < b) ? a : b;
13 }
14
15 int tsp(int pos, int mask) {
16     if (mask == (1 << n) - 1) {
17         return graph[pos][0];
18     }
19
20     if (dp[pos][mask] != -1) {
21         return dp[pos][mask];
22     }
23
24     int ans = INT_MAX;
25
26     for (int city = 0; city < n; city++) {
27         if ((mask & (1 << city)) == 0) {
28             int newAns = graph[pos][city] + tsp(city, mask | (1 << city));
29             ans = min(ans, newAns);
30         }
31     }
32
33     return dp[pos][mask] = ans;
34 }
35
36 int main() {
37     printf("Enter number of cities: ");
38     scanf("%d", &n);
39
40     printf("Enter cost matrix:\n");
41     for (int i = 0; i < n; i++) {
42         for (int j = 0; j < n; j++) {
43             scanf("%d", &graph[i][j]);
44         }
45     }
46
47     for (int i = 0; i < MAX; i++) {
48         for (int j = 0; j < (1 << MAX); j++) {
49             dp[i][j] = -1;
50         }
51     }
```

```c
    int result = tsp(0, 1);

    printf("Minimum cost: %d\n", result);

    return 0;
}
```

```
>_pwsh    Exp_07    main ≡    7ms
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_08\" ; gcc
e" ; if ($?) { .\"TravellingSalesman.exe" ; del "TravellingSalesman.exe" }
Enter number of cities: 7
Enter cost matrix:
0 2 9 10 7 3 8
1 0 6 4 3 9 2
15 7 0 8 3 6 4
6 3 12 0 11 5 9
10 4 8 9 0 2 6
9 7 5 6 3 0 4
8 6 3 7 5 4 0
Minimum cost: 24
```

```c
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  typedef struct {
 5      int src, dest, weight;
 6  } Edge;
 7
 8  typedef struct {
 9      int parent, rank;
10  } Subset;
11
12  int find(Subset subsets[], int i) {
13      if (subsets[i].parent != i)
14          subsets[i].parent = find(subsets, subsets[i].parent);
15      return subsets[i].parent;
16  }
17
18  void unionSets(Subset subsets[], int x, int y) {
19      int xroot = find(subsets, x);
20      int yroot = find(subsets, y);
21
22      if (subsets[xroot].rank < subsets[yroot].rank)
23          subsets[xroot].parent = yroot;
24      else if (subsets[xroot].rank > subsets[yroot].rank)
25          subsets[yroot].parent = xroot;
26      else {
27          subsets[yroot].parent = xroot;
28          subsets[xroot].rank++;
29      }
30  }
31
32  int compareEdges(const void* a, const void* b) {
33      return ((Edge*)a)->weight - ((Edge*)b)->weight;
34  }
35
36  void kruskalMST(Edge edges[], int V, int E) {
37      qsort(edges, E, sizeof(Edge), compareEdges);
38
39      Subset* subsets = (Subset*)malloc(V * sizeof(Subset));
40      for (int i = 0; i < V; i++) {
41          subsets[i].parent = i;
42          subsets[i].rank = 0;
43      }
44
45      Edge result[V];
46      int e = 0, i = 0;
47
48      printf("Edges in MST:\n");
49      while (e < V - 1 && i < E) {
50          Edge next_edge = edges[i++];
51          int x = find(subsets, next_edge.src);
```

```c
            int y = find(subsets, next_edge.dest);

            if (x != y) {
                result[e++] = next_edge;
                printf("%d -- %d == %d\n", next_edge.src, next_edge.dest,
    next_edge.weight);
                unionSets(subsets, x, y);
            }
        }

    int totalWeight = 0;
    for (int i = 0; i < e; i++)
        totalWeight += result[i].weight;
    printf("Total weight of MST: %d\n", totalWeight);

    free(subsets);
}

int main() {
    int V, E;
    printf("Enter number of vertices: ");
    scanf("%d", &V);
    printf("Enter number of edges: ");
    scanf("%d", &E);

    Edge* edges = (Edge*)malloc(E * sizeof(Edge));
    printf("Enter edges (src dest weight):\n");
    for (int i = 0; i < E; i++) {
        scanf("%d %d %d", &edges[i].src, &edges[i].dest, &edges[i].weight);
    }

    kruskalMST(edges, V, E);

    free(edges);
    return 0;
}
```

```
>_pwsh  ⮕ Exp_08  main ≡ ?2  2ms
 >> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_09\"
($?) { .\"minSpaningTree.exe" ; del "minSpaningTree.exe" }
Enter number of vertices: 4
Enter number of edges: 5
Enter edges (src dest weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in MST:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total weight of MST: 19
```

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <stdbool.h>
4
5   bool isSafe(int** board, int row, int col, int n) {
6       for (int i = 0; i < col; i++)
7           if (board[row][i])
8               return false;
9
10      for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
11          if (board[i][j])
12              return false;
13
14      for (int i = row, j = col; i < n && j >= 0; i++, j--)
15          if (board[i][j])
16              return false;
17
18      return true;
19  }
20
21  bool solveNQueenUtil(int** board, int col, int n) {
22      if (col >= n)
23          return true;
24
25      for (int i = 0; i < n; i++) {
26          if (isSafe(board, i, col, n)) {
27              board[i][col] = 1;
28
29              if (solveNQueenUtil(board, col + 1, n))
30                  return true;
31
32              board[i][col] = 0;
33          }
34      }
35
36      return false;
37  }
38
39  void printSolution(int** board, int n) {
40      for (int i = 0; i < n; i++) {
41          for (int j = 0; j < n; j++)
42              printf("%d ", board[i][j]);
43          printf("\n");
44      }
45  }
46
47  bool solveNQueen(int n) {
48      int** board = (int**)malloc(n * sizeof(int*));
49      for (int i = 0; i < n; i++) {
50          board[i] = (int*)calloc(n, sizeof(int));
51      }
```

```c
    if (!solveNQueenUtil(board, 0, n)) {
        printf("Solution does not exist\n");
        for (int i = 0; i < n; i++)
            free(board[i]);
        free(board);
        return false;
    }

    printSolution(board, n);

    for (int i = 0; i < n; i++)
        free(board[i]);
    free(board);
    return true;
}

int main() {
    int n;
    printf("Enter the value of N: ");
    scanf("%d", &n);

    solveNQueen(n);

    return 0;
}
```

```
>_pwsh    ⮞ Exp_09    () main ≡ ☑ ?2    ⊞ 2ms
└──>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_10\"
een.exe" ; del "N_Queen.exe" }
Enter the value of N: 4
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
   >_pwsh    ⮞ Exp_10    () main ≡ ☑ ?2    ⊞ 43s 811ms
└──>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_10\"
een.exe" ; del "N_Queen.exe" }
Enter the value of N: 3
Solution does not exist
```

```c
1  #include <stdio.h>
2
3  int max(int a, int b) {
4      return (a > b) ? a : b;
5  }
6
7  int knapsack(int W, int wt[], int val[], int n) {
8      int dp[n + 1][W + 1];
9
10     for (int i = 0; i <= n; i++) {
11         for (int w = 0; w <= W; w++) {
12             if (i == 0 || w == 0)
13                 dp[i][w] = 0;
14             else if (wt[i - 1] <= w)
15                 dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1]
   [w]);
16             else
17                 dp[i][w] = dp[i - 1][w];
18         }
19     }
20
21     return dp[n][W];
22 }
23
24 int main() {
25     int n, W;
26
27     printf("Enter number of items: ");
28     scanf("%d", &n);
29
30     int val[n], wt[n];
31
32     printf("Enter values: ");
33     for (int i = 0; i < n; i++)
34         scanf("%d", &val[i]);
35
36     printf("Enter weights: ");
37     for (int i = 0; i < n; i++)
38         scanf("%d", &wt[i]);
39
40     printf("Enter knapsack capacity: ");
41     scanf("%d", &W);
42
43     int maxValue = knapsack(W, wt, val, n);
44     printf("Maximum value: %d\n", maxValue);
45
46     return 0;
47 }
```

```
>_pwsh    ⬛➡ Exp_10    🐙 main ≡ 📝?2    🔳 1ms
└─> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algo
 .\"0_1Knapsack.exe" ; del "0_1Knapsack.exe" }
Enter number of items: 3
Enter values: 60 100 120
Enter weights: 10 20 30
Enter knapsack capacity: 50
Maximum value: 220
```

```c
 1  #include <stdio.h>
 2  #include <limits.h>
 3  #include <stdbool.h>
 4
 5  #define MAX 100
 6  #define INF INT_MAX
 7
 8  int minDistance(int dist[], bool visited[], int n) {
 9      int min = INF, minIndex;
10      for (int i = 0; i < n; i++) {
11          if (!visited[i] && dist[i] < min) {
12              min = dist[i];
13              minIndex = i;
14          }
15      }
16      return minIndex;
17  }
18
19  void dijkstra(int graph[MAX][MAX], int n, int src) {
20      int dist[MAX];
21      bool visited[MAX];
22
23      for (int i = 0; i < n; i++) {
24          dist[i] = INF;
25          visited[i] = false;
26      }
27
28      dist[src] = 0;
29
30      for (int count = 0; count < n - 1; count++) {
31          int u = minDistance(dist, visited, n);
32          visited[u] = true;
33
34          for (int v = 0; v < n; v++) {
35              if (!visited[v] && graph[u][v] && dist[u] != INF &&
36                  dist[u] + graph[u][v] < dist[v]) {
37                  dist[v] = dist[u] + graph[u][v];
38              }
39          }
40      }
41
42      printf("Vertex\t\tDistance from Source (%d)\n", src);
43      for (int i = 0; i < n; i++) {
44          printf("%d\t\t", i);
45          if (dist[i] == INF)
46              printf("INF\n");
47          else
48              printf("%d\n", dist[i]);
49      }
50  }
51
```

```c
int main() {
    int n, src;
    int graph[MAX][MAX];

    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;
}
```

```
>_pwsh    Exp_11    main ≡  ?2    1ms
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithm
Dijkastra.exe" ; del "Dijkastra.exe" }
Enter number of vertices: 5
Enter adjacency matrix (0 for no edge):
0 10 0 30 100
0 0 50 0 0
0 0 0 0 10
0 0 20 0 60
0 0 0 0 0
Enter source vertex: 0
Vertex              Distance from Source (0)
0                   0
1                   10
2                   50
3                   30
4                   60
```

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Structure to represent an item
5  typedef struct {
6      int weight;
7      int value;
8      double ratio; // value-to-weight ratio for greedy approach
9  } Item;
10
11 // Function to find maximum of two integers
12 int max(int a, int b) {
13     return (a > b) ? a : b;
14 }
15
16 // Dynamic Programming approach for 0/1 Knapsack
17 int knapsackDP(int weights[], int values[], int n, int capacity) {
18     int dp[n + 1][capacity + 1];
19
20     // Build table dp[][] in bottom-up manner
21     for (int i = 0; i <= n; i++) {
22         for (int w = 0; w <= capacity; w++) {
23             if (i == 0 || w == 0)
24                 dp[i][w] = 0;
25             else if (weights[i - 1] <= w)
26                 dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]],
27                                dp[i - 1][w]);
28             else
29                 dp[i][w] = dp[i - 1][w];
30         }
31     }
32
33     return dp[n][capacity];
34 }
35
36 // Comparison function for qsort (descending order of ratio)
37 int compare(const void *a, const void *b) {
38     Item *item1 = (Item *)a;
39     Item *item2 = (Item *)b;
40     if (item2->ratio > item1->ratio)
41         return 1;
42     else if (item2->ratio < item1->ratio)
43         return -1;
44     return 0;
45 }
46
47 // Greedy approach for 0/1 Knapsack (approximation)
48 int knapsackGreedy(int weights[], int values[], int n, int capacity) {
49     Item items[n];
50
51     // Create items array with value-to-weight ratio
```

```c
    for (int i = 0; i < n; i++) {
        items[i].weight = weights[i];
        items[i].value = values[i];
        items[i].ratio = (double)values[i] / weights[i];
    }

    // Sort items by ratio in descending order
    qsort(items, n, sizeof(Item), compare);

    int totalValue = 0;
    int currentWeight = 0;

    // Pick items based on greedy choice
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            totalValue += items[i].value;
        }
    }

    return totalValue;
}

int main() {
    int n, capacity;

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

    int weights[n], values[n];

    printf("Enter the weights of items:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d", &weights[i]);
    }

    printf("Enter the values of items:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d", &values[i]);
    }

    printf("Enter the knapsack capacity: ");
    scanf("%d", &capacity);

    // Dynamic Programming approach
    int dpResult = knapsackDP(weights, values, n, capacity);
    printf("\n--- Dynamic Programming Approach ---\n");
    printf("Maximum value (DP): %d\n", dpResult);

    // Greedy approach
    int greedyResult = knapsackGreedy(weights, values, n, capacity);
    printf("\n--- Greedy Approach ---\n");
```

```c
    printf("Maximum value (Greedy): %d\n", greedyResult);

    printf("\nNote: Greedy approach may not always give optimal solution for 0/1
Knapsack.\n");

    return 0;
}
```

```
 >_pwsh    Exp_12    main ≡ ?2    1ms
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithm
edy.exe" ; if ($?) { .\"0_1Knapsack_DP_Greedy.exe" ; del "
Enter the number of items: 3
Enter the weights of items:
Item 1: 10
Item 2: 20
Item 3: 30
Enter the values of items:
Item 1: 60
Item 2: 100
Item 3: 120
Enter the knapsack capacity: 50

--- Dynamic Programming Approach ---
Maximum value (DP): 220

--- Greedy Approach ---
Maximum value (Greedy): 160
```

```c
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define MAX 30
5
6   typedef struct edge {
7       int u, v, w;
8   } edge;
9
10  typedef struct edge_list {
11      edge data[MAX];
12      int n;
13  } edge_list;
14
15  edge_list elist;
16  int Graph[MAX][MAX], n;
17  edge_list spanlist;
18
19  void kruskal();
20  int find(int belongs[], int vertexno);
21  void union1(int belongs[], int c1, int c2);
22  void sort();
23  void print();
24
25  void kruskal() {
26      int belongs[MAX], i, j, cno1, cno2;
27      elist.n = 0;
28
29      for (i = 1; i < n; i++)
30          for (j = 0; j < i; j++) {
31              if (Graph[i][j] != 0) {
32                  elist.data[elist.n].u = i;
33                  elist.data[elist.n].v = j;
34                  elist.data[elist.n].w = Graph[i][j];
35                  elist.n++;
36              }
37          }
38
39      sort();
40
41      for (i = 0; i < n; i++)
42          belongs[i] = i;
43
44      spanlist.n = 0;
45
46      for (i = 0; i < elist.n; i++) {
47          cno1 = find(belongs, elist.data[i].u);
48          cno2 = find(belongs, elist.data[i].v);
49
50          if (cno1 != cno2) {
51              spanlist.data[spanlist.n] = elist.data[i];
```

```c
                spanlist.n++;
                union1(belongs, cno1, cno2);
            }
        }
}

int find(int belongs[], int vertexno) {
    return belongs[vertexno];
}

void union1(int belongs[], int c1, int c2) {
    int i;
    for (i = 0; i < n; i++)
        if (belongs[i] == c2)
            belongs[i] = c1;
}

void sort() {
    int i, j;
    edge temp;

    for (i = 1; i < elist.n; i++)
        for (j = 0; j < elist.n - 1; j++)
            if (elist.data[j].w > elist.data[j + 1].w) {
                temp = elist.data[j];
                elist.data[j] = elist.data[j + 1];
                elist.data[j + 1] = temp;
            }
}

void print() {
    int i, cost = 0;
    printf("\nMinimum Spanning Tree Edges:\n");
    for (i = 0; i < spanlist.n; i++) {
        printf("\nEdge %d: (%d, %d) with weight %d",
               i + 1, spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
        cost += spanlist.data[i].w;
    }
    printf("\n\nTotal Cost of Spanning Tree: %d\n", cost);
}

int main() {
    int i, j, total_cost;

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

    printf("\nEnter the adjacency matrix:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &Graph[i][j]);

    kruskal();
    print();
```
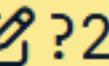
```
106
107        return 0;
108    }
```

```
>_pwsh    ⇒ Exp_13    main ≡ ?2    1ms
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-A
kal.exe" ; del "Kruskal.exe" }
Enter number of vertices: 5

Enter the adjacency matrix:
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0


Minimum Spanning Tree Edges:

Edge 1: (1, 0) with weight 2
Edge 2: (2, 1) with weight 3
Edge 3: (4, 1) with weight 5
Edge 4: (3, 0) with weight 6

Total Cost of Spanning Tree: 16
```

```c
 1  #include <stdio.h>
 2  #include <limits.h>
 3
 4  #define V 4
 5  #define INF 99999
 6
 7  void printSolution(int dist[][V]);
 8
 9  void floydWarshall(int graph[][V]) {
10      int dist[V][V], i, j, k;
11
12      // Initialize the solution matrix same as input graph matrix
13      for (i = 0; i < V; i++) {
14          for (j = 0; j < V; j++) {
15              dist[i][j] = graph[i][j];
16          }
17      }
18
19      // Add all vertices one by one to the set of intermediate vertices
20      for (k = 0; k < V; k++) {
21          // Pick all vertices as source one by one
22          for (i = 0; i < V; i++) {
23              // Pick all vertices as destination for the above picked source
24              for (j = 0; j < V; j++) {
25                  // If vertex k is on the shortest path from i to j,
26                  // then update the value of dist[i][j]
27                  if (dist[i][k] != INF && dist[k][j] != INF &&
28                      dist[i][k] + dist[k][j] < dist[i][j]) {
29                      dist[i][j] = dist[i][k] + dist[k][j];
30                  }
31              }
32          }
33      }
34
35      printSolution(dist);
36  }
37
38  void printSolution(int dist[][V]) {
39      printf("Following matrix shows the shortest distances between every pair of
    vertices:\n");
40      for (int i = 0; i < V; i++) {
41          for (int j = 0; j < V; j++) {
42              if (dist[i][j] == INF)
43                  printf("%7s", "INF");
44              else
45                  printf("%7d", dist[i][j]);
46          }
47          printf("\n");
48      }
49  }
50
51  int main() {
```

```c
    int graph[V][V] = {
        {0, 5, INF, 10},
        {INF, 0, 3, INF},
        {INF, INF, 0, 1},
        {INF, INF, INF, 0}
    };

    floydWarshall(graph);

    return 0;
}
```

```
>_pwsh    Exp_14    main ≡  ?2    1ms
  >> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_15\" ; gcc "a_
oyd.exe" ; del "a_Floyd.exe" }
Following matrix shows the shortest distances between every pair of vertices:
      0       5       8       9
    INF       0       3       4
    INF     INF       0       1
    INF     INF     INF       0
```

```c
1   #include <stdio.h>
2   #include <limits.h>
3
4   #define N 4   // Number of cities
5   #define INF INT_MAX
6
7   int dist[N][N];   // Distance matrix
8   int dp[1 << N][N];   // DP table: dp[mask][i] = min cost to visit cities in mask
    ending at city i
9
10  int min(int a, int b) {
11      return (a < b) ? a : b;
12  }
13
14  // Function to solve TSP using Dynamic Programming
15  int tsp(int mask, int pos) {
16      // If all cities have been visited
17      if (mask == (1 << N) - 1) {
18          return dist[pos][0];   // Return to starting city
19      }
20
21      // If already computed
22      if (dp[mask][pos] != -1) {
23          return dp[mask][pos];
24      }
25
26      int ans = INF;
27
28      // Try visiting all unvisited cities
29      for (int city = 0; city < N; city++) {
30          // If city is not visited and there is a path
31          if ((mask & (1 << city)) == 0 && dist[pos][city] != INF) {
32              int newAns = dist[pos][city] + tsp(mask | (1 << city), city);
33              ans = min(ans, newAns);
34          }
35      }
36
37      return dp[mask][pos] = ans;
38  }
39
40  int main() {
41      // Initialize distance matrix
42      printf("Enter the distance matrix (%d x %d):\n", N, N);
43      printf("(Enter %d for no direct path)\n", INF);
44
45      for (int i = 0; i < N; i++) {
46          for (int j = 0; j < N; j++) {
47              scanf("%d", &dist[i][j]);
48          }
49      }
50
51      // Initialize DP table with -1
```

```c
    for (int i = 0; i < (1 << N); i++) {
        for (int j = 0; j < N; j++) {
            dp[i][j] = -1;
        }
    }

    // Start from city 0 with only city 0 visited
    int minCost = tsp(1, 0);

    if (minCost >= INF) {
        printf("\nNo valid tour exists!\n");
    } else {
        printf("\nMinimum cost of TSP tour: %d\n", minCost);
    }

    return 0;
}
```

```
>_pwsh  ⇒ Exp_15  ⓦ main ≡ ✍?2  ⛛ 297ms
  ↳>> cd "d:\SelfRepoClone\Design-and-Analysis-of-AT
sman_DP.exe" ; if ($?) { .\"b_TravellingSalesman_DF
Enter the distance matrix (4 x 4):
(Enter 2147483647 for no direct path)
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0

Minimum cost of TSP tour: 80
```

```c
 1  #include <stdio.h>
 2  #include <stdbool.h>
 3
 4  #define MAX 20
 5
 6  int graph[MAX][MAX];
 7  int path[MAX];
 8  int n;
 9  int cycleCount = 0;
10
11  void printPath() {
12      printf("Hamiltonian Cycle %d: ", ++cycleCount);
13      for (int i = 0; i < n; i++) {
14          printf("%d ", path[i]);
15      }
16      printf("%d\n", path[0]); // Return to starting vertex
17  }
18
19  bool isSafe(int v, int pos) {
20      // Check if vertex v is adjacent to the previous vertex
21      if (graph[path[pos - 1]][v] == 0)
22          return false;
23
24      // Check if vertex v is already in path
25      for (int i = 0; i < pos; i++) {
26          if (path[i] == v)
27              return false;
28      }
29
30      return true;
31  }
32
33  bool hamiltonianCycleUtil(int pos) {
34      // Base case: if all vertices are included in path
35      if (pos == n) {
36          // Check if there is an edge from the last vertex to the first vertex
37          if (graph[path[pos - 1]][path[0]] == 1) {
38              printPath();
39              return true; // Continue to find more cycles
40          }
41          return false;
42      }
43
44      // Try different vertices as the next candidate
45      for (int v = 1; v < n; v++) {
46          if (isSafe(v, pos)) {
47              path[pos] = v;
48
49              // Recur to construct rest of the path
50              hamiltonianCycleUtil(pos + 1);
51
```

```c
52              // Backtrack
53              path[pos] = -1;
54          }
55      }
56
57      return false;
58  }
59
60  void findHamiltonianCycles() {
61      // Initialize path array
62      for (int i = 0; i < n; i++)
63          path[i] = -1;
64
65      // Start from vertex 0
66      path[0] = 0;
67      cycleCount = 0;
68
69      printf("\nFinding all Hamiltonian Cycles:\n");
70      hamiltonianCycleUtil(1);
71
72      if (cycleCount == 0) {
73          printf("No Hamiltonian Cycle exists in the given graph.\n");
74      } else {
75          printf("\nTotal number of Hamiltonian Cycles: %d\n", cycleCount);
76      }
77  }
78
79  int main() {
80      printf("Enter the number of vertices: ");
81      scanf("%d", &n);
82
83      printf("\nEnter the adjacency matrix:\n");
84      for (int i = 0; i < n; i++) {
85          for (int j = 0; j < n; j++) {
86              scanf("%d", &graph[i][j]);
87          }
88      }
89
90      printf("\nAdjacency Matrix:\n");
91      for (int i = 0; i < n; i++) {
92          for (int j = 0; j < n; j++) {
93              printf("%d ", graph[i][j]);
94          }
95          printf("\n");
96      }
97
98      findHamiltonianCycles();
99
100     return 0;
101 }
```

```
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_1
; if ($?) { .\"HamiltonianCycles.exe" ; del "HamiltonianCycles.e
```

Enter the number of vertices: 5

Enter the adjacency matrix:
0 1 0 1 1
1 0 1 0 1
0 1 0 1 1
1 0 1 0 1
1 1 1 1 0

Adjacency Matrix:
0 1 0 1 1
1 0 1 0 1
0 1 0 1 1
1 0 1 0 1
1 1 1 1 0

Finding all Hamiltonian Cycles:
Hamiltonian Cycle 1: 0 1 2 3 4 0
Hamiltonian Cycle 2: 0 1 2 4 3 0
Hamiltonian Cycle 3: 0 1 4 2 3 0
Hamiltonian Cycle 4: 0 3 2 1 4 0
Hamiltonian Cycle 5: 0 3 2 4 1 0
Hamiltonian Cycle 6: 0 3 4 2 1 0
Hamiltonian Cycle 7: 0 4 1 2 3 0
Hamiltonian Cycle 8: 0 4 3 2 1 0

Total number of Hamiltonian Cycles: 8

```
>> cd "d:\SelfRepoClone\Design-and-Analysis-of-Algorithms\Exp_1
; if ($?) { .\"HamiltonianCycles.exe" ; del "HamiltonianCycles.e
```

Enter the number of vertices: 4

Enter the adjacency matrix:
0 1 0 0
1 0 1 0
0 1 0 0
0 0 0 0

Adjacency Matrix:
0 1 0 0
1 0 1 0
0 1 0 0
0 0 0 0

Finding all Hamiltonian Cycles:
No Hamiltonian Cycle exists in the given graph.