```
// Lab 1 : Implementation of Bubble , Selection and Insertion Sort :  Selection

// sort Source Code :

#include <stdio.h>

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

  printf("Before: ");
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  putchar(’\n’);
  for (int i = 0; i < n; i++) {
    int swapped = 0;
    for (int j = 0; j < n - i - 1; j++) {
      step_count++;
      if (arr[j] > arr[j + 1]) {
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
        swapped = 1;
      }
    }
    if (!swapped)
      break;
  }

  printf("After:");
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\nSteps: %d\n", step_count);

  return 0;
}
```

```
// Insertion Sort :
// Source Code :

#include <stdio.h>

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

  printf("Before: ");
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  putchar(’\n’);
  for (int i = 0; i < n - 1; i++) {
    int min = i;
    for (int j = i + 1; j < n; j++) {
      step_count++;
      if (arr[j] < arr[min])
        min = j;
    }
    if (min != i) {
      int t = arr[i];
      arr[i] = arr[min];
      arr[min] = t;
    }
  }

  printf("After:");
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\nSteps: %d\n", step_count);

  return 0;
}
```

```
// BubbleSort :
// Source Code :

#include <stdio.h>

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

  printf("Before: ");
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  putchar(’\n’);
  for (int i = 1; i < n; i++) {
    int key = arr[i], j = i - 1;
    while (j >= 0 && arr[j] > key) {
      step_count++;
      arr[j + 1] = arr[j];
      j--;
    }
    if (j >= 0)
      step_count++;
    arr[j + 1] = key;
  }

  printf("After: ");
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\nSteps: %d\n", step_count);

  return 0;
}
```

```
// Lab2 : Implementation of Merge Sort :
// Source Code :

#include <stdio.h>

int step_count = 0;

void merge(int a[], int l, int m, int r) {
  int n1 = m - l + 1, n2 = r - m;
  int L[n1], R[n2];
  for (int i = 0; i < n1; i++)
    L[i] = a[l + i];
  for (int i = 0; i < n2; i++)
    R[i] = a[m + 1 + i];
  int i = 0, j = 0, k = l;
  while (i < n1 && j < n2) {
    step_count++;
    if (L[i] <= R[j])
      a[k++] = L[i++];
    else
      a[k++] = R[j++];
  }
  while (i < n1)
    a[k++] = L[i++];
  while (j < n2)
    a[k++] = R[j++];
}

void merge_sort(int a[], int l, int r) {
  if (l < r) {
    int m = (l + r) / 2;
    merge_sort(a, l, m);
    merge_sort(a, m + 1, r);
    merge(a, l, m, r);
  }
}

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

  printf("Before: ");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  putchar(’\n’);
  merge_sort(a, 0, n - 1);

  printf("After: ");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\nSteps: %d\n", step_count);

  return 0;
}
```

```
// Lab3 : Implementation of Quick Sort :

#include <stdio.h>

int step_count = 0;

int partition(int a[], int low, int high) {
  int p = a[high], i = low - 1;
  for (int j = low; j < high; j++) {
    step_count++;
    if (a[j] < p) {
      int t = a[++i];
      a[i] = a[j];
      a[j] = t;
    }
  }
  int t = a[i + 1];
  a[i + 1] = a[high];
  a[high] = t;
  return i + 1;
}

void quick_sort(int a[], int low, int high) {
  if (low < high) {
    int pi = partition(a, low, high);
    quick_sort(a, low, pi - 1);
    quick_sort(a, pi + 1, high);
  }
}

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

  printf("Before: ");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  putchar(’\n’);

  quick_sort(a, 0, n - 1);

  printf("After: ");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\nSteps: %d\n", step_count);
  return 0;
}
```

```
// Lab4 : Implementation of  Randomized Quick Sort

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

int step_count = 0;

int partition(int a[], int l, int h) {
  int p = a[h], i = l - 1;
  for (int j = l; j < h; j++) {
    step_count++;
    if (a[j] < p) {
      int t = a[++i];
      a[i] = a[j];
      a[j] = t;
    }
  }
  int t = a[i + 1];
  a[i + 1] = a[h];
  a[h] = t;
  return i + 1;
}

int rand_partition(int a[], int l, int h) {
  int r = l + rand() % (h - l + 1);
  int t = a[r];
  a[r] = a[h];
  a[h] = t;
  return partition(a, l, h);
}

void quick_sort(int a[], int l, int h) {
  if (l < h) {
    int pi = rand_partition(a, l, h);
    quick_sort(a, l, pi - 1);
    quick_sort(a, pi + 1, h);
  }
}

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

  printf("Before: ");
  for (int i = 0; i < n; i++)
```

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

  quick_sort(a, 0, n - 1);

  printf("After: ");
  for (int i = 0; i < n; i++)
    printf("%d ", a[i]);
  printf("\nSteps: %d\n", step_count);

  return 0;
}
```

```
// Lab5 : implementation of 0/1 Knapsack problem using Dynamic approach

#include <stdio.h>

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

int main() {
  int n, cap;
  printf("Enter number of items and capacity: ");
  scanf("%d %d", &n, &cap);
  int w[n], p[n], dp[n + 1][cap + 1];

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

  for (int i = 0; i <= n; i++)
    for (int j = 0; j <= cap; j++) {
      if (i == 0 || j == 0)
        dp[i][j] = 0;
      else if (w[i - 1] <= j)
        dp[i][j] = max(p[i - 1] + dp[i - 1][j - w[i - 1]], dp[i - 1][j]);
      else
        dp[i][j] = dp[i - 1][j];
    }

  printf("Table:\n");
  for (int i = 0; i <= n; i++) {
    for (int j = 0; j <= cap; j++)
      printf("%2d ", dp[i][j]);
    putchar(’\n’);
  }

  printf("Max Profit: %d\n", dp[n][cap]);
  return 0;
}
```

// Lab 6: implementation Of Matrix chain Multiplication Problem Source Code :

```
#include <stdio.h>

int main() {
  int n, step = 0;
  printf("Enter number of matrices: ");
  scanf("%d", &n);
  int d[n + 1];
  printf("Enter dimensions: ");
  for (int i = 0; i <= n; i++)
    scanf("%d", &d[i]);

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

  for (int L = 2; L <= n; L++) {
    for (int i = 0; i <= n - L; i++) {
      int j = i + L - 1;
      m[i][j] = 1e9;
      for (int k = i; k < j; k++) {
        step++;
        int cost = m[i][k] + m[k + 1][j] + d[i] * d[k + 1] * d[j + 1];
        if (cost < m[i][j])
          m[i][j] = cost;
      }
    }
  }

  printf("Table:\n");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      if (j < i)
        printf(" - ");
      else
        printf("%3d ", m[i][j]);
    putchar(’\n’);
  }

  printf("Min Multiplications: %d\n", m[0][n - 1]);
  printf("Steps: %d\n", step);
  return 0;
}
```

```
// Lab 7 : Implementation of Dynamic Programming based C++ program t
o find
// minimum number operations to convert str1 to str2

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

int min(int a, int b, int c) {
  return a < b ? (a < c ? a : c) : (b < c ? b : c);
}

int main() {
  char str1[100], str2[100];
  printf("Enter first string: ");
  scanf("%s", str1);
  printf("Enter second string: ");
  scanf("%s", str2);

  int m = strlen(str1), n = strlen(str2);
  int dp[m + 1][n + 1];

  for (int i = 0; i <= m; i++)
    dp[i][0] = i;
  for (int j = 0; j <= n; j++)
    dp[0][j] = j;

  for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
      if (str1[i - 1] == str2[j - 1])
        dp[i][j] = dp[i - 1][j - 1];
      else
        dp[i][j] = 1 + min(dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]);
    }
  }

  printf("Minimum operations: %d\n", dp[m][n]);
  return 0;
}
```

```
// Lab 8: Program for Floyd Warshall Algorithm Source Code :

#include <stdio.h>

int main() {
  int n, step = 0;
  printf("Enter number of vertices: ");
  scanf("%d", &n);
  int dist[n][n];

  printf("Enter the adjacency matrix (use a large number for infinity):\n");
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf("%d", &dist[i][j]);

  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        step++;
        if (dist[i][j] > dist[i][k] + dist[k][j])
          dist[i][j] = dist[i][k] + dist[k][j];
      }
    }
  }

  printf("Shortest paths matrix:\n");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      printf("%4d ", dist[i][j]);
    putchar(’\n’);
  }

  printf("Steps: %d\n", step);
  return 0;
}
```

```
// Lab 9: program for Dijkstra’s single source shortest path

#include <limits.h>
#include <stdio.h>

#define INF INT_MAX

int min_distance(int dist[], int spt_set[], int n) {
  int min = INF, min_index;
  for (int v = 0; v < n; v++) {
    if (spt_set[v] == 0 && dist[v] <= min) {
      min = dist[v];
      min_index = v;
    }
  }
  return min_index;
}

int main() {
  int n, source, step = 0;
  printf("Enter number of vertices: ");
  scanf("%d", &n);

  int graph[n][n], dist[n], spt_set[n];

  printf("Enter the adjacency matrix (use a large number for infinity):\n");
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      scanf("%d", &graph[i][j]);

  printf("Enter the source vertex (0-based index): ");
  scanf("%d", &source);

  for (int i = 0; i < n; i++) {
    dist[i] = INF;
    spt_set[i] = 0;
  }
  dist[source] = 0;

  for (int count = 0; count < n - 1; count++) {
    int u = min_distance(dist, spt_set, n);
    spt_set[u] = 1;

    for (int v = 0; v < n; v++) {
      step++;
      if (!spt_set[v] && graph[u][v] != INF && dist[u] != INF &&
          dist[u] + graph[u][v] < dist[v]) {
        dist[v] = dist[u] + graph[u][v];
      }
    }
  }

  printf("Shortest distances from source vertex %d:\n", source);
  for (int i = 0; i < n; i++) {
    if (dist[i] == INF)
      printf("Vertex %d: INF\n", i);
    else
      printf("Vertex %d: %d\n", i, dist[i]);
  }

  printf("Steps: %d\n", step);
  return 0;
}
```

```
// Lab 10 : Program to solve fractional Knapsack Problem Source Code :

#include <stdio.h>

typedef struct {
  int weight;
  int value;
  float ratio;
} Item;

int main() {
  int n, capacity, step = 0;
  printf("Enter number of items and capacity of knapsack: ");
  scanf("%d %d", &n, &capacity);

  Item items[n];
  for (int i = 0; i < n; i++) {
    printf("Enter weight and value for item %d: ", i + 1);
    scanf("%d %d", &items[i].weight, &items[i].value);
    items[i].ratio = (float)items[i].value / items[i].weight;
  }

  for (int i = 0; i < n - 1; i++) {
    for (int j = i + 1; j < n; j++) {
      step++;
      if (items[i].ratio < items[j].ratio) {
        Item temp = items[i];
        items[i] = items[j];
        items[j] = temp;
      }
    }
  }

  int totalValue = 0;
  float totalWeight = 0.0;

  for (int i = 0; i < n; i++) {
    if (totalWeight + items[i].weight <= capacity) {
      totalWeight += items[i].weight;
      totalValue += items[i].value;
    } else {
      int remainingWeight = capacity - totalWeight;
      totalValue += items[i].value * ((float)remainingWeight / items[i].weig
ht);
      break;
    }
  }

  printf("Maximum value in Knapsack = %d\n", totalValue);
  printf("Steps: %d\n", step);
  return 0;
}
```

```
// Lab 11: program to solve N Queen Problem using backtracking

#include <stdio.h>

int step = 0;

int is_safe(int board[][10], int row, int col, int n) {
  for (int i = 0; i < row; i++) {
    if (board[i][col] == 1)
      return 0;
    if (col - (row - i) >= 0 && board[i][col - (row - i)] == 1)
      return 0;
    if (col + (row - i) < n && board[i][col + (row - i)] == 1)
      return 0;
  }
  return 1;
}

int solve_nqueens(int board[][10], int row, int n) {
  step++;
  if (row == n)
    return 1;

  for (int col = 0; col < n; col++) {
    if (is_safe(board, row, col, n)) {
      board[row][col] = 1;
      if (solve_nqueens(board, row + 1, n))
        return 1;
      board[row][col] = 0;
    }
  }
  return 0;
}

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

  int board[10][10] = {0};

  if (solve_nqueens(board, 0, n)) {
    printf("Solution:\n");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        printf("%d ", board[i][j]);
      }
      printf("\n");
    }
    printf("Steps: %d\n", step);
  } else {
    printf("No solution exists\n");
  }

  return 0;
}
```

```
// Lab12 : Kruskal’s algorithm to find Minimum Spanning Tree of a given
// connected, undirected graph

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

#define MAX 10

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

int parent[MAX], rank[MAX];

int find(int i) {
  if (parent[i] != i)
    parent[i] = find(parent[i]);
  return parent[i];
}

void union_set(int u, int v) {
  int root_u = find(u);
  int root_v = find(v);

  if (root_u != root_v) {

    if (rank[root_u] > rank[root_v]) {
      parent[root_v] = root_u;
    } else if (rank[root_u] < rank[root_v]) {
      parent[root_u] = root_v;
    } else {
      parent[root_v] = root_u;
      rank[root_u]++;
    }
  }
}

int compare(const void *a, const void *b) {
  return ((Edge *)a)->weight - ((Edge *)b)->weight;
}

int main() {
  int n, m, total_weight = 0, steps = 0;

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

  Edge edges[m];

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

  printf("Enter the edges (u v weight):\n");
  for (int i = 0; i < m; i++) {
    scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].weight);
  }

  qsort(edges, m, sizeof(Edge), compare);

  printf("Minimum Spanning Tree (MST) edges:\n");
  for (int i = 0; i < m; i++) {
    steps++;
    int u = edges[i].u;
    int v = edges[i].v;
    int weight = edges[i].weight;

    if (find(u) != find(v)) {
      union_set(u, v);
      total_weight += weight;
      printf("%d - %d: %d\n", u, v, weight);
    }
  }

  printf("Total weight of MST: %d\n", total_weight);
  printf("Steps: %d\n", steps);

  return 0;
}
```

```c
// Lab 13 :program for Prim’s Minimum

#include <limits.h>
#include <stdio.h>

#define MAX 10

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

void primMST(int graph[MAX][MAX], int n) {
  int parent[n];
  int key[n];
  int mstSet[n];
  int totalWeight = 0, steps = 0;

  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;
    steps++;

    for (int v = 0; v < n; v++) {
      if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {
        key[v] = graph[u][v];
        parent[v] = u;
      }
    }
  }
  printf("Minimum Spanning Tree (MST) edges:\n");
  for (int i = 1; i < n; i++) {
    printf("%d - %d: %d\n", parent[i], i, graph[i][parent[i]]);
    totalWeight += graph[i][parent[i]];
  }

  printf("Total weight of MST: %d\n", totalWeight);
  printf("Steps: %d\n", steps);
}

int main() {
  int n;

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

  int graph[MAX][MAX];
  printf("Enter the adjacency matrix of the graph:\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;
}
```

```
// Lab 14 : implementation of Subset sum problem

#include <stdio.h>

#define MAX 20

int count = 0;

void findSubsetSum(int set[], int n, int target, int index, int current[],
                   int current_size) {
  if (index == n) {
    int sum = 0;
    for (int i = 0; i < current_size; i++) {
      sum += current[i];
    }
    if (sum == target) {
      count++;
      printf("Subset %d: {", count);
      for (int i = 0; i < current_size; i++) {
        printf("%d", current[i]);
        if (i < current_size - 1)
          printf(", ");
      }
      printf("}\n");
    }
    return;
  }

  current[current_size] = set[index];
  findSubsetSum(set, n, target, index + 1, current, current_size + 1);

  findSubsetSum(set, n, target, index + 1, current, current_size);
}

int main() {
  int set[MAX], n, target, current[MAX];

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

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

  printf("Subsets that sum to %d:\n", target);
  findSubsetSum(set, n, target, 0, current, 0);

  printf("Total subsets found: %d\n", count);

  return 0;
}
```

```
// Lab 15: implementation of job sequence in deadlines

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

#define MAX 100

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

int compare(const void *a, const void *b) {
  return ((Job *)b)->profit - ((Job *)a)->profit;
}

void jobSequencing(Job jobs[], int n) {
  int result[MAX];
  int slot[MAX];
  int totalProfit = 0, count = 0;

  for (int i = 0; i < n; i++) {
    slot[i] = -1;
  }

  qsort(jobs, n, sizeof(Job), compare);

  for (int i = 0; i < n; i++) {

    for (int j = jobs[i].deadline - 1; j >= 0; j--) {
      if (slot[j] == -1) {
        slot[j] = i;
        totalProfit += jobs[i].profit;
        count++;
        break;
      }
    }
  }

  printf("Job Sequence that maximizes profit:\n");
  for (int i = 0; i < n; i++) {
    if (slot[i] != -1) {
      printf("Job %d with profit %d, Deadline %d\n", jobs[slot[i]].id,
             jobs[slot[i]].profit, jobs[slot[i]].deadline);
    }
  }
  printf("Total Profit: %d\n", totalProfit);
  printf("Total Jobs Scheduled: %d\n", count);
}

int main() {
  int n;

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

  Job jobs[n];

  printf("Enter job details (ID, Deadline, Profit):\n");
  for (int i = 0; i < n; i++) {
    jobs[i].id = i + 1;
    scanf("%d %d", &jobs[i].deadline, &jobs[i].profit);
  }

  jobSequencing(jobs, n);

  return 0;
}
```