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

struct Student {
    char name[50];
    int roll;
    int marks;
};

int i,j;

void swap(struct Student *a, struct Student *b, int *swapCount) {
    struct Student temp = *a;
    *a = *b;
    *b = temp;
    (*swapCount)++;
}

int partition(struct Student arr[], int low, int high, int *swapCount) {
    struct Student pivot = arr[high];
    int i = low - 1;
    for (j = low; j < high; j++) {
        if (arr[j].roll <= pivot.roll) {
            i++;
            swap(&arr[i], &arr[j], swapCount);
        }
    }
    swap(&arr[i + 1], &arr[high], swapCount);
    return (i + 1);
}

void quickSort(struct Student arr[], int low, int high, int *swapCount) {
    if (low < high) {
        int pi = partition(arr, low, high, swapCount);
        quickSort(arr, low, pi - 1, swapCount);
        quickSort(arr, pi + 1, high, swapCount);
    }
}

int main() {
    struct Student students[] = {
        {"Alice", 3, 85},
        {"Bob", 2, 72},
        {"Charlie", 5, 90},
        {"David", 1, 65},
        {"Eve", 4, 78}
    };
    int n = sizeof(students) / sizeof(students[0]);
```

```
    int swapCount = 0;
    printf("Before sorting:\n");
    printf("Name\tRoll\tMarks\n");
    for (i = 0; i < n; i++) {
        printf("%s\t%d\t%d\n", students[i].name, students[i].roll,
students[i].marks);
    }

    quickSort(students, 0, n - 1, &swapCount);

    printf("\nAfter sorting:\n");
    printf("Name\tRoll\tMarks\n");
    for (i = 0; i < n; i++) {
        printf("%s\t%d\t%d\n", students[i].name, students[i].roll,
students[i].marks);
    }

    printf("\nTotal number of swaps performed: %d\n", swapCount);

    return 0;
}
```

```
Before sorting:
Name    Roll    Marks
Alice   3       85
Bob     2       72
Charlie 5       90
David   1       65
Eve     4       78

After sorting:
Name    Roll    Marks
David   1       65
Bob     2       72
Alice   3       85
Eve     4       78
Charlie 5       90

Total number of swaps performed: 7
```

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

#define V 4
int i, v, j;
int minKey(int key[], bool mstSet[])
```

```
{
    int min = INT_MAX, min_index;

    for (v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}
int printMST(int parent[], int graph[V][V])
{
    int totalWeight = 0;
    printf("Edge \tWeight\n");
    for (i = 1; i < V; i++){
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
        totalWeight += graph[i][parent[i]];
    }
    printf("Total Weight = %d\n", totalWeight);
}
void primMST(int graph[V][V])
{
    int parent[V];
    int key[V], count;
    bool mstSet[V];

    for (i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    key[0] = 0;

    parent[0] = -1;

    for (count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);

        mstSet[u] = true;
        for (v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph);
}

int main()
{
    int graph[V][V] = { { 0, 3, 0, 1 },
                        { 3, 0, 4, 8 },
                        { 0, 4, 0, 2 },
                        { 1, 8, 2, 0 }};
```

```
    primMST(graph);

    return 0;
}
```

```
Edge    Weight
0 - 1   3
3 - 2   2
0 - 3   1
Total Weight = 6
```