



SCHOOL OF  
COMPUTING

M M VASANTH

CH.SC.U4CSE24227

Week - 4

Date - 26/12/2025

Design and Analysis of Algorithm(23CSE211)

## 1. PRIM'S ALGORITHM

### Code:

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

#define INF 9999
int n;

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

void printMST(int parent[], int graph[20][20]) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < n; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}

void primMST(int graph[20][20]) {
    int parent[20];
    int key[20];
    int mstSet[20];

    for (int i = 0; i < n; i++)
        key[i] = INF, mstSet[i] = 0;

    key[0] = 0;
    parent[0] = -1;
```

```

    for (int count = 0; count < n - 1; count++) {
        int u = minKey(key, mstSet);
        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];
    }
    printMST(parent, graph);
}

int main() {
    int graph[20][20];
    printf("Name: M M VASANTH\n");
    printf("Roll No: CH.SC.U4CSE24227\n\n");
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    primMST(graph);
    return 0;
}

```

## Output:

```
vasanth@VasanthMM:/mnt/c/Users/vasan$ nano prims.c
vasanth@VasanthMM:/mnt/c/Users/vasan$ gcc prims.c -o prims
vasanth@VasanthMM:/mnt/c/Users/vasan$ ./prims
Name: M M VASANTH
Roll No: CH.SC.U4CSE24227

Enter number of vertices: 3
Enter adjacency matrix:
24
45
56
78
98
96
55
43
33
Edge      Weight
0 - 1     78
0 - 2     55
vasanth@VasanthMM:/mnt/c/Users/vasan$ |
```

### ☑ Time Complexity: $O(N^2)$

Since the graph is represented using an Adjacency Matrix, we use two nested loops. The outer loop runs  $N$  times to include every vertex in the MST. The inner loop (searching for the minimum key and updating neighbors) iterates through all  $N$  vertices. This results in  $N * N$  operations.

### ☑ Space Complexity: $O(N)$

I used three auxiliary arrays (key, parent, and mstSet), each of size  $N$ . Therefore, the space required is proportional to the number of vertices  $N$ .

## 2. KRUSKAL'S ALGORITHM

### Code:

```
#include <stdio.h>

#include <stdlib.h>

struct Edge {
    int src, dest, weight;
};

struct Edge edges[100];
int parent[20];
int n, e_cnt = 0;

int find(int i) {
    if (parent[i] == -1)
        return i;
    return find(parent[i]);
}

void Union(int x, int y) {
    int xset = find(x);
    int yset = find(y);
    if(xset != yset)
        parent[xset] = yset;
}
```

```
int compare(const void* a, const void* b) {  
    struct Edge* a1 = (struct Edge*)a;  
    struct Edge* b1 = (struct Edge*)b;  
    return a1->weight - b1->weight;  
}
```

```
void kruskalMST() {  
    int mst_weight = 0;  
    qsort(edges, e_cnt, sizeof(edges[0]), compare);  
  
    for (int i = 0; i < n; i++)  
        parent[i] = -1;  
  
    printf("Edge \tWeight\n");  
    for (int i = 0; i < e_cnt; i++) {  
        int x = find(edges[i].src);  
        int y = find(edges[i].dest);  
  
        if (x != y) {  
            printf("%d - %d \t%d \n", edges[i].src, edges[i].dest,  
edges[i].weight);  
            mst_weight += edges[i].weight;  
            Union(x, y);  
        }  
    }  
}
```

```
    printf("Total Cost: %d\n", mst_weight);
}

int main() {
    int weight;

    printf("Name: M M VASANTH\n");
    printf("Roll No: CH.SC.U4CSE24227\n\n");
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter number of edges: ");
    scanf("%d", &e_cnt);

    for(int i = 0; i < e_cnt; i++) {
        printf("Enter src, dest, weight for edge %d: ", i+1);
        scanf("%d %d %d", &edges[i].src, &edges[i].dest,
&edges[i].weight);
    }

    kruskalMST();
    return 0;
}
```

## Output:

```
vasanth@VasanthMM:/mnt/c/Users/vasan$ nano kruskals.c
vasanth@VasanthMM:/mnt/c/Users/vasan$ gcc kruskals.c -o kruskals
vasanth@VasanthMM:/mnt/c/Users/vasan$ ./kruskals
Name: M M VASANTH
Roll No: CH.SC.U4CSE24227

Enter number of vertices: 3
Enter number of edges: 3
Enter src, dest, weight for edge 1: 2
3
4
Enter src, dest, weight for edge 2: 5
6
7
Enter src, dest, weight for edge 3: 44
5
34
Edge    Weight
2 - 3    4
Total Cost: 4
vasanth@VasanthMM:/mnt/c/Users/vasan$ |
```

### ☐ Time Complexity: $O(E \log E)$ or $O(E \log N)$

The most time-consuming step is sorting all the edges by weight, which takes  $O(E \log E)$ . The Union-Find operations take nearly constant time on average ( $O(\alpha(N))$ ), so the sorting dominates the complexity.

### ☐ Space Complexity: $O(E + N)$

I used an array to store all  $E$  edges and a parent array of size  $N$  for the Union-Find structure. Thus, the space complexity depends on both the number of edges and vertices.