KRUSKAL'S ALGORITHM

```
#include <iostream>
#include <cstdlib>
#include <sys/time.h>
using namespace std;

int **randomGraph(int n)
{
	int **a = new int*[n];
	for (int i = 0; i < n; i++)
	{
	a[i] = new int[n];
	for (int j = 0; j < n; j++)
	{
	int k = (rand() % ((n - 1) - 0 + 1)) + 0;
	a[i][j] = k;
	}
	}
	return a;
}

int comparator(const void *p1, const void *p2)
{
	const int(*x)[3] = (const int(*)[3])p1;
	const int(*y)[3] = (const int(*)[3])p2;
	return (*x)[2] - (*y)[2];
}

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

int findParent(int parent[], int component)
{
	if (parent[component] == component)
	return component;
	return parent[component] = findParent(parent, parent[component]);
```

```
}

void unionSet(int u, int v, int parent[], int rank[], int n)
{
	u = findParent(parent, u);
	v = findParent(parent, v);
	if (rank[u] < rank[v])
	{
	parent[u] = v;
	}
	else if (rank[u] > rank[v])
	{
	parent[v] = u;
	}
	else
	{
	parent[v] = u;
	rank[u]++;
	}
}

void kruskalAlgo(int n, int **edge)
{
	qsort(edge, n, sizeof(edge[0]), comparator);
	int parent[n];
	int rank[n];
	makeSet(parent, rank, n);
	int minCost = 0;
	for (int i = 0; i < n; i++)
	{
	int v1 = findParent(parent, edge[i][0]);
	int v2 = findParent(parent, edge[i][1]);
	int wt = edge[i][2];
	if (v1 != v2)
	{
	unionSet(v1, v2, parent, rank, n);
	minCost += wt;
	}
	}
}

int main()
{
	struct timeval t1, t2;
```

```
        struct timezone z1, z2;
        int n = 4000;
        int **edge = randomGraph(n);
        gettimeofday(&t1, &z1);
        kruskalAlgo(n, edge);
        gettimeofday(&t2, &z2);
        cout << "Seconds: " << t2.tv_sec - t1.tv_sec << ", Microseconds: " <<
t2.tv_usec - t1.tv_usec << endl;
        return 0;
}
```

OBSERVATION TABLE:

No. of Vertices	1st Reading	2nd Reading	3rd Reading	4th Reading	5th Reading	Average
100	19	20	19	7	21	17.2
200	50	57	50	53	48	51.6
800	103	94	83	96	99	95
2000	241	204	225	209	218	219.4
3000	321	314	302	308	308	310.6
4000	442	423	433	414	421	426.6

ANALYSIS:

From the observation table, we can see that as the no of vertices increases, the time complexity is increasing linearly. With 4000 vertices, it takes small time to find MST. It is because, it has the graph with order n x 3.

PRIMS ALGORITHM

```
#include <limits.h>
#include <stdbool.h>
#include <iostream>
#include <sys/time.h>

#define V 4000

int **randomGraph() {
        int **a = new int*[V];
        for (int i = 0; i < V; i++) {
        a[i] = new int[V];
        for (int j = 0; j < V; j++) {
        int k = (rand() % (9 - 0 + 1)) + 0;
        a[i][j] = k;
        }
        }
        return a;
}
int minKey(int key[], bool mstSet[]) {
        int min = INT_MAX, min_index;
        for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
        min = key[v], min_index = v;
        return min_index;
}
void printMST(int parent[], int **graph) {
        std::cout << "Edge \tWeight\n";
        for (int i = 1; i < V; i++)
        std::cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << "\n";
}
void primMST(int **graph) {
        int parent[V];
        int key[V];
        bool mstSet[V];
        for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
        key[0] = 0;
        parent[0] = -1;
        for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
```

```
        for (int 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() {
        struct timeval t1, t2;
        struct timezone z1, z2;
        int **graph = randomGraph();
        gettimeofday(&t1, &z1);
        primMST(graph);
        gettimeofday(&t2, &z2);
        std::cout << "Seconds : " << t2.tv_sec - t1.tv_sec << ", Microseconds : " <<
t2.tv_usec - t1.tv_usec << "\n";
        return 0;
}
```

OBSERVATION TABLE:

No. of Vertices	1st Reading	2nd Reading	3rd Reading	4th Reading	5th Reading	Average
100	202	197	205	195	194	198.6
200	725	693	696	735	706	711
800	3797	4055	3997	3976	4137	3992.4
2000	23500	23171	23341	23212	23397	23324.2
3000	51803	51521	52102	52100	51970	51899.2
4000	95325	93602	94550	94159	91913	93909.8

ANALYSIS:

From the observation table, we can see that as the no of vertices increases, the time complexity is increasing exponentially. Here for just 4000 vertices, it takes very much time to find MST because it has 2 dimensional graph with n x n order.

Conclusion :

From the observation table, result table and by doing analysis, we can clearly say that kruskal's algorithm perform better with respect to prims's. It is more efficient algorithm to find MST.

BREADTH FIRST SEARCH

```
#include <iostream>
#include <stdlib.h>
#include <sys/time.h>
using namespace std;
#define N 1000

int Queue[10000];

int front = 0, rear = 0;

void Enqueue(int var)
{
	Queue[rear] = var;
	rear++;
}

void Dequeue()
{
	Queue[front] = 0;
	front++;
}

int visited[N] = {0};

int main()
{
	int graph[N][N];
	struct timeval t1, t2;
	struct timezone z1, z2;

	for (int i = 0; i < N; i++)
	{
	for (int j = 0; j < N; j++)
	{
	if (i == j)
	{
		graph[i][j] = 0;
		continue;
	}
	int k = rand() % 2;
	graph[i][j] = k;
```

```
        graph[j][i] = k;
        }
        }


        gettimeofday(&t1, &z1);
        Enqueue(1);
        visited[0] = 1;

        while (front != rear)
        {
        int curr = Queue[front];
        Dequeue();

        for (int i = 0; i < N; i++)
        {
        if ((graph[curr - 1][i] == 1) && (visited[i] == 0))
        {
                visited[i] = 1;
                Enqueue(i + 1);
                //cout<<i+1<<"  ";
        }
        }
        }
        gettimeofday(&t2, &z2);

        cout<<"\nSeconds : "<< t2.tv_sec - t1.tv_sec<< "Microseconds : "<<
t2.tv_usec - t1.tv_usec;
        return 0;
}
```

OBSERVATION TABLE:

No. of Vertices	1st Reading	2nd Reading	3rd Reading	4th Reading	5th Reading	Average
50	53	48	48	50	48	49.4
100	207	200	206	224	199	207.2
200	779	794	791	795	779	787.6
300	932	928	976	991	950	955.4
500	1665	1794	2518	1842	1777	1919.2
750	4351	4163	4360	4367	4428	4333.8
1000	7091	7088	6963	6883	7028	7010.6

ANALYSIS:

Observation table shows that as the no of vertices increases, the time is increasing means for bigger graph, the traversing time is bigger but here the time complexity is considerable to graph traversal.

DEPTH FIRST SEARCH

```
#include <iostream>
#include <stdlib.h>
#include <sys/time.h>
using namespace std;
#define N 1000
 int graph[N][N];
int visited[N] = {0};
void Dfs(int x)
{
        visited[x]=1;
        for (int  i = 0; i < N; i++)
        {
        if(graph[x][i]==1 && !visited[i])
        {
        Dfs(i);
        }
        }
}

int main()
{
        struct timeval t1, t2;
        struct timezone z1, z2;
        for (int i = 0; i < N; i++)
        {
        for (int j = 0; j < N; j++)
        {
        if (i == j)
        {
                graph[i][j] = 0;
                continue;
        }
        int k = rand() %  2;
        graph[i][j] = k;
```

```
        graph[j][i] = k;
        }
        }
        gettimeofday(&t1, &z1);
        Dfs(1);
        gettimeofday(&t2, &z2);
        cout <<endl;
        cout <<"Seconds" << t2.tv_sec - t1.tv_sec <<"  Microseconds" << t2.tv_usec -
t1.tv_usec<< endl;
        return 0;
}
```

OBSERVATION TABLE:

No. of Vertices	1st Reading	2nd Reading	3rd Reading	4th Reading	5th Reading	Average
50	53	49	48	51	47	49.6
100	200	209	202	199	199	201.8
200	779	790	792	795	793	789.8
300	916	928	932	993	950	943.8
500	1726	1723	1856	1819	1727	1770.2
750	4571	4706	4593	4435	4484	4557.8
1000	7187	7128	7299	7130	7211	7191

ANALYSIS:

From the observation table, we can see that as the no of vertices increases, the time complexity is increasing but the rate is not much compared to other algorithms. It traverse better even with bigger graph.

CONCLUSION:

As we can see from the tables that the values for BFS and DFS is considerably nearer. But we can conclude through it that the value in case of DFS is slightly better than BFS.