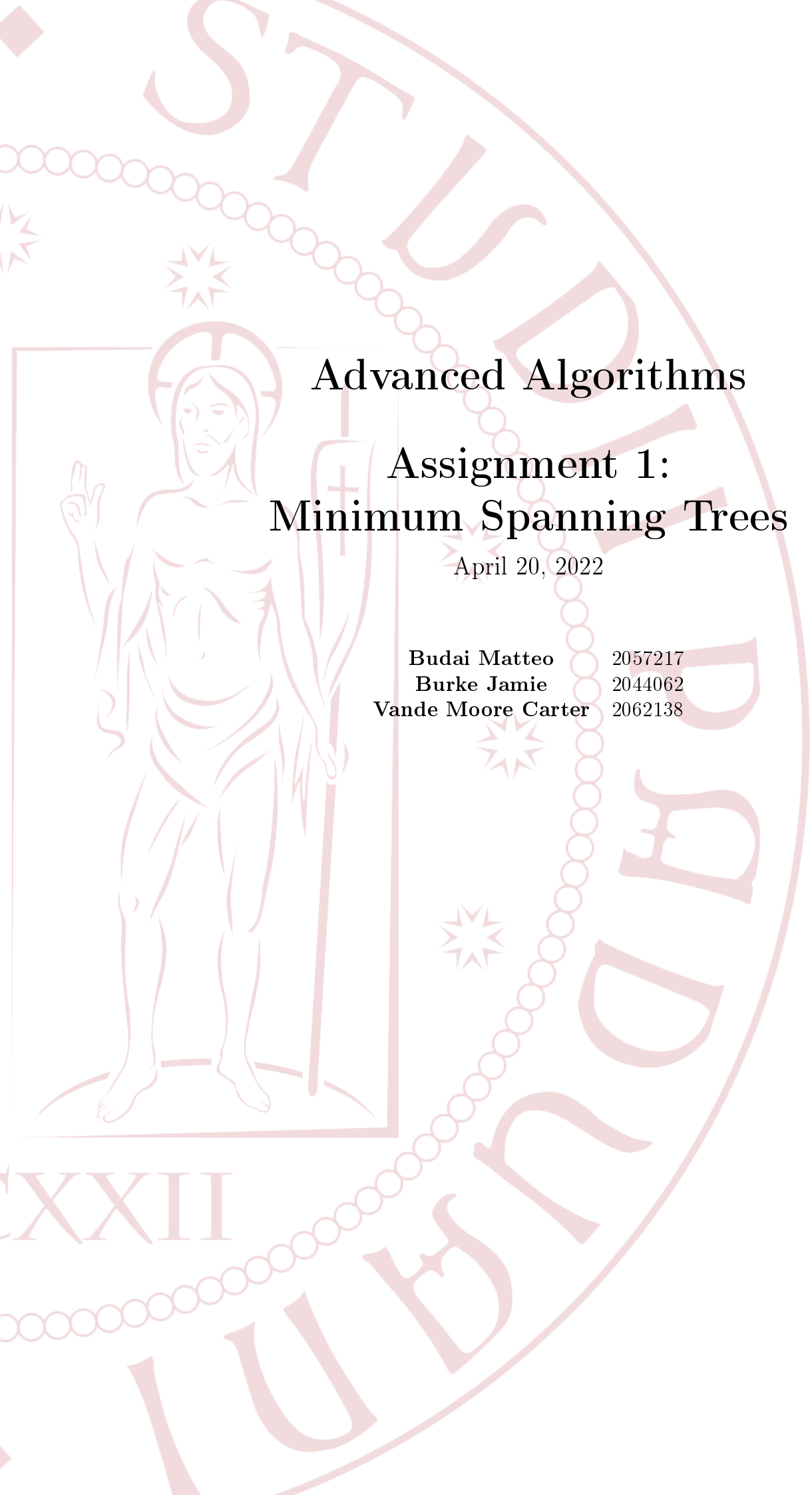

# Advanced Algorithms

## Assignment 1: Minimum Spanning Trees

April 20, 2022


| | |
|---|---|
| **Budai Matteo** | 2057217 |
| **Burke Jamie** | 2044062 |
| **Vande Moore Carter** | 2062138 |

# Contents

# 1 Introduction

# 2 Prim's Algorithm

```
1 PRIM (G, s)
2    X = {s} //set of vertexes included in the MST
3    A = ∅ //set of edges included in the MST
4    while there is an edge (u, v) with u ∈ X and v ∉ X do
5       (u*, v*) = a minimum cost such edge //light edge
6       add vertex v* to X
7       add edge (u*, v*) to A
8    return A
```

## 2.1 Data Structure

### 2.1.1 MinHeap

### 2.1.2 Node

### 2.1.3 Graph

## 2.2 Implementation

## 2.3 Complexity

# 3 Kruskal's Algorithm with "naive" implementation

```
1 KRUSKAL (G)
2 A = ∅
3 sort edges of G by cost
4 for each edge e, in non decreasing order of cost do
5    if A ∪ {e} is acyclic then
6       A = A ∪ {e}
7 return A
```

## 3.1 Data Structure

### 3.1.1 Graph

### 3.1.2 Edge

## 3.2 Implementation

## 3.3 Complexity

# 4 Kruskal's Algorithm with Union-Find

```
KRUSKAL(G):
    A = ∅
    For each vertex v ∈ G.V:
        MAKE-SET(v)
    sort edges of G by cost
    for each edge e, in non decreasing order of cost do
        if FIND-SET(u) ≠ FIND-SET(v):
            A = A ∪ {(u, v)}
            UNION(u, v)
    return A
```

## 4.1 Data Structure

### 4.1.1 Graph and Edge

### 4.1.2 Union Find

## 4.2 Implementation

## 4.3 Complexity

# 5 Results

## 5.1 Table with calculated MST

| Input file | num_nodes | num_edges | MST |
|---|---|---|---|
| input_random_01_10.txt | 10 | 9 | 29316 |
| input_random_02_10.txt | 10 | 11 | 16940 |
| input_random_03_10.txt | 10 | 13 | -44448 |
| input_random_04_10.txt | 10 | 10 | 25217 |
| input_random_05_20.txt | 20 | 24 | -32021 |
| input_random_06_20.txt | 20 | 24 | 25130 |
| input_random_07_20.txt | 20 | 28 | -41693 |
| input_random_08_20.txt | 20 | 26 | -37205 |
| input_random_09_40.txt | 40 | 56 | -114203 |
| input_random_10_40.txt | 40 | 50 | -31929 |
| input_random_11_40.txt | 40 | 50 | -79570 |
| input_random_12_40.txt | 40 | 52 | -79741 |
| input_random_13_80.txt | 80 | 108 | -139926 |
| input_random_14_80.txt | 80 | 99 | -198094 |
| input_random_15_80.txt | 80 | 104 | -110571 |
| input_random_16_80.txt | 80 | 114 | -233320 |
| input_random_17_100.txt | 100 | 136 | -141960 |
| input_random_18_100.txt | 100 | 129 | -271743 |
| input_random_19_100.txt | 100 | 137 | -288906 |
| input_random_20_100.txt | 100 | 132 | -229506 |
| input_random_21_200.txt | 200 | 267 | -510185 |
| input_random_22_200.txt | 200 | 269 | -515136 |
| input_random_23_200.txt | 200 | 269 | -444357 |
| input_random_24_200.txt | 200 | 267 | -393278 |
| input_random_25_400.txt | 400 | 540 | -1119906 |

| Input file | num_nodes | num_edges | MST |
|---|---|---|---|
| input_random_26_400.txt | 400 | 518 | -788168 |
| input_random_27_400.txt | 400 | 538 | -895704 |
| input_random_28_400.txt | 400 | 526 | -733645 |
| input_random_29_800.txt | 800 | 1063 | -1541291 |
| input_random_30_800.txt | 800 | 1058 | -1578294 |
| input_random_31_800.txt | 800 | 1076 | -1664316 |
| input_random_32_800.txt | 800 | 1049 | -1652119 |
| input_random_33_1000.txt | 1000 | 1300 | -2089013 |
| input_random_34_1000.txt | 1000 | 1313 | -1934208 |
| input_random_35_1000.txt | 1000 | 1328 | -2229428 |
| input_random_36_1000.txt | 1000 | 1344 | -2356163 |
| input_random_37_2000.txt | 2000 | 2699 | -4811598 |
| input_random_38_2000.txt | 2000 | 2654 | -4739387 |
| input_random_39_2000.txt | 2000 | 2652 | -4717250 |
| input_random_40_2000.txt | 2000 | 2677 | -4537267 |
| input_random_41_4000.txt | 4000 | 5360 | -8722212 |
| input_random_42_4000.txt | 4000 | 5315 | -9314968 |
| input_random_43_4000.txt | 4000 | 5340 | -9845767 |
| input_random_44_4000.txt | 4000 | 5368 | -8681447 |
| input_random_45_8000.txt | 8000 | 10705 | -17844628 |
| input_random_46_8000.txt | 8000 | 10670 | -18798446 |
| input_random_47_8000.txt | 8000 | 10662 | -18741474 |
| input_random_48_8000.txt | 8000 | 10757 | -18178610 |
| input_random_49_10000.txt | 10000 | 13301 | -22079522 |
| input_random_50_10000.txt | 10000 | 13340 | -22338561 |
| input_random_51_10000.txt | 10000 | 13287 | -22581384 |
| input_random_52_10000.txt | 10000 | 13311 | -22606313 |
| input_random_53_20000.txt | 20000 | 26667 | -45962292 |

| Input file | num_nodes | num_edges | MST |
|---|---|---|---|
| input_random_54_20000.txt | 20000 | 26826 | -45195405 |
| input_random_55_20000.txt | 20000 | 26673 | -47854708 |
| input_random_56_20000.txt | 20000 | 26670 | -46418161 |
| input_random_57_40000.txt | 40000 | 53415 | -92003321 |
| input_random_58_40000.txt | 40000 | 53446 | -94397064 |
| input_random_59_40000.txt | 40000 | 53242 | -88771991 |
| input_random_60_40000.txt | 40000 | 53319 | -93017025 |
| input_random_61_80000.txt | 80000 | 106914 | -186834082 |
| input_random_62_80000.txt | 80000 | 106633 | -185997521 |
| input_random_63_80000.txt | 80000 | 106586 | -182065015 |
| input_random_64_80000.txt | 80000 | 106554 | -180793224 |
| input_random_65_100000.txt | 100000 | 133395 | -230698391 |
| input_random_66_100000.txt | 100000 | 133214 | -230168572 |
| input_random_67_100000.txt | 100000 | 133524 | -231393935 |
| input_random_68_100000.txt | 100000 | 133463 | -231011693 |

## 5.2 Graph of the performance of Prim's Algorithm

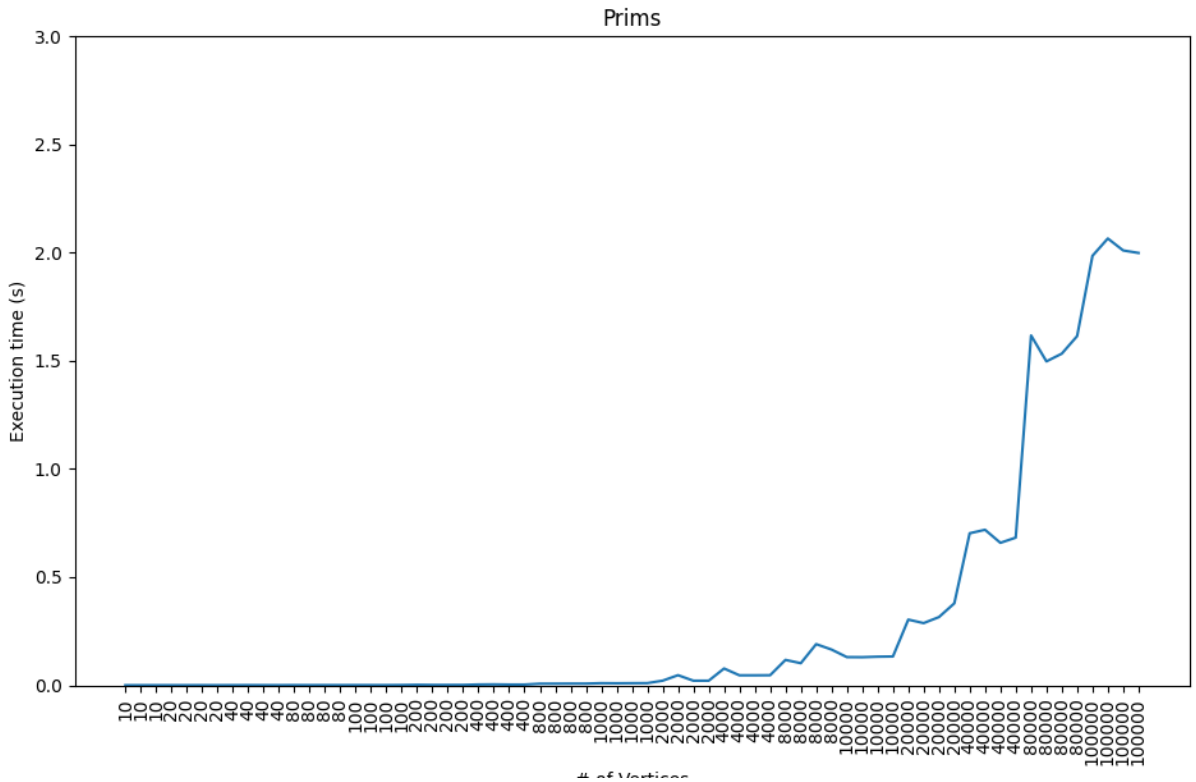


Figure 1: Performance of Prim's Algorithm

## 5.3 Graph of the performance of Kruskal's "naive" Algorithm

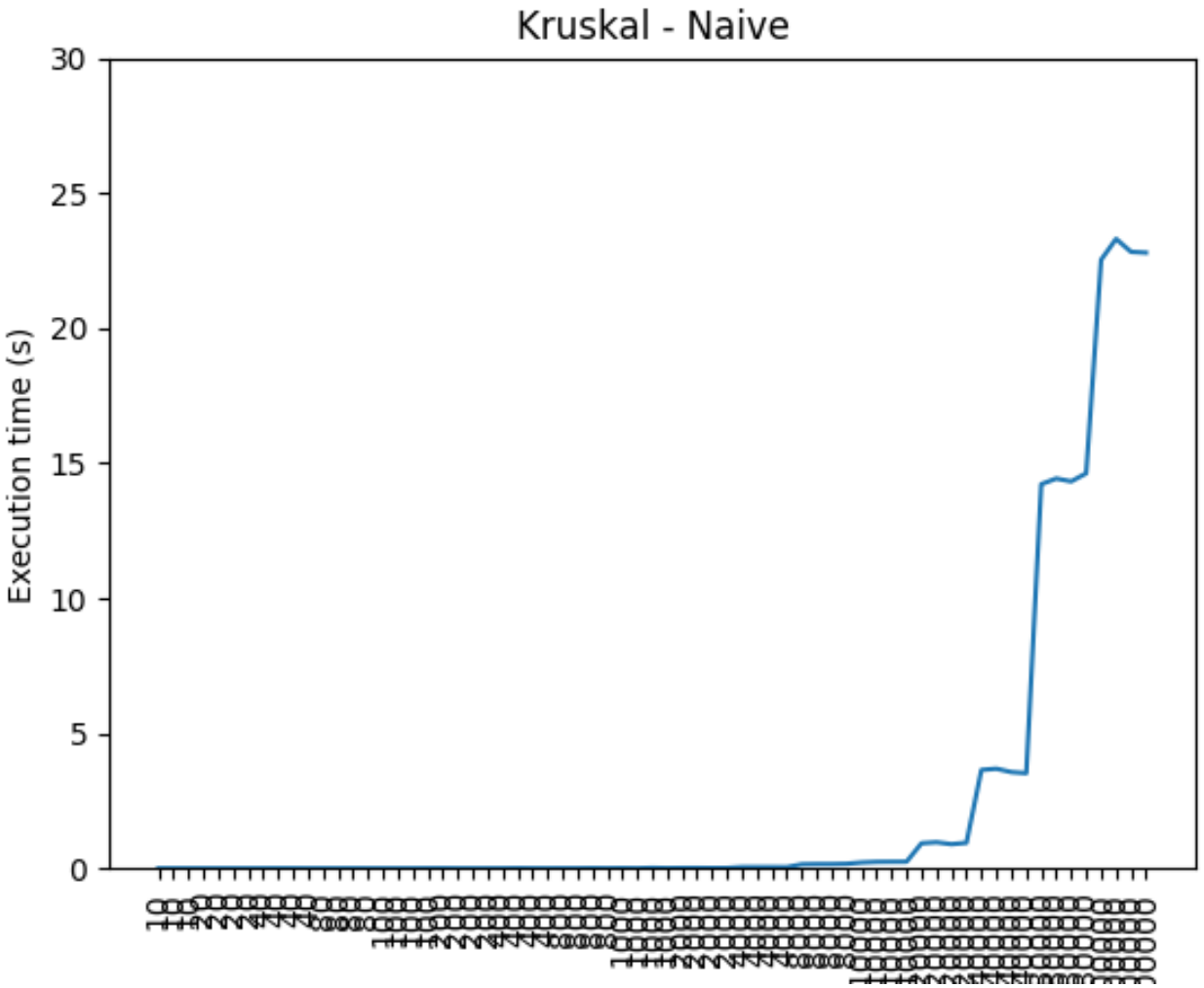


Figure 2: Performance of Kruskal's "naive" Algorithm

## 5.4 Graph of the performance of Kruskal's Union Find Algorithm

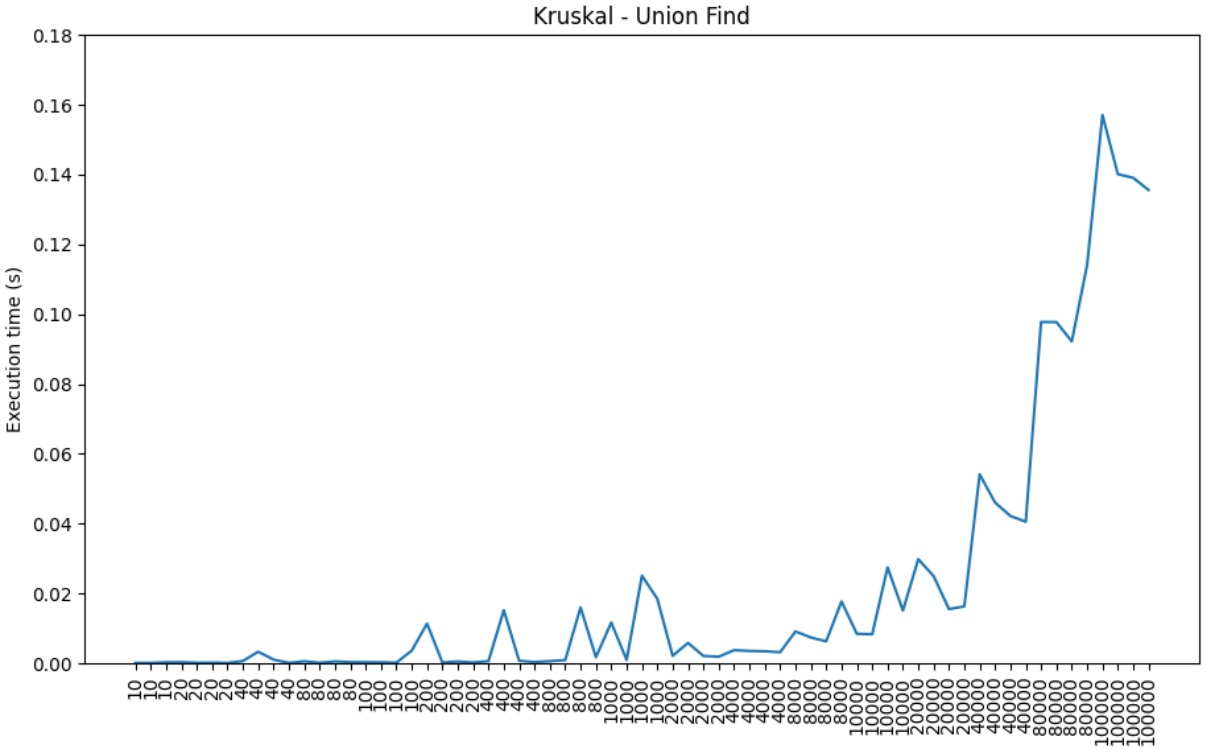


Figure 3: Performance of Kruskal's Union Find Algorithm

# 6 Conclusion

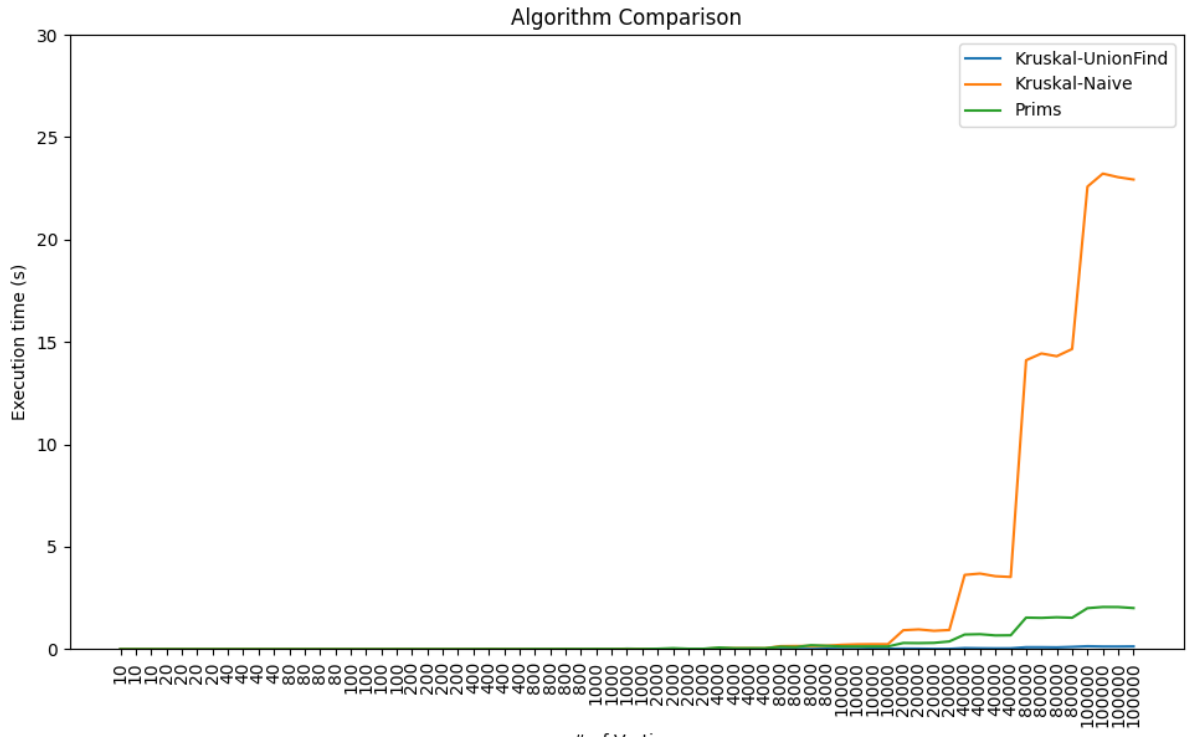


Figure 4: Comparison between the performances of the three algorithms

From Fig. 4 we can observe that for graphs with 1000 vertices or less the performance are more or less the same. When the number of vertices grows, the execution time of the Kruskal's "naive" algorithm grows exponentially and it could even take some minutes to calculate the MST of the bigger graphs of the dataset. The execution time of Prim's and Kruskal's Union Find are very similar. They take a few seconds to calculate the MST of the bigger graphs of the dataset. The results are consistent with the complexity of the algorithms. Kruskal's "naive" is the most complex algorithm, instead the other two have the same algorithmic complexity but the Kruskal's Union Find has an execution time lower than Prim's.