

# Algorithm project

This project includes two tasks related to the sorting and graph algorithms. The goal of the project is to modify and re-implement the Kruskal's algorithm in more efficient way if we assume that the edge weights in the input graphs are integer numbers in the range 0 to  $j$ , for some integer  $j \leq n$  where  $n$  is the number of edges. In addition, you need to compare the time complexity of Kruskal's algorithm before and after the modification. Note: based on the above assumption, apply only one modification on the Kruskal's algorithm presented in the lecture.

## Part 1:

Kruskal's before:

```
Kruskal(G):  
    for each vertex:  
        makeSet(v)  
    sort each edge in non decreasing order by weight using merge sort #O(ElogE)  
    for each edge (u,v):  
        if findSet(u) != findSet(v) :  
            MST = MST + edge(u,v)  
            union(u,v)
```

analysis:

Line 2-3: makeSet() is  $V$   
Line 4: sort the edges is  $O(E \log E)$   
Line 5: for loop is  $O(E)$   
Line 6-8: find and union is  $O(\log V)$

there for complexity is  $O(E \log E + E \log V)$

So, complexity will be  $O(E \log E)$  or  $O(E \log V)$

Kruskal's after:

```
Kruskal(G):  
    for each vertex:  
        makeSet(v)  
    sort each edge in non decreasing order by weight using counting sort #O(E)  
    for each edge (u,v):  
        if findSet(u) != findSet(v) :  
            MST = MST + edge(u,v)  
            union(u,v)
```

analysis:

Line 2-3: makeSet() is  $V$   
Line 4: sort the edges is  $O(E)$

Line 5: for loop is  $O(E)$

Line 6-8: find and union is  $O(\log V)$

there for complexity is  $O(E + E \log V)$

So, complexity will be  $O(E \log V)$  more efficient then  $O(E \log E)$

Algorithms:

makeSet(x){creates a new set whose members is x}

Union(x,y){units the set containing x with the set containing y}

findSet(x){returns a pointer to set containing x}

## Part 2:

Kruskal before:

```
def kruskal_merge(self):
    i,e = 0,0
    ds = dis.disjointset(self.nodes)
    self.graph = merge.merge_sort(self.graph)
    while e<self.V-1:
        w,d,s = self.graph[i]
        i+=1
        x = ds.find(s)
        y= ds.find(d)
        if x!= y:
            e += 1
            self.MST.append( [w,d,s] )
            ds.union(x,y)
    self.display(s,d,w)
```

Kruskal after:

```
def kruskal_counting(self):
    i,e = 0,0
    ds = dis.disjointset(self.nodes)
    self.graph = counting.counting_sort(self.graph)
    while e<self.V-1:
        w,d,s = self.graph[i]
        i+=1
        x = ds.find(s)
        y= ds.find(d)
        if x!= y:
            e += 1
            self.MST.append( [w,d,s] )
            ds.union(x,y)
    self.display(s,d,w)
```

## Table A:

For each algorithm (Kruskal\_before and Kruskal\_after) and each size (50, 100, 150, and 200) of the edges, provide the average running time performed by the algorithm.

Algorithm	E=50 (seconds)	E=100 (seconds)	E=150 (seconds)	E=200 (seconds)	E=1000 (seconds)
Kruskal before	0.003808577 85542806	0.004401286 443074544	0.005856355 03133138	0.006313800 811767578	0.055845657 98441569
Kruskal after	0.003293991 0888671875	0.004252751 6682942705	0.005573272 705078125	0.005383332 57039388	0.054981470 10803223

## Table B:

For each algorithm (Kruskal\_before and Kruskal\_after) and each size (50, 100, 150, and 200) of the edges, provide the best case of the algorithm (case where the minimum running time by the algorithm is reached).

Algorithm	E=50	E=100	E=150	E=200	E=1000
-----------	------	-------	-------	-------	--------

	(seconds)	(seconds)	(seconds)	(seconds)	(seconds)
Kruskal before	0.003472805 0231933594	0.004062891 0064697266	0.005456924 4384765625	0.005738019 943237305	0.055026292 80090332
Kruskal after	0.002917051 315307617	0.003901004 7912597656	0.005280971 527099609	0.005159854 888916016	0.054368019 104003906

## Table C:

For each algorithm (Kruskal\_before and Kruskal\_after) and each size (50, 100, 150, and 200) of the edges, provide the worst case of the algorithm (case where the maximum running time by the algorithm is reached).

Algorithm	E=50 (seconds)	E=100 (seconds)	E=150 (seconds)	E=200 (seconds)	E=1000 (seconds)
Kruskal before	0.003996849 060058594	0.004923105 239868164	0.006061077 117919922	0.006947278 97644043	0.056998014 45007324
Kruskal after	0.003728151 321411133	0.004890203 4759521484	0.005903005 599975586	0.005815029 144287109	0.055659055 70983887

## Results analysis:

### Table A:

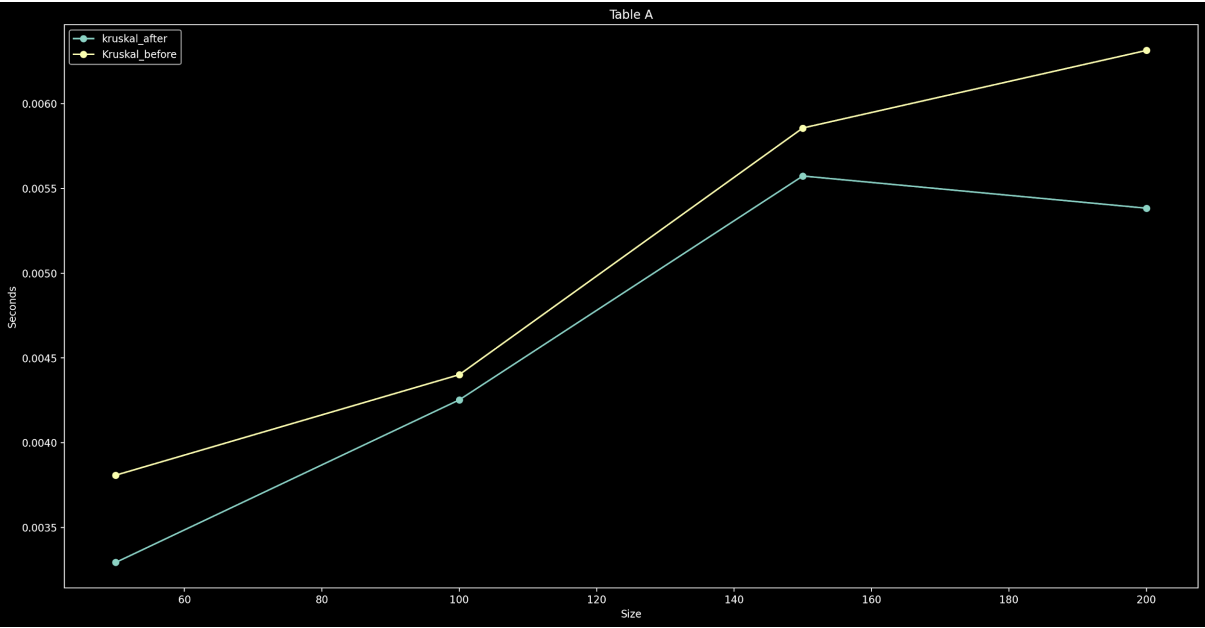


Table B:

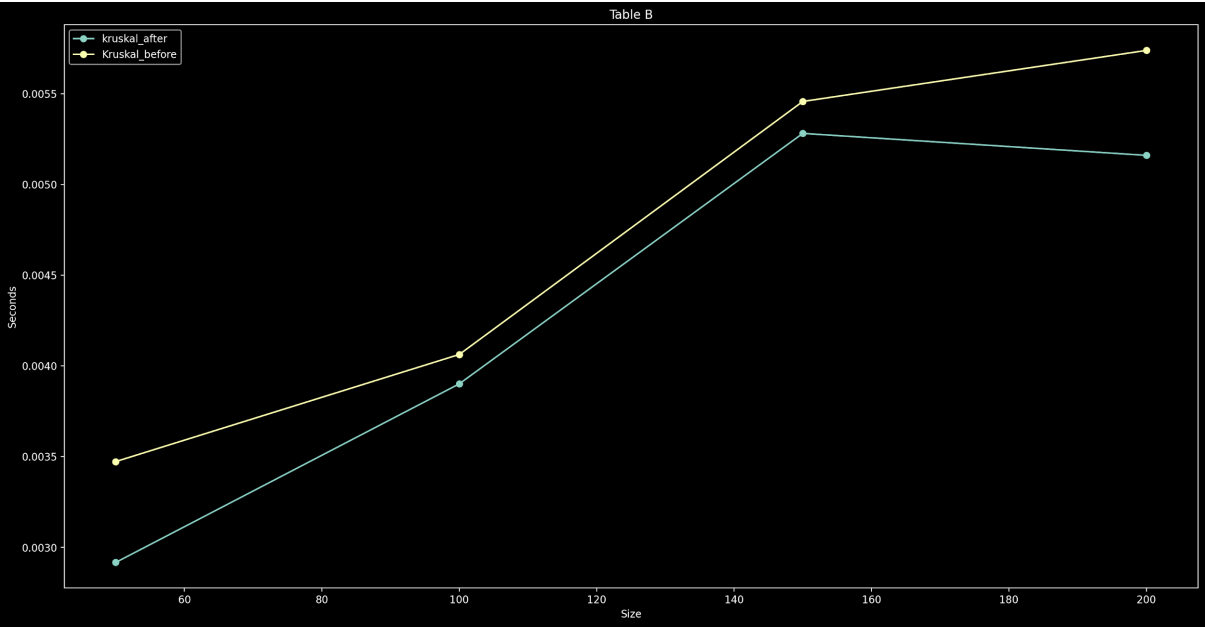
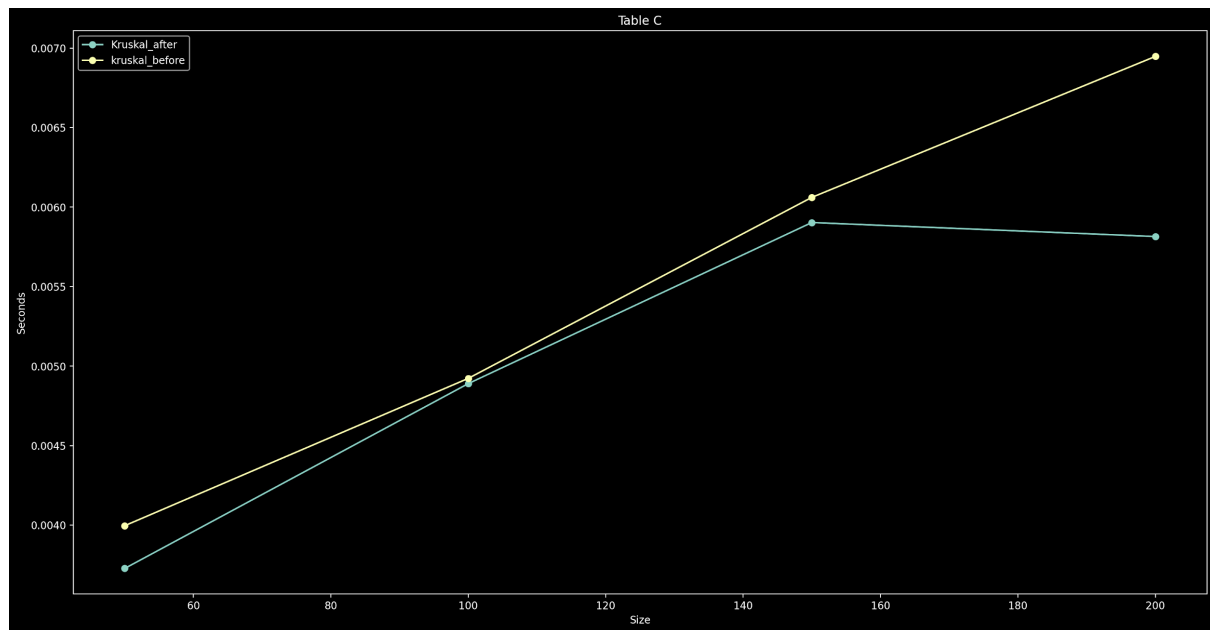


Table C:



## Overall:

As we can see from the Big-O in the theoretical section and the results of the algorithm's testing, storing the data in tables, and analyzing them there to find the improved Kruskal is more efficient