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 <= 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(ElogE)

Line 5: for loop is O(E)

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

there for complexity is O(ElogE + ElogV)

So, complexity will be O(ElogE) or O(ElogV)

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 + ElogV)

So, complexity will be O(ElogV) more efficient then O(ElogE)

Algorithms:

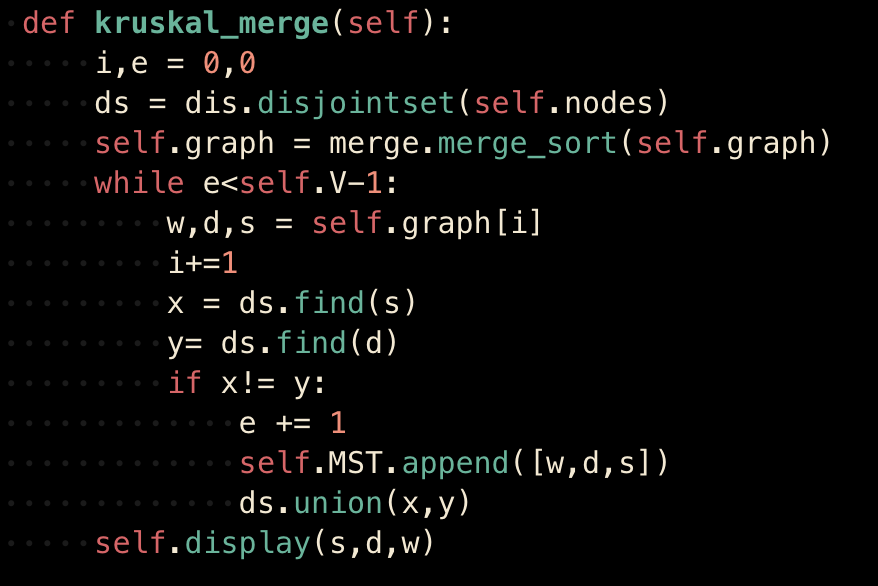
makeSet(x){creates a new set whose only members is x}

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:



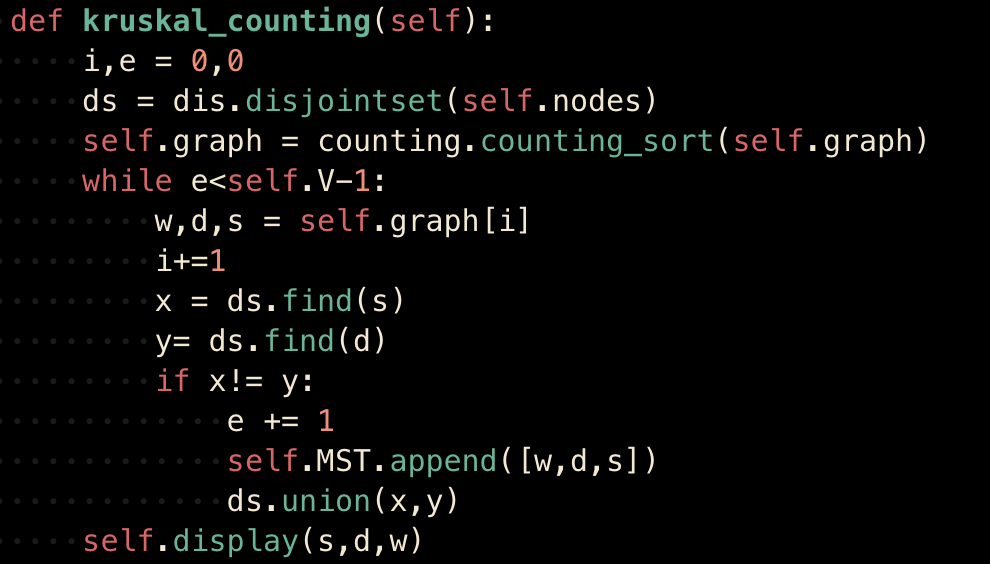
Kruskal after:

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 | 0.00380857785542806 | 0.004401286443074544 | 0.00585635503133138 | 0.006313800811767578 | 0.05584565798441569 |
| Kruskal after | 0.0032939910888671875 | 0.0042527516682942705 | 0.005573272705078125 | 0.00538333257039388 | 0.05498147010803223 |

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  (seconds) | E=100  (seconds) | E=150  (seconds) | E=200  (seconds) | E=1000  (seconds) |
| --- | --- | --- | --- | --- | --- |
| Kruskal before | 0.0034728050231933594 | 0.0040628910064697266 | 0.0054569244384765625 | 0.005738019943237305 | 0.05502629280090332 |
| Kruskal after | 0.002917051315307617 | 0.0039010047912597656 | 0.005280971527099609 | 0.005159854888916016 | 0.054368019104003906 |

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 | 0.003996849060058594 | 0.004923105239868164 | 0.006061077117919922 | 0.00694727897644043 | 0.05699801445007324 |
| Kruskal after | 0.003728151321411133 | 0.0048902034759521484 | 0.005903005599975586 | 0.005815029144287109 | 0.05565905570983887 |

Results analysis:

Table A:

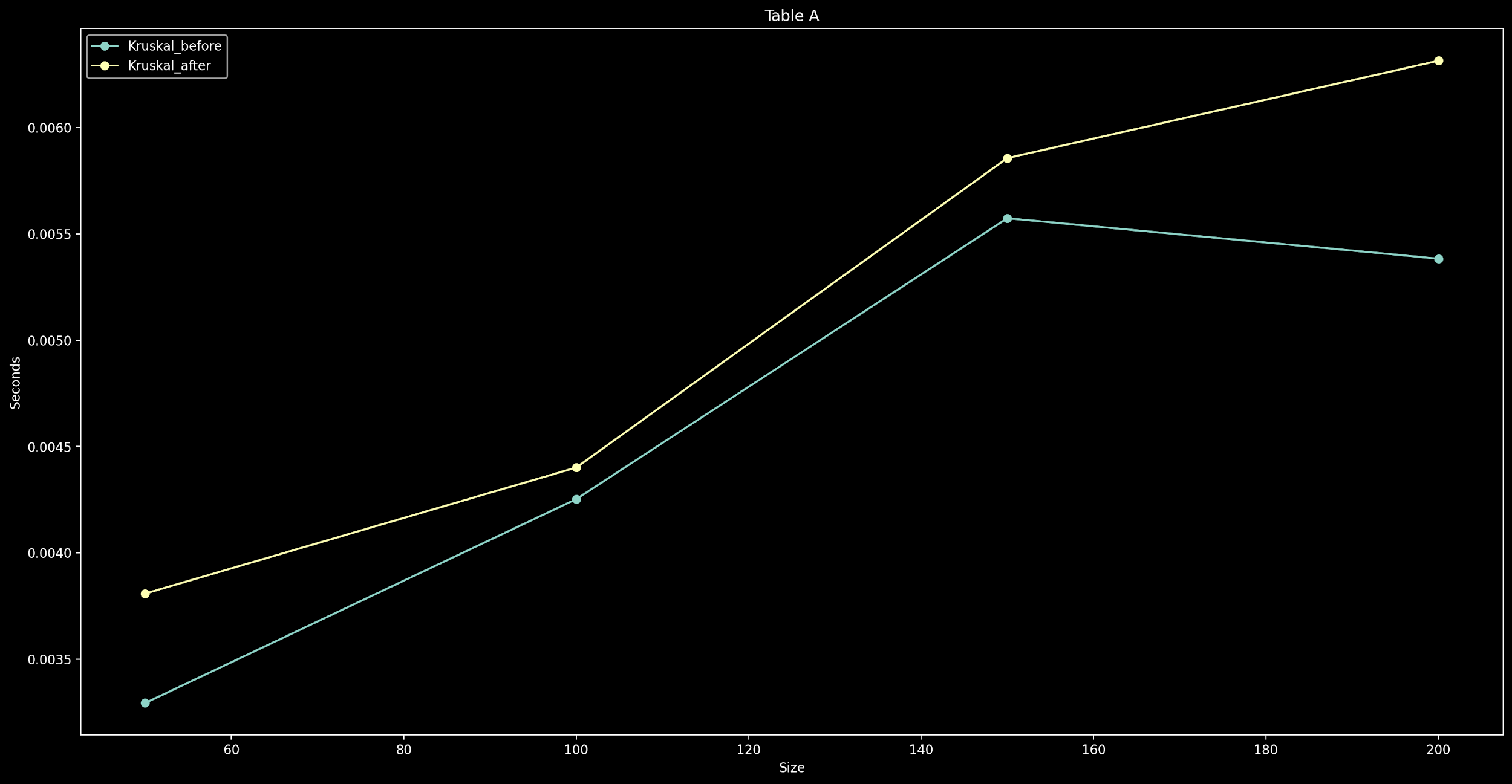


Table B:

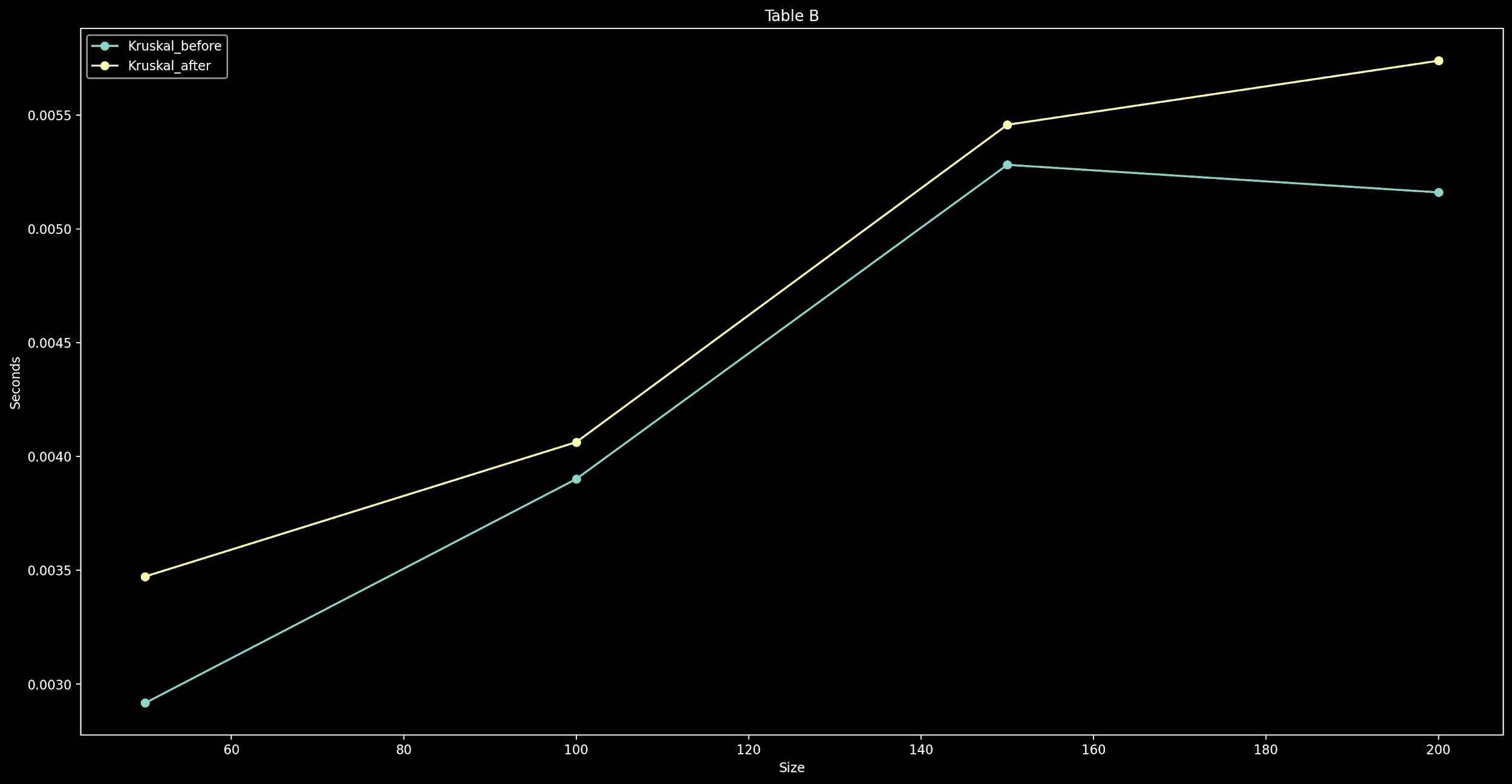
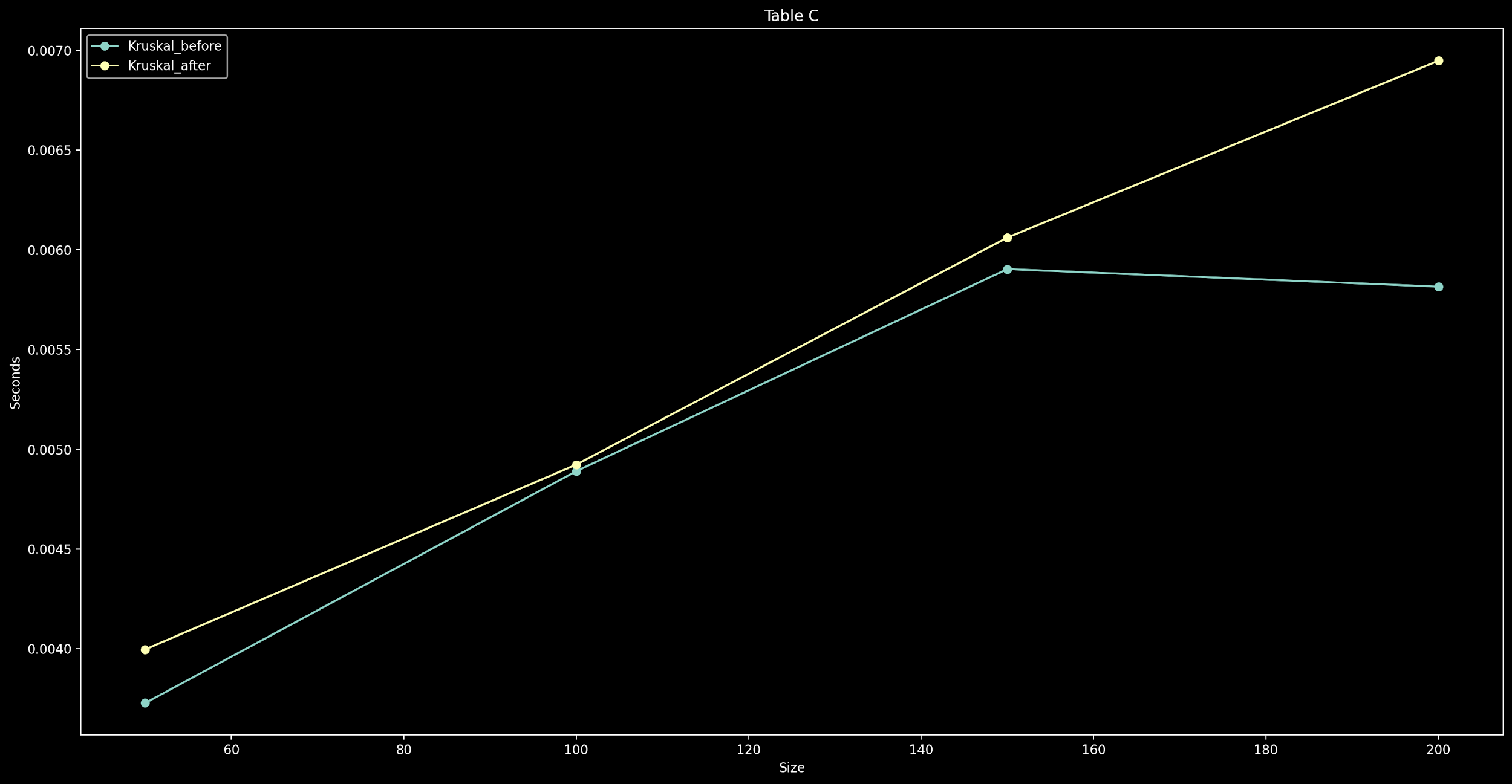


Table C:



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