Overall Organization

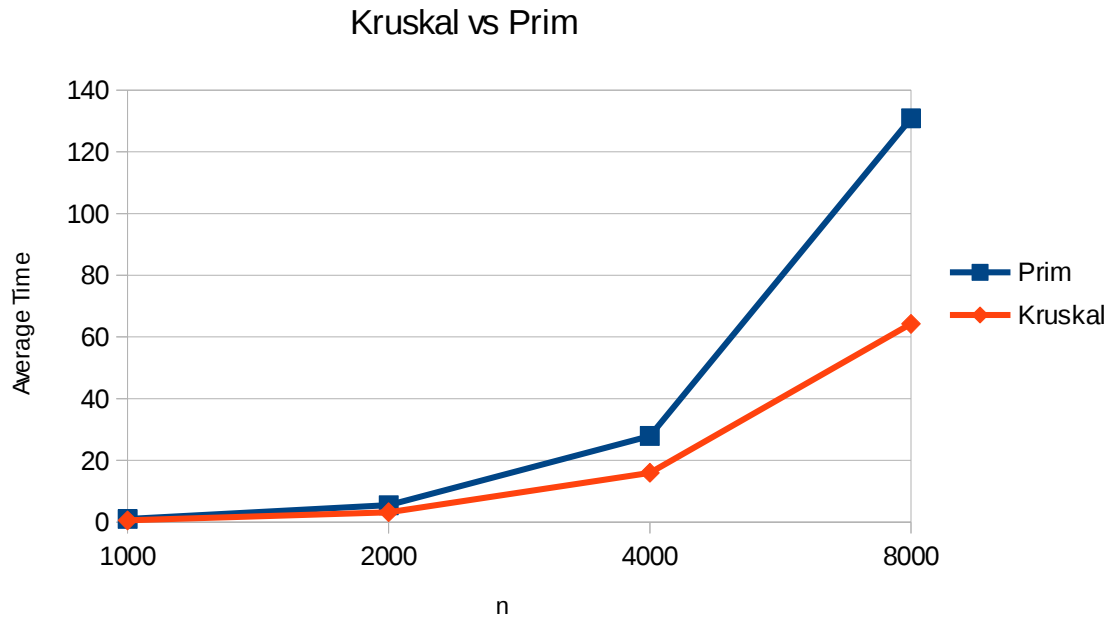
I structured my experiment by using nested for loops. My outermost loop ran 5 times in order to accommodate the 5 trials. Then the next loop would go through the values of n, starting from 1000 going to 8000. The innermost loop would calculate the value for x, then build the graph. After that it would run the algorithm to build the span tree. Timing started after building ended, so it would only measure the amount of time taken by the algorithm itself. Each loop would run with a different seed (1-20) to ensure fairness when testing.

Raw Data

Prim's algorithm for size of 1000: 1.03373
Kruskal's algorithm for size of 1000: 0.543764
Prim's algorithm for size of 1000: 0.995798
Kruskal's algorithm for size of 1000: 0.608036
Prim's algorithm for size of 1000: 1.00526
Kruskal's algorithm for size of 1000: 0.567714
Prim's algorithm for size of 1000: 1.0652
Kruskal's algorithm for size of 1000: 0.617793
Prim's algorithm for size of 1000: 0.986294
Kruskal's algorithm for size of 1000: 0.56766
Prim's algorithm for size of 2000: 5.57974
Kruskal's algorithm for size of 2000: 3.27951
Prim's algorithm for size of 2000: 5.17839
Kruskal's algorithm for size of 2000: 3.081
Prim's algorithm for size of 2000: 5.35839
Kruskal's algorithm for size of 2000: 3.2706
Prim's algorithm for size of 2000: 5.16767
Kruskal's algorithm for size of 2000: 3.09065
Prim's algorithm for size of 2000: 5.57496
Kruskal's algorithm for size of 2000: 3.37339
Prim's algorithm for size of 4000: 26.704
Kruskal's algorithm for size of 4000: 15.0889
Prim's algorithm for size of 4000: 28.9763
Kruskal's algorithm for size of 4000: 16.1269
Prim's algorithm for size of 4000: 26.1756
Kruskal's algorithm for size of 4000: 15.3512
Prim's algorithm for size of 4000: 28.1593
Kruskal's algorithm for size of 4000: 16.4417
Prim's algorithm for size of 4000: 28.2306
Kruskal's algorithm for size of 4000: 15.2703
Prim's algorithm for size of 8000: 130.483
Kruskal's algorithm for size of 8000: 67.234
Prim's algorithm for size of 8000: 132.124
Kruskal's algorithm for size of 8000: 62.152
Prim's algorithm for size of 8000: 128.294

Kruskal's algorithm for size of 8000: 63.246
Prim's algorithm for size of 8000: 129.592
Kruskal's algorithm for size of 8000: 66.998
Prim's algorithm for size of 8000: 130.953
Kruskal's algorithm for size of 8000: 61.487

Prim's avg for 1000: 1.0282
Kruskal's avg for 1000: 0.57219
Prim's avg for 2000: 5.48532
Kruskal's avg for 2000: 3.20243
Prim's avg for 4000: 27.91823
Kruskal's avg for 4000: 15.99287
Prim's avg for 8000: 130.85443
Kruskal's avg for 8000: 64.27589



Summary of Results

Kruskal's algorithm seemed on average to compute twice as fast as Prim's algorithm could. It followed this trend for every single value of n.

Observation and conclusion

It seems that Kruskal's algorithm was quite a bit more efficient than Prim's algorithm, working almost twice as fast. This makes sense as Kruskal connects everything at t global level then works

down, so it runs mostly in O(logN) time, whereas Prim checks through each vertex before connecting the next edge, making it run at O(N^2) time.