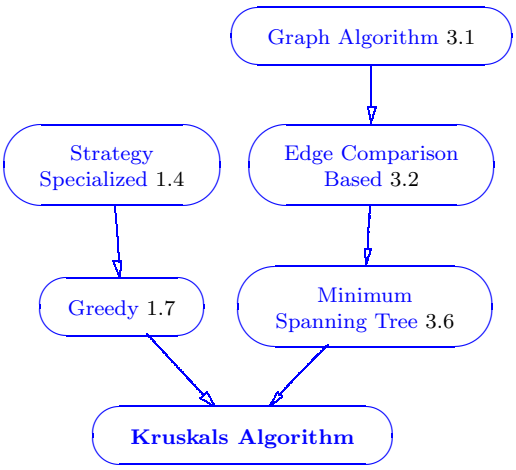
3.6.2 Kruskal's Minimum Spanning Tree

Section authors: Nilanjana De, Nathan Preston, and Bettina Schimanski.



Refinement of: Greedy (§1.7), Minimum Spanning Tree (§3.6). Kruskal's algorithm is a greedy algorithm that uses disjoint sets to solve the Minimum Spanning Tree problem for an undirected weighted graph.

Prototype:
```
template <class Graph, class OutputIterator>
void kruskal_minimum_spanning_trees(
        const Graph& G,
        OutputIterator spanning_tree_edges)
```

Input:

- Explicit:
  - Undirected graph G, which is a model of VertexListGraph and EdgeListGraph.
  - OutputIterator spanning_tree_edges, which are the edges that make up the minimum spanning tree, initially empty. [1]

- Implicit: The graph G must implicitly have the following properties:
  - WeightMap, which maps each edge ∈ E to real values.
  - RankMap, which is used by the disjoint sets data structure. Each disjoint set is identified by a representative, which is some member of the set.
  - Empty PredecessorMap, which is used by the disjoint sets data structure, and is **not** used for storing predecessors in the spanning tree.
  - VertexIndexMap, which maps each vertex ∈ V to an integer in the range from 0 to the number of vertices in G.[1]

Output: Modified OutputIterator spanning_tree_edges, making up a Minimum Spanning Tree (§3.6).

Effects: No effects on the input, other than that mentioned above.

Algorithm Animation: An animation for Kruskal's MST algorithm can be found here.

Asymptotic complexity: Let E = number edges in the graph, and let V = number of vertices in the graph.

- Average case: $O(E \lg E)$

V	E	Time (msec)
1000	2000	1.500
1000	4000	3.000
1000	8000	5.500
1000	16000	15.500
1000	32000	45.600
1000	64000	126.150
1000	128000	322.450
1000	256000	789.600

V	E	Time (msec)
2000	4000	3.000
2000	8000	6.050
2000	16000	16.500
2000	32000	46.100
2000	64000	126.150
2000	128000	321.950
2000	256000	806.650
2000	512000	1937.350
2000	1024000	4369.300

V	E	Time (msec)
3000	4000	3.500
3000	8000	6.500
3000	16000	17.050
3000	32000	47.550
3000	64000	129.650
3000	128000	325.950
3000	256000	810.650
3000	512000	1885.200
3000	1024000	4413.850
3000	2048000	10284.750

V	E	Time (msec)
4000	8000	6.500
4000	16000	17.050
4000	32000	47.050
4000	64000	130.150
4000	128000	335.950
4000	256000	800.650
4000	512000	1929.250
4000	1024000	4495.450
4000	2048000	10321.850

V	E	Time (msec)
5000	8000	6.500
5000	16000	18.550
5000	32000	48.100
5000	64000	131.700
5000	128000	337.500
5000	256000	803.650
5000	512000	1942.300
5000	1024000	4522.500
5000	2048000	10566.700

- Special case
  (complete graph where $E = V^2$): $O(E \lg V)$

Vertices	Edges	Time (msec)
100	10000	7.50
200	40000	50.550
300	90000	140.700
400	160000	298.450
500	250000	515.700
600	360000	954.900
700	490000	1150.150
800	640000	1623.350
900	810000	2107.050
1000	1000000	2613.750

Time Formula in Average Case (μ sec):

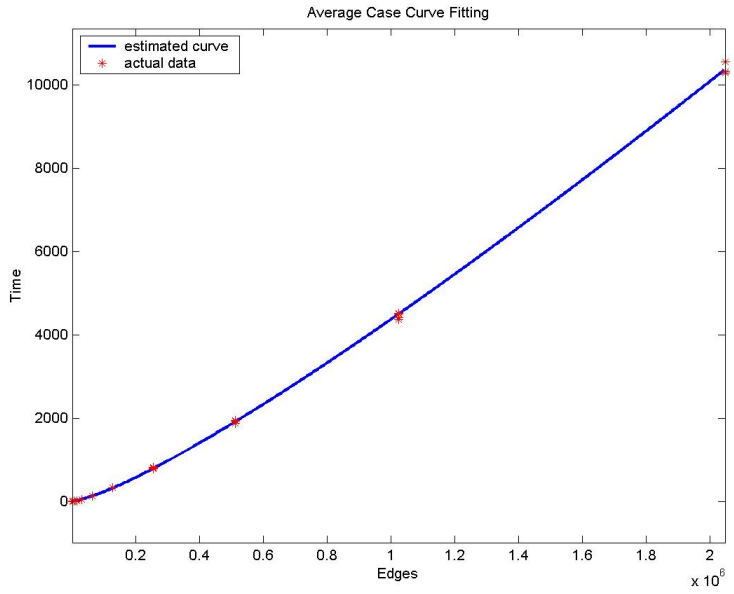
0.69 $E \lg E$ - 9.34 E + 19,141.16

Time Formula in Special Case (μ sec):

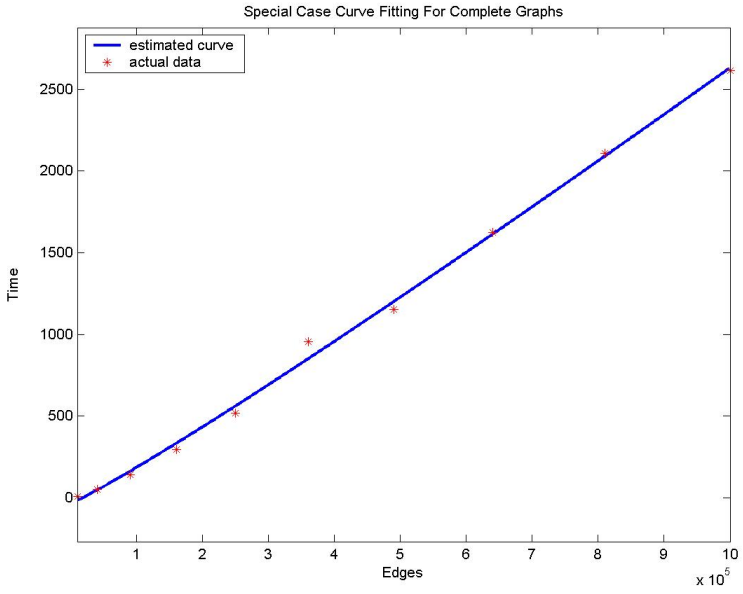
0.29 $E \lg V$ - 0.20 E - 35,080.70

Curve Fitting Diagrams:

Plot of the estimated time calculated w.r.t. the data corresponding to the average case:



Plot of the estimated time calculated w.r.t. the data corresponding to the complete graph case:



Note: The testing above was done using Cygwin on a Windows XP machine with 256 MB of RAM and 1 GHz Pentium III processor.