Student Number: 122415616

For this report, I tested many different densities and numbers of vertices. I initially began with testing 4 different densities:

* 0.05
* 0.5
* 0.75
* 1

With a density of one meaning that every edge in the graph is connected to every other edge in the graph.

I also ran many different numbers of vertices:

* 5
* 10
* 20
* 50
* 100
* 200
* 500
* 1000
* 2500
* 5000

The reason for running with so many different parameters is to get an idea of how both the **Heap APQ** and **Unsorted APQ** perform in Prim’s algorithm. I predict that there will be little difference between the smaller graphs. All these results are measured in seconds.

**Prim’s Algorithm using the Heap APQ.**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 5 | 10 | 15 | 20 | 50 | 100 | 200 | 500 | 1000 | 2500 | 5000 |
| 0.05 | 8.15E-05 | 1.37E-04 | 0.000233 | 0.000218 | 0.0005837 | 0.001642 | 0.005832 | 0.018841 | 0.078545 | 0.424756 | 1.829007 |
| 0.5 | 6.91E-05 | 0.000144 | 0.000439 | 0.000292 | 0.0015975 | 0.005999 | 0.02251 | 0.138475 | 0.561725 | 4.09517 | 17.84041 |
| 0.75 | 1.00E-04 | 0.000168 | 0.000435 | 0.000371 | 0.0020881 | 0.007717 | 0.031658 | 0.211698 | 0.938614 | 6.583297 | 32.18411 |
| 1 | 9.28E-05 | 0.000248 | 0.000285 | 0.000594 | 0.0026013 | 0.011881 | 0.04311 | 0.274505 | 1.231866 | 8.614761 | 41.08493 |

**Prim’s Algorithm using an Unsorted List APQ.**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 5 | 10 | 15 | 20 | 50 | 100 | 200 | 500 | 1000 | 2500 | 5000 |
| 0.05 | 3.38E-05 | 5.00E-05 | 8.17E-05 | 9.76E-05 | 0.0003144 | 0.001114 | 0.004263 | 0.02631 | 0.110593 | 0.671508 | 2.71659 |
| 0.5 | 3.96E-05 | 9.43E-05 | 0.000207 | 0.000288 | 0.0012732 | 0.004817 | 0.019702 | 0.17304 | 0.674456 | 4.295041 | 18.50761 |
| 0.75 | 2.76E-05 | 9.19E-05 | 0.000229 | 0.000272 | 0.001682 | 0.007016 | 0.031022 | 0.249393 | 0.969283 | 6.757123 | 33.95173 |
| 1 | 4.53E-05 | 0.000156 | 0.000201 | 0.000456 | 0.002035 | 0.00956 | 0.041885 | 0.345283 | 1.306038 | 8.919178 | 43.54801 |

As predicted, there’s very little difference between them in smaller graphs. As well, in larger graphs with high density, the difference between the Heap and Unsorted List APQs tends to level out. I believe this is due to the fact that when you use Prim’s Algorithm with a Heap APQ on sparse graphs, is has a time complexity of ***O(n log n),*** however on dense graphs this time complexity jumps to ***O(n^2 log n).*** (where *n* is the number of vertices)

Though I tested and examined many different parameters, I still wasn’t satisfied with the difference between the two implementations. I thought the results didn’t show a large enough difference between the two implementations.

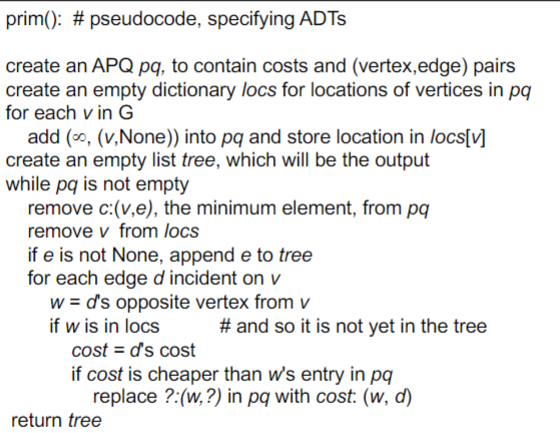
I noticed that the difference seemed to be much larger on **large, sparse** graphs. Say we have *n* vertices and *m* edges, in a sparse graph *m* will be lower than *n,* which means instead of the normal complexity of the heap-based Prim’s algorithm ***O(m log n),*** we can simplify it to ***O(n log n)***. When a list-based Prim’s algorithm takes on sparse graph, it’s generally much less efficient since it depends much more on the number of vertices in graph. It must move through and remove each vertex from the APQ, each time doing this taking O(n), done *n* times and it’s ***O(n^2)***.

I went about more testing on sparse graphs, and as predicted the difference between a **Heap-based** Prim’s algorithm and **list-based** Prim’s algorithm is much larger. These are all done on a **minimum density connected graph**.

|  |  |  |
| --- | --- | --- |
|  | **Heap-based Prim’s** | **Unsorted-list Prim’s** |
| 5000 vertices | 0.120s | 1.435s |
| 10,000 vertices | 0.247s | 6.496s |
| 15,000 vertices | 0.402s | 15.052s |

When there are many edges in the graph (a dense graph), the two implementations become much closer in performance since they must loop through every edge in the graph, which becomes exponentially larger with higher densities.

However, the **Heap-based algorithm** is still slightly faster even on very dense graphs, I believe this is due to how much more efficient it is to remove a minimum element from the APQ with a heap-based implementation.

Within the first loop, the **heap-based** implementation is much faster, as it’s much more efficient at removing the minimum value from the APQ. In the nested loop down at the bottom, who’s efficiency is based on the number of edges, the **unsorted list** implementation can catch up to the heap-based one, however it can only catch up in efficiency if there are enough edges to slow them both down. Hence the two implementations are much closer on dense graphs, and one sparse graphs, the heap is much more efficient.