ANALYSIS AND DESIGN OF ALGORITHMS

Term Project

Habeeba Sakr - 900183103

1) This is the ranking code I used:

```
    for (auto const& i : webgraph) {
        for (auto const& j : i.second) {
            for (auto const& l : pagerank)
            {
          //   std::cout << i.first << " -> " << j.first << j.second<< std::endl;
                if(l.first == i.first)
                    count[l.first]++;


            }
        }
    }

  for (int k=0; k<3; k++)
  {
    for (auto const& i : webgraph) {
            for (auto l : pagerank)
            {


                l.second = l.second /count[l.first];
                cout<<l.second;

            }



  }
```

First you check how many webpages you have and we divide the page ranks of each element by this number initially.

There are a few loops that are a bit confusing, as they loop over maps, however, I will try to simplify.

The first one loops over the webpage map and the map inside it, and the pagerank map.

We increment a counter if found in the sources of the graph (as it means that it is a source more than once).

The second loop is for the page rank iterations. We do 3 iterations. We loop over the webgraph and the pagerank and do this calculation for every webpage "l.second = l.second /count[l.first];"

2) Time complexity for ranking would be O(k*n^3)

Space complexity would be O(1)

3) My algorithm relied mainly on unordered maps to store most things, including results, keywords, web pages and even the directed graph. Unordered maps have really good access times, and they were beneficial to me especially in the directed graph as I had an unordered map of string and another unordered map of string to store the multiple destinations to the same source.

4) I assumed 3 iterations for the page rank which might reduce accuracy if we have a bigger graph.