**Rawan Ali**

**900192037**

**Analysis and Design of Algorithms lab project report.**

**The pseudo code of the page rank:**

for (i = 0 to i < 4) oldPR.push\_back(0.0);

for(i=0 to i<100){ count = -1;

for (auto& x : g)

{ count2 = 0; count++;

if (count3 = 0)

rankStep initialized by 0.25;

else rankStep = oldPR[count];

for (auto& it : x.second)

{ it2 = g.find(the url);

size = it2->second.size(); //vector size

if (count2 = 0 & count3=0)

rank.push\_back((d \* rankStep) / size);

else rank[count] = rank[count] + ((d \* rankStep) / size); count2++;

}

rank[count] += ((1 - d) / 4); oldPR[count] = rank[count];} count3++;}

for (i = 0 to i < rank.size)

pagerank.insert((PR[i], rank[i]));

**The pseudo code of indexing:**

if (found("AND"))

{

pos = keyword.find("AND"); s1 = keyword.substr(0, pos-1); s2 = keyword.substr(pos + 4);

for (auto x : keywords)

{

it = keywords.find(x.first);

s = it->second;

if ((found s1) & (found s2)

results.push\_back(it->first);

}

}

else if (( found OR) || ((not found “”) && (not found AND)))

{ if (found OR)

{ pos = keyword.find("OR"); s1 = keyword.substr(0, pos - 1); s2 = keyword.substr(pos 3);

}

else if (keyword.find(" ") != -1)

{ pos = keyword.find(" "); s1 = keyword.substr(0, pos); s2 = keyword.substr(pos + 1);

}

for (auto x : keywords)

{ it = keywords.find(x.first);

s = it->second;

if (( found s1 ) || (found s2))

results.push\_back(it->first);

}

} else

{ for (auto x : keywords)

{

it = keywords.find(x.first);

vector<string> s = it->second;

erase ('"');

if (found keyword)

results.push\_back(it->first);

}

}

**The complexity of indexing algorithm:**

It is used to retrieve the search query from the user and determine if the user inserted OR, AND, quotations, or plain. To make the program perform its operations I used the find function in each step to determine the search type the user inserted and it takes the word before AND or OR and the word after it. Its complexity is O(n) where n is the number of words the user inserted.

**The complexity of the page rank algorithm:**

Page rank algorithm shows the importance of a page according to their link to other webpages. The webpage of the higher rank has the higher priority to be sorted first, and the lowest rank will be shown at the end. This will take a complexity of O(n+m) where n is the nodes (key in the map) and m are the size of the vector in front of each key.

**The main data structure used in the project is Maps**

I used maps in my code instead of graphs because graphs are more complex and are not preferably to be used in the page rank. Maps consist of a key value pair, and the values are stored in an array. I used maps because they are more manageable than graphs to deal with many types. Most of my maps’ types are maps of string and int for the site and the number of clicks, impressions, and page score. I used maps also in my searching and indexing processes.

**Tradeoffs implemented in the project:**

At the beginning of the project I used vectors however I found that maps but I found that I will use many of them so I used maps instead of them.