Pseudo Code for Algorithms:

```
1)  Init_page_ranks(Graph &g) //function which calculates the page rank

    -   Float sum = 0; //to store sum of all PR’s of all pages
    -   For i: 1 to number of webpages //loop 1
            o   webpage[i].pr = 1/n;
    -   for i: 1 to 2: //loop 2
            o   for i: 1 to number of pages //loop 3
                    ▪   graph temp = transpose(g);
                    ▪   vector<int> nodes_pointing_to_curr =
                        get_nodes_pointing_to_v(current node);
                    ▪   for i: 1 to number of nodes pointing to curr //this loop calculates
                        the pr for each node (loop4)
                            •   pr = 0;
                            •   temp_pr = prev_pr / number of nodes going out of current
                                node
                            •   page_rank += temp_pr
                    ▪   page_rank *= 0.85; //0.85 is the damping factor
                    ▪   page_rank = page_rank + (  (1-0.85) / number of webpages); //rest
                        of the formula for pr
                    ▪   push the page rank along with the id into a new vector
                    ▪   sum+= current page rank
    -   norm_pr = sum / number of pages;

2)  new_search() //process search query

    -   take in input string
    -   find(‘ “ ‘); //look for  quotations
    -   if (quotations found)
            o   if (word exits)
                    ▪   remove quotations from the word and add it to the keywords
                        array
                    ▪   call display_results(keywords[])
    -   else if (“AND”) exists //AND case
            o   push the first word into the keywords array
            o   push the second word into the keywords array
            o   push “AND” into the keywords array
            o   call display_results(keywords[])

    -   else if (“OR”) exists //OR case
            o   push the first word into the keywords array
```

```
        o  push the second word into the keywords array
        o  push “AND” into the keywords array
        o  call display_results(keywords[])
    -   else //two words next to each other treating them like “OR” case
        o  parse the words from the string into the keywords array
        o  call display_results(keywords[])

3)  display_results(keywords[])

    -   if (last word in the keywords array is “AND”)
        o  loop over the map containing the id of the page and all the keywords it
           contains
            ▪  if (word1 && word2 stored in map[webpage])
                •  calculate_score(current webpage);
                •  add this page with its score to the “pages_to”display”
                   array
                •  increase the impression count for this webpage //since it
                   will be viewed
    -   else //two OR cases
        o  for i: 1 to number of keywords passed (2)
            ▪  loop over map containing the keyword and all the pages storing it
               //”keys” in search engine class
                •  get the score and name of the that webpage
                •  add it to the pages_to_display array
                •  increase the impression count
    -   sort(pages to display) //by score
    -   call display_webpage_choice(pages_to_display[])

4)  display_webpage_choice(vector<pair<float, string>> pages_to_display) //takes in a
    vector containing all the pages to be displayed, each pair element is the score and
    name of that page

    -   display “webpage choice” menu
    -   if (open new webpage)
        o  ask for webpage number
        o  increase clicks for that page
        o  display contents of webpage
    -   else if (new_search)
        o  return to new search
    -   else if (exit)
        o  save_impressions_csv() //updates the number of impressions in the file
        o  exit program
```

Time and Space Complexity of Algorithms:

1) init_page_ranks()

//let n be the number of pages, the number of nodes pointing to a node be m, and the size of an adjacency list element be r

Time Complexity:

- first loop O(n)
- second loop: 2
    - third loop: loop over all pages O(n)
        - fourth loop: loop over nodes pointing to current pages O(m)
            - for each node, loop over its adj list O(r)
- total = n + 2*n*m*r = O(nmr)

Space Complexity:

- Graph: O(n^2) //adj list
- Transpose graph: O(n^2)
- Array of webpages: O(n)
- Array of nodes pointing to current: O(m)

Total: O(n^2)

2) new_search()

//let x be the length of the string passed, and s be the number of total keywords in the program

Time Complexity:

- Find quotations: O(x) //
    - Loop over keywords in map O(s)
        - Loop over pages in the map O(n)
- Find “AND”: O(x)
    - Add first word and second word and “AND” to array O(1)
- Find “OR”: O(x)
    - Add first word and second word and “AND” to array O(1)
- Last Case:
    - Add first word and second word and “AND” to array O(1)
- Total = O(x)

Space Complexity:

- Vector storing keywords O(n)

3) display_results()

Time Complexity:

- loop over map containing webpages O(n)
  - for each webpage loop over keywords to find the keywords passed 2*O(m)
- calculate score, add to pages to display array, and increase impression count O(1)
- Sort pages by score O(nlogn)
- Total = O(nm) + O(nlogn) = O(nm)

Space Complexity:

- Map 1 storing the webpage and all its keywords O(n)
- Map 2 storing the keywords and all its webpages O(m)
- Vector storing pages to display O(n)
- Since keywords >= number of webpages, total space complexity = O(m)

4) display_webpage_choice(vector<pair<float, string>> pages_to_display)

Time Complexity:

- increase clicks and display webpage: O(1)
- call save_impressions_csv() O(n) since we loop over the map containing the impression for each page
- total: O(n)

Space Complexity:

- Map to store impressions O(n)
- Total = O(n)

Main DS:

1) Map to store keywords and all of the pages that have them “**keys**” in search engine class
2) Map to id’s of pages and all the keywords it contains “**index**” in search engine class
3) Vectors (too many to count).. used all over
4) Vectors of pairs to store pages to display and their scores
5) Graph to store the webpages (for page ranking algorithm)

Design Tradeoffs:

- In theory, the search engine class was unnecessary and the array of webpages it contains is not needed. I could have simply used numerous maps in the main function, and it would’ve gotten the job done. However, for cleaner and more structured code, I decided to create this class. I used the idea of “encapsulation” and “abstraction” here from OOP concepts to try to link everything and store everything in the search engine class. Note: it is more of a “webpage repository” rather than a search engine. I just named it search engine class.