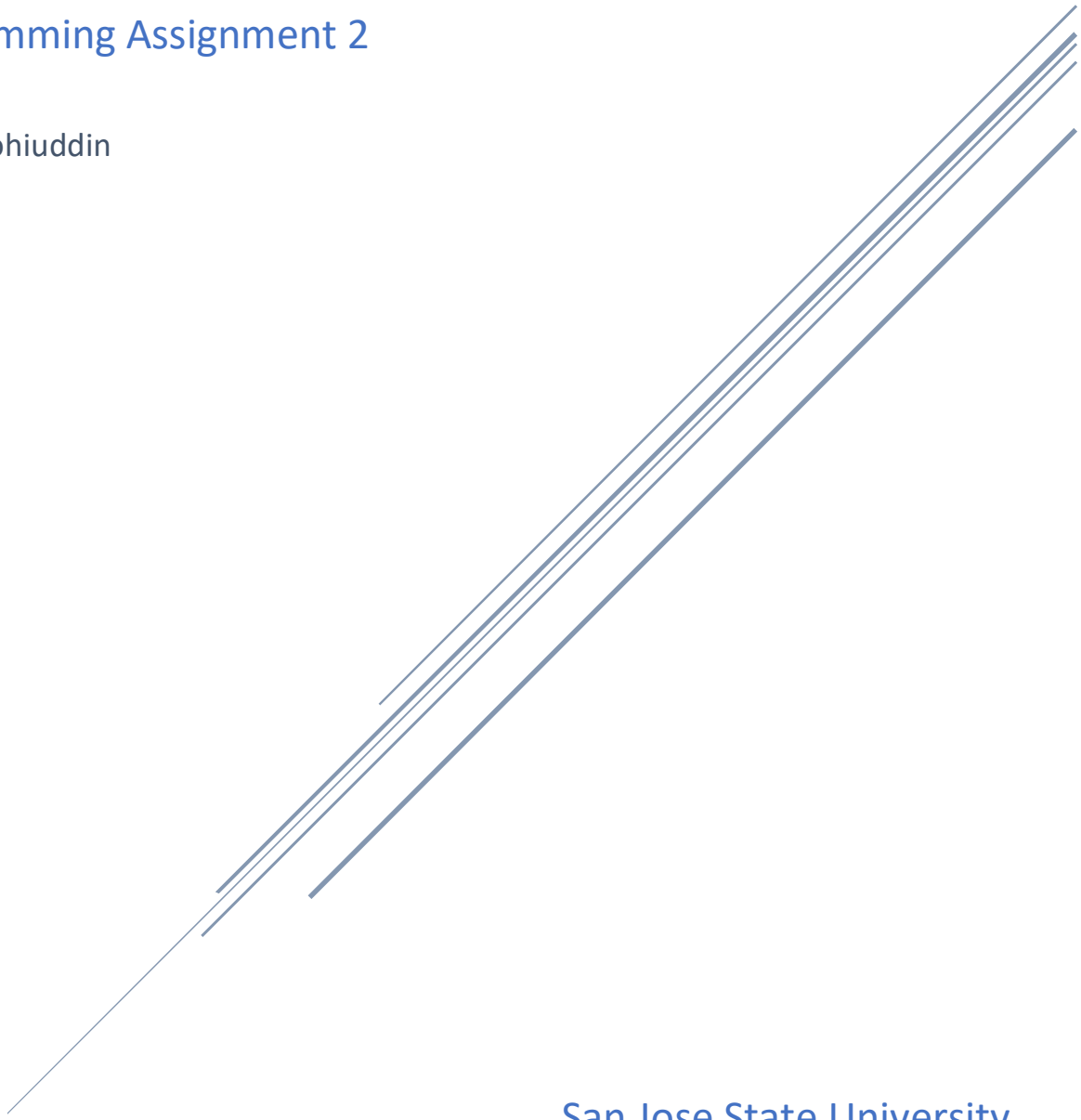
GOOGLE SEARCH ENGINE SIMULATOR USING BINARY SEARCH TREES, QUICKSORT, AND BUCKETSORT

Programming Assignment 2

Rabia Mohiuddin



San Jose State University
CS 146-S8

Table of Contents

How to Run Application

1. Unzip application
2. Open a terminal or console window
3. Change directories to the bin folder of the unzipped project
4. Execute java -jar BuildApp.jar
5. Try all options to view complete functionality of application

Implementation Overview

The Google Search Engine Simulator allows a user to conduct a search of a term they desire with eight main sub functions. Once the search is submitted, a list of 30 results are displayed to the user along with options on what to do next. These options include:

1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search

The Simulator consists of seven java classes and one interface: BST, Quicksort, BucketSort, FunnyCrawler, BuildApp, WebPageURL, SearchTerms, and Rank Interface.

The BuildApp class uses all classes (BST, Quicksort, BucketSort, FunnyCrawler, BuildApp, WebPageURL, SearchTerms).

BST, Quicksort, and BucketSort use Rank Interface for their ArrayList.

Test Cases

1. View search results from Quicksort (displays original search results)

```
Welcome to my mock Google search simulator!
Author: Rabia Mohiuddin

Enter a search term: google
Your term 'google' found 39 results
What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 1


Rank          Total Score         Index              URL
1                       46                2             https://www.blog.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghaMBE&usg=AOvVaw2rMG_Tp-wXFLpbcNYSl0Fz
2                       45                20            https://www.google.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFggVMAA&usg=AOvVaw2vnGOsAJSS77ai01o7fKBe
3                       44                22            https://domains.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghEMA0&usg=AOvVaw2mg82IT3DnLXdsvk3XdOyi
4                       44                16            https://diversity.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiAATAY&usg=AOvVaw2RLZtVFKu-8u9scsmG6iW-
5                       43                3             https://twitter.com/google%3Flang%3Den&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgg4MAs&usg=AOvVaw3zW0MuGELGNq6JaHJKb8
6                       40                13            https://www.blog.google/products/maps/see-your-messages-local-businesses-google-maps/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoM
D2AQFghvMBU&usg=AOvVaw2QWHkB_btejNBnIQ0fEGqw
7                       37                1             https://translate.google.com/m/translate&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQjBAIHjAE&usg=AOvVaw2MnUHlAncgfOJ5C_O
8                       33                24            https://www.theverge.com/google&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiFATAZ&usg=AOvVaw3-_OJ1P5CWuBE7WpkauMxQ
9                       33                14            https://www.cnbc.com/2018/11/16/kurian-has-to-overcome-a-bitter-feud-between-google-and-oracle.html&sa=U&ved=0ahUKEwjHh
reAhVsjoMKHcadD2AQpwIIKg&usg=AOvVaw14sTcMgJQU_m45gKM1UU-r
10                      32                35            https://techspective.net/2018/03/14/5-ways-the-google-logo-has-changed-over-its-20-year-history/&sa=U&ved=0ahUKEwjHhI7Y
hVsjoMKHcadD2AQndQBCMoBMCM&usg=AOvVaw3NgaiuBKdTRLcC1gaV5sGT
11                      32                15            https://www.cnn.com/videos/business/2018/11/16/is-google-a-monopoly.cnn-business&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcad
AIIrgEwIFAB&usg=AOvVaw0Pa_pEgPr_Tx-J3uiCmt73
12                      32                11            https://www.cnn.com/videos/business/2018/11/16/is-google-a-monopoly.cnn-business&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcad
wIIrQEwIA&usg=AOvVaw3w4WuZoZDR2d0e5SIngtIV
13                      32                6             https://www.facebook.com/Google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghkMBM&usg=AOvVaw3Mll3ISi-Yj0hpLgw8WlfZ
14                      31                21            https://www.blog.google/products/pixel/see-light-night-sight/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgh1MBY&usg=AOv
ZXvgHV7-WWGCAVBnWnbP
15                      30                25            https://www.google.ru/maps&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgihATAe&usg=AOvVaw0_9i6r-NIKKvxkpkzcchJ5
16                      29                34            https://www.google.es/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiWATAc&usg=AOvVaw1lUCcLXhHa494DU_22u0AO
17                      28                10            https://www.blog.google/products/assistant/create-your-smart-home-holidays-google-assistant/&sa=U&ved=0ahUKEwjHhI7Y39re
oMKHcadD2AQFghpMBQ&usg=AOvVaw1hlqPZGC8rjQUbp7r93AKP
18                      28                8             https://www.google.com.vn/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiwATAh&usg=AOvVaw1tO8rExwwfjzVaW40Ixvl4
19                      27                33            https://www.youtube.com/Google&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghVMBA&usg=AOvVaw1AjZJd4xooFWjbdHXMVVLP
20                      26                39            http://ai.googleblog.com/2018/11/night-sight-seeing-in-dark-on-pixel.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgi
```

Figure 1: Original search results from Quicksort

```
/**
 * Quicksort original search results from URLObjects by Total score
 *
 * @param none
 * @return none
 */
public void quicksortResults() {
    Quicksort qsort = new Quicksort();                  // Instantiate quicksort object
    qsort.quicksort(URLObjects, 0, URLObjects.size() - 1);          // Call quicksort on URLobjects from index 0 to size
}
```

Figure 2: Calling quicksort on the URLObjects array

2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data

```
What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 2


Rank            Total Score         Index               URL
1                       46                  2               https://www.blog.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghaMBE&usg=AOvVaw2rMG_Tp-wXFLpbcNYSl0Fz
2                       45                  20              https://www.google.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFggVMAA&usg=AOvVaw2vnGOsAJ5577ai01o7fkBe
3                       44                  22              https://domains.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghEMA0&usg=AOvVaw2mg82IT3DnLXdsvk3XdOyi
4                       44                  16              https://diversity.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiAATAY&usg=AOvVaw2RLZtVFKu-8u9scsmG6iW-
5                       43                  3               https://twitter.com/google%3Flang%3Den&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgg4MAs&usg=AOvVaw3zW0MuGELGNq6JaHJKb8i5
6                       40                  13              https://www.blog.google/products/maps/see-your-messages-local-businesses-google-maps/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcad
D2AQFghvMBU&usg=AOvVaw2QWHkB_btejNBnIQ0fEGqw
7                       37                  1               https://translate.google.com/m/translate&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQjBAIHjAE&usg=AOvVaw2MnUHlAncgfOJ5C_O4aXtZ
8                       33                  24              https://www.theverge.com/google&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiFATAZ&usg=AOvVaw3-_OJ1P5CWuBE7WpkauMxQ
9                       33                  14              https://www.cnbc.com/2018/11/16/kurian-has-to-overcome-a-bitter-feud-between-google-and-oracle.html&sa=U&ved=0ahUKEwjHhI7Y39
reAhVsjoMKHcadD2AQpwIIKg&usg=AOvVaw14sTcMgJQU_m45gKM1UU-r
10                      32                  35              https://techspective.net/2018/03/14/5-ways-the-google-logo-has-changed-over-its-20-year-history/&sa=U&ved=0ahUKEwjHhI7Y39reA
hVsjoMKHcadD2AQndQBCMoBMCM&usg=AOvVaw3NgaiuBKdTRLcC1gaV5sGT
11                      32                  15              https://www.cnn.com/videos/business/2018/11/16/is-google-a-monopoly.cnn-business&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQu
AIIrgEwIFAB&usg=AOvVaw0Pa_pEgPr_Tx-J3uiCmt73
12                      32                  11              https://www.cnn.com/videos/business/2018/11/16/is-google-a-monopoly.cnn-business&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQt
wIIrQEwIA&usg=AOvVaw3w4WuZoZDR2d0e5SIngtIV
13                      32                  6               https://www.facebook.com/Google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghkMBM&usg=AOvVaw3Mll3I5i-Yj0hpLgw8WlfZ
14                      31                  21              https://www.blog.google/products/pixel/see-light-night-sight/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgh1MBY&usg=AOvVaw10
ZXvgHV7-WWGCAVBnWnbP
15                      30                  25              https://www.google.ru/maps&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgihATAe&usg=AOvVaw0_9i6r-NIKKvxkpkzcchJ5
16                      29                  34              https://www.google.es/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiWATAc&usg=AOvVaw1lUCcLXhHa494DU_22u0AO
17                      28                  10              https://www.blog.google/products/assistant/create-your-smart-home-holidays-google-assistant/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsj
oMKHcadD2AQFghpMBQ&usg=AOvVaw1hlqPZGC8rjQUbp7r93AKP
18                      28                  8               https://www.google.com.vn/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiwATAh&usg=AOvVaw1tO8rExwwfjzVaW40Ixvl4
19                      27                  33              https://www.youtube.com/Google&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghVMBA&usg=AOvVaw1AjZJd4xooFWjbdHXMVVLP
20                      26                  39              http://ai.googleblog.com/2018/11/night-sight-seeing-in-dark-on-pixel.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiKATAa
&usg=AOvVaw1m-jBPFkqjGFIm30MSnIN4
21                      25                  17              https://android-developers.googleblog.com/2018/11/an-update-on-project-treble.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2A
QFgiQATAb&usg=AOvVaw0PwNGZmp39yndiMSCwTFDZ
22                      24                  7               https://www.google.com/docs/about/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQjBAIIDAC&usg=AOvVaw3TYQv8pyfQePckIs9Eq_CO
23                      24                  5               https://www.cnbc.com/2018/11/16/google-san-jose-campus-documents-no-subsidies.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2A
QqQIIMDAJ&usg=AOvVaw0vDVDMPQ4n_WpMi4XLDJ5r
24                      23                  28              https://www.google.org/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFggyMAo&usg=AOvVaw2Ct0kozOcUT0RMOGLR2_kg
```

Figure 3: Binary Search Tree results

```
/**
 * public search method that retrieves search string from user, calls search and retrieves results, then builds BST, and stores the results if first
 * time being searched
 *
 * @param searchTerm
 *          String -> search term
 * @return none
 */
public void enterSearch(String searchTerm) {
    addSearchTerm(searchTerm);              // Given searh term user entered, call method to add term to the search term heap

    URLObjects.clear();             // Clear the public URLObjects ArrayList to ensure no old results left behind

    URLObjects = search(searchTerm);       // Get new search results and store them in public ArrayList

    System.out.println("Your term  '" + searchTerm + "' found " + URLObjects.size() + " results");          // Print # of results to user

    for (Rank urlobj : URLObjects) {                // For every Rank (url) object in the ArrayList, insert it into BST by creating new node
            urlBST.treeInsert(new Node(urlobj));              // Create new node with Rank object, insert in BST
    }
```

Figure 4: URL BST is created once search results come back

3. Search for a specific page rank

```
33                      19              12              https://maps.google.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQjBAIIjAF&usg=AOvVaw1dRTSqjuywtOig0BSq9yB8
34                      18              9               https://www.cnbc.com/2018/11/16/google-san-jose-campus-documents-no-subsidies.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2A
QFgg-MAw&usg=AOvVaw3LRfVCwK9KaS0wVlCMdfR9
35                      16              30              https://deepmind.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgimATAf&usg=AOvVaw3j7a8P_jIfra2h5ibZ_iGN
36                      15              23              https://en.wikipedia.org/wiki/Google&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghKMA4&usg=AOvVaw0vlAUUAjc4ae72VFk2vvww
37                      15              4               https://news.google.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQjBAIJjAG&usg=AOvVaw1xz2jXo6JyImTc_6q_UjET
38                      12              31              https://developer.android.com/about/versions/pie/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgicATAd&usg=AOvVaw1TCm6VME2gCzn
2EQuQF_YL
39                      9               19              https://learndigital.withgoogle.com/digitalgarage&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgi2ATAi&usg=AOvVaw3Y5ZtVvvExnTh
8jHXenVfR

What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 3
Which Rank item would you like to display?
There are currently 39URLs

Search URL Rank: 36


Rank            Total Score         Index               URL
36                      15              23              https://en.wikipedia.org/wiki/Google&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghKMA4&usg=AOvVaw0vlAUUAjc4ae72VFk2vvww
```

Figure 5: Searching for page rank 36 (shown at top of screenshot).

```
case "3":       // Search for a specific page rank
        System.out.println("Which Rank item would you like to display?");
        int size = urlBST.getSize();            // Get size (number of nodes) of current BST
        System.out.println("There are currently " + size + "URLs");          // Print size of BST to the user
        System.out.print("\nSearch URL Rank: ");       // Ask user to select a URL rank
        try {
            int rankNode = Integer.parseInt(reader.nextLine());          // Read in user rank choice
            if (rankNode > size || rankNode < 0) {              // Make sure choice is within the BST size
                    throw new InputMismatchException("Out of range");          // If not in BST size, throw exception
            }
            Node rankItem = urlBST.search(rankNode - 1);            // Retrieve the node returned from searching for rank
            printHeader();          // Print header columns and print object attributes with given format
            System.out.format("%-25d%-20d%-15d%-15s\n", rankItem.key.getRanking(), rankItem.key.getTotalScore(), rankItem.ke

        } catch (InputMismatchException e) {            // Exception thrown if not in range or not integer
            System.out.println(e.getMessage());             // Print message to the console
        } catch (NumberFormatException e) {             // If not integer, catch exception
            System.out.println("Invalid rank - " + e.getMessage());             // Print message to the console
        }
        break;
```

Figure 6: How chosen rank is searched for and returned to user

4. Insert a URL (based on Total Score) into search results (BST)

```
What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 4
Enter a URL that starts with http:// or https://
URL: http://rabia.com
Total Score of URL: 50

What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 2


Rank            Total Score         Index               URL
1                       50              39              http://rabia.com
2                       46              2               https://www.blog.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghaMBE&usg=AOvVaw2rMG_Tp-wXFLpbcNYSl0Fz
3                       45              20              https://www.google.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFggVMAA&usg=AOvVaw2vnGOsAJSS77ai01o7fKBe
4                       44              22              https://domains.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghEMA0&usg=AOvVaw2mg82IT3DnLXdsvk3XdOyi
5                       44              16              https://diversity.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiAATAY&usg=AOvVaw2RLZtVFKu-8u9scsmG6iW-
6                       43              3               https://twitter.com/google%3Flang%3Den&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgg4MAs&usg=AOvVaw3zW0MuGELGNq6JaHJKb8i5
7                       40              13              https://www.blog.google/products/maps/see-your-messages-local-businesses-google-maps/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcad
```

Figure 7: Inserting http://rabia.com with score 50 (page rank 1)

```java
/**
 * Inserts a new node into BST while maintaining BST properties. Updates ranking of each node
 *
 * @param Node
 *            newNode
 * @return none
 */
public void treeInsert(Node newNode) {
    Node parent = null;             // Parent of root is null
    Node iterator = this.root;      // Start iterator at root of tree
    while (iterator != null) {      // Iterate through nodes till find null
            parent = iterator;      // Store parent to compare new node with children
            if (newNode.compareTo(iterator) == -1) {    // If new node is less than iterator
                iterator = iterator.left;               // Go to left subtree
            } else {            // If new node is greater than iterator
                iterator = iterator.right;              // Go to right subtree
            }
    }
    newNode.parent = parent;        // New node's parent is parent of null node
    if (parent == null) {           // If Tree was empty
            this.root = newNode;            // Set root of tree to newNode
    } else if (newNode.compareTo(parent) == -1) {       // If newNode is less than its parent
            parent.left = newNode;          // Set parent's left node to newNode
    } else {                // If newNode is greater than its parent
            parent.right = newNode;         // Set parent;s right node to newNode
    }
    updateRanking();    // Update ranking of each node
}
```

Figure 8: Insert a new node into BST and update rankings

5. Delete a URL (based on Page Rank) from search results (BST

```
What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 5
Which Rank item would you like to delete?
There are currently 40URLs

Delete URL Rank: 1
Rank item 1 was deleted

What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 2


Rank          Total Score        Index              URL
1                   46               2             https://www.blog.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghaMBE&usg=AOvVaw2rMG_Tp-wXFLpbcNYSl0Fz
2                   45               20            https://www.google.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFggVMAA&usg=AOvVaw2vnGOsAJSS77ai01o7fKBe
3                   44               22            https://domains.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghEMA0&usg=AOvVaw2mg82IT3DnLXdsvk3XdOyi
4                   44               16            https://diversity.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiAATAY&usg=AOvVaw2RLZtVFKu-8u9scsmG6iW-
5                   43               3             https://twitter.com/google%3Flang%3Den&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgg4MAs&usg=AOvVaw3zW0MuGELGNq6JaHJKb8i5
```

Figure 9: Delete node based on rank (Rank 1 – http://rabia.com). Results show rank 1 changed.

6. View the top 10 searches of the day

```
What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 6
Top 10 search terms today:
google : 1 times
```

Figure 10: Show search terms and number of times searched

```
/**
 * Retreive top 10 searches and print to the user
 *
 * @param none
 * @return none
 */
public void getTop10searches() {
    System.out.println("Top 10 search terms today: ");        // Header to print to the user

    BST bstTopSearches = new BST();              // Create bst object to sort searchOccurrences
    for (Rank searchOcc : searchOccurrences) {           // For every rank object in searchOccurrences
            bstTopSearches.treeInsert(new Node(searchOcc));        // Create a node from Rank and insert into BST for top searches
    }

    top10searchTerms.clear();            // Clear top 10 search terms arraylist in case populated

    for (int i = 0; i < 10 && i < searchOccurrences.size(); i++) {       // Iterate 10 times or size of searchOccurrences to extract top 10
            top10searchTerms.add(bstTopSearches.search(i).key);        // add to top10searches ArrayList
    }

    print(top10searchTerms);             // Print top 10 searches and their attributes to the user

}
```

Figure 11: Insert top searches into BST and print

7. View search results from first search today (Will display in URL alphabetical order)

```
What would you like to do next?
1. View search results from Quicksort (displays original search results)
2. View search results from Binary Search Tree Inorder Walk (displays any modifications to data)
3. Search for a specific page rank
4. Insert a URL (based on Total Score) into search results (BST)
5. Delete a URL (based on Page Rank) from search results (BST)
6. View the top 10 searches of the day
7. View search results from first search today (Will display in URL alphabetical order)
8. Run another search
0. Quit

Option: 7


Rank          Total Score         Index               URL
20                      26                  39              http://ai.googleblog.com/2018/11/night-sight-seeing-in-dark-on-pixel.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiKATAa
&usg=AOvVaw1m-jBPFkqjGFIm30MSnIN4
21                      25                  17              https://android-developers.googleblog.com/2018/11/an-update-on-project-treble.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2A
QFgiQATAb&usg=AOvVaw0PwNGZmp39yndiMSCwTFDZ
1                       46                  2               https://www.blog.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghaMBE&usg=AOvVaw2rMG_Tp-wXFLpbcNYSl0Fz
17                      28                  10              https://www.blog.google/products/assistant/create-your-smart-home-holidays-google-assistant/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsj
oMKHcadD2AQFghpMBQ&usg=AOvVaw1hlqPZGC8rjQUbp7r93AKP
6                       40                  13              https://www.blog.google/products/maps/see-your-messages-local-businesses-google-maps/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcad
D2AQFghvMBU&usg=AOvVaw2QWHkB_btejNBnIQ0fEGqw
14                      31                  21              https://www.blog.google/products/pixel/see-light-night-sight/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgh1MBY&usg=AOvVaw10
ZXvgHV7-WWGCAVBnWnbP
31                      19                  37              https://classroom.google.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQjBAIJDAD&usg=AOvVaw2s9Qyq86a9uzB-XhVud9pe
34                      18                  9               https://www.cnbc.com/2018/11/16/google-san-jose-campus-documents-no-subsidies.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2A
QFgg-MAw&usg=AOvVaw3LRfVCwK9KaS0wVlCMdfR9
23                      24                  5               https://www.cnbc.com/2018/11/16/google-san-jose-campus-documents-no-subsidies.html&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2A
QqQIIMDAJ&usg=AOvVaw0vDVDMPQ4n_WpMi4XLDJ5r
9                       33                  14              https://www.cnbc.com/2018/11/16/kurian-has-to-overcome-a-bitter-feud-between-google-and-oracle.html&sa=U&ved=0ahUKEwjHhI7Y39
reAhVsjoMKHcadD2AQpwIIKg&usg=AOvVaw14sTcMgJQU_m45gKM1UU-r
25                      23                  27              https://www.cnbc.com/2018/11/16/kurian-has-to-overcome-a-bitter-feud-between-google-and-oracle.html&sa=U&ved=0ahUKEwjHhI7Y39
reAhVsjoMKHcadD2AQqQIILDAH&usg=AOvVaw1oMDw3fNcmeUbc7DYXaRnX
12                      32                  11              https://www.cnn.com/videos/business/2018/11/16/is-google-a-monopoly.cnn-business&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQt
wIIrQEwIA&usg=AOvVaw3w4WuZoZDR2d0e55IngtIV
11                      32                  15              https://www.cnn.com/videos/business/2018/11/16/is-google-a-monopoly.cnn-business&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQu
AIIrgEwIFAB&usg=AOvVaw0Pa_pEgPr_Tx-J3uiCmt73
35                      16                  30              https://deepmind.com/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgimATAf&usg=AOvVaw3j7aBP_jIfra2h5ibZ_iGN
32                      19                  29              https://design.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghfMBI&usg=AOvVaw3IhWXual_qridTK0GCZRa4
38                      12                  31              https://developer.android.com/about/versions/pie/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgicATAd&usg=AOvVaw1TCm6VME2gCzn
2EQuQF_YL
4                       44                  16              https://diversity.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFgiAATAY&usg=AOvVaw2RLZtVFKu-8u9scsmG6iW-
3                       44                  22              https://domains.google/&sa=U&ved=0ahUKEwjHhI7Y39reAhVsjoMKHcadD2AQFghEMA0&usg=AOvVaw2mg82IT3DnLXdsvk3XdOyi
```

Figure 12: Displays first search today results in alphabetical order

```
public void enterSearch(String searchTerm) {
    addSearchTerm(searchTerm);              // Given searh term user entered, call method to add term to the search term heap

    URLObjects.clear();             // Clear the public URLObjects ArrayList to ensure no old results left behind

    URLObjects = search(searchTerm);        // Get new search results and store them in public ArrayList

    System.out.println("Your term  '" + searchTerm + "' found " + URLObjects.size() + " results");          // Print # of results to user

    for (Rank urlobj : URLObjects) {                // For every Rank (url) object in the ArrayList, insert it into BST by creating new node
            urlBST.treeInsert(new Node(urlobj));                // Create new node with Rank object, insert in BST
    }

    if (searchOccurrences.size() == 1 && searchOccurrences.get(0).getTotalScore() == 1) {
            for (Rank obj : URLObjects) {
                firstSearchResults.add(obj.clone());
            }
            bucketSort.bucketSort(firstSearchResults);              // Bucketsort the cloned list
    }
}
```

Figure 13: First search results are set once there is one term in search occurrences and the first element in search occurrences has a total score of 1. Clones objects to keep original search results.

Classes and Methods

BST Class

The purpose of the BST class is to provide functionality for searching an ArrayList of objects of type Rank. BST implements all Binary Search Tree required methods but instead of being dependent on integers, nodes are filled with Rank objects.

Variables

- **root** – private Node that holds pointer to the top of the Binary Search Tree

Methods

- **public Node getRoot**() – Retrieves root of tree.
  - Input – None
  - Output – Node of root of BST
  - Algorithm – Returns the root of tree.
  - Complexity – O(1)
- **public void inorderTreeWalk**(Node node) – BST in order walk that prints largest to smallest by going through the right side of tree first, then left side
  - Input – Node of tree
  - Output – None
  - Algorithm – Recursive function that iterates to right side of tree first, then left in order to print in order of largest to smallest node.
  - Complexity – O(n)
- **public Node treeMinimum**(Node node) - BST smallest node in tree
  - Input – Node of tree
  - Output – Smallest node in tree
  - Algorithm – Iterates through tree to keep going left till node has no left child.
  - Complexity – O(h)
- **public Node treeMaximum**(Node node) - BST largest node in tree
  - Input – Node of tree
  - Output – largest node in tree
  - Algorithm – Iterates through tree to keep going right till node has no right child.
  - Complexity – O(h)
- **public Node treeSuccessor**(Node node) - Finds the next largest smallest value in the tree
  - Input – Node of tree
  - Output – largest smallest node in tree
  - Algorithm – Find minimum value of the node's right subtree.
  - Complexity – O(h)
- **public void treeInsert(**Node newNode) - Inserts a new node into BST while maintaining BST properties. Updates ranking of each node.
  - Input – new Node want to insert into tree
  - Output – None

    - Algorithm – Iterate through tree to find where to place new node while maintaining BST properties. Then call updateRanking() on tree.
    - Complexity – O(h)
- **public void transplant**(Node u, Node v) - Transplant subtrees of two given nodes
    - Input – Two nodes whose subtrees will be transplanted
    - Output – None
    - Algorithm – Find minimum value of the node's right subtree.
    - Complexity – O(2)
- **public void treeDelete**(Node delete) - Deletes a given node from BST while maintaining BST properties. Updates ranking of each node
    - Input – Node want to delete
    - Output – None
    - Algorithm – Check if node has a left child, right child, or both and based on that transplant the delete node with the left, right, or minimum node's subtree
    - Complexity – O(h)
- **private int size**(Node subtree) – Finds and returns the size of a given Node's subtree
    - Input – Node who's subtree's size want to get
    - Output – size of tree (int)
    - Algorithm – Recursively return current node (size 1) + size of left subtree + size of right subtree
    - Complexity – O(n)
- **public int getSize**() – Retrieves size of entire tree
    - Input – None
    - Output – size of tree from root
    - Algorithm – Calls size(bst.root)
    - Complexity – O(n)
- **public void updateRanking**() – Search for rankings 0 to size of tree and set ranking of Rank object
    - Input – None
    - Output – None
    - Algorithm – Iterates through size of tree and searches for a specific rank, then sets the found object's rank
    - Complexity – O(n)
- **public Node search**(int k)– Search for the node at a specific ranking
    - Input – integer of ranking
    - Output – Node found at ranking
    - Algorithm – Checks input to ensure is in the tree, then calls select helper function to retrieve Node.
    - Complexity – O(n)
- **public Node select**(Node x, int k)– Get k'th largest node in tree

    - Input – integer of ranking, Node of root
    - Output – Node found at ranking
    - Algorithm – Iterate through left and right subtrees based on if k is less than or greater than size of subtree.
    - Complexity – O(n)

Node Class

The purpose of this class is to encase the Rank object and hold Node attributes for the Binary Search Tree.

Variables

- **Key** – Rank object
- **Parent** – Node pointer to parent
- **Left** – Node pointer to left child
- **Right** – Node pointer to right child

Methods

- **public Node**(Rank obj) – Construct node with Rank object
    - Input – Rank object
    - Output – None
    - Algorithm – Set Rank object to key. Parent, left, and right are set to null.
    - Complexity – O(1)
- **public int compareTo**(Node o) – Compare two nodes by comparing their Rank
    - Input – Node object
    - Output – integer showing less than, greater than, or equal to
    - Algorithm – Compare keys of both nodes.
    - Complexity – O(1)

Quicksort Class

The purpose of the Quicksort class is to sort an ArrayList of Rank objects using the Quicksort sorting algorithm according to the Node's total score.

Variables

None

Methods

- **public void quicksort**(ArrayList<Rank> array, int first, int last) – Partitions and sorts Rank objects by total score
    - Input – ArrayList of Rank objects, index of first element, index of last element
    - Output – None
    - Algorithm – Call partition and quicksort on the first half and second half of the array.
    - Complexity – O(n logn)

- **private int partition**(ArrayList<Rank> array, int first, int pivotIndex) – Compares every element to partition element and swaps to correct order
    - Input – ArrayList of Rank objects, index of first element, index of pivot element
    - Output – index of pivot
    - Algorithm – Compare elements to pivot value and swap.
    - Complexity – O(n)

BucketSort Class

The purpose of the BucketSort class is to sort Rank objects by the URL instead of the total score into buckets. The buckets are then sorted and concatenated to reform a list.

Variables

None

Method

- **public void bucketSort**(ArrayList<Rank> array) – Bucket Sorts URLs into buckets and calls insertion sort on each bucket
    - Input – ArrayList of Rank objects
    - Output – None
    - Algorithm – Sort Rank objects into calculated buckets and call Insertion sort on each bucket.
    - Complexity – O(n+k)
- **public String getHostName**(String url) – Returns name of url without http or www
    - Input – URL string
    - Output – hostname string
    - Algorithm – Get hostname by creating URI object and checking if url begins with "www".
    - Complexity – O(1)
- **private void insertion_sort**(ArrayList<Rank> vals) – Insertion sort each bucket by url (name)
    - Input – ArrayList of Rank objects
    - Output – None
    - Algorithm – Compares each element to a key and swaps to sort in order.
    - Complexity – O(n)

FunnyCrawler Class

The purpose of the FunnyCrawler class is to take a search term and send a request to Google's search engine and return a unique set of results from that query.

** Source: http://www.mkyong.com/java/jsoup-send-search-query-to-google/ **

Variables

None

Methods

- **public** Set<String> **getDataFromGoogle**(String query) – Uses the entered query to send a request to Google and return a results set of links.
    - Input – query: String the user wants information on
    - Output – Set<String>: Set of unique URL strings that match the query
    - Algorithm – Uses a Jsoup http request protocol object to connect to Google and retrieve all href links
    - Complexity – O(n)

Rank Interface

The purpose of the Rank interface is allow the HeapSort class to build and sort heaps of both WebPageURL and SearchTerms objects. This interface implements some basic methods that will be used in the two classes that implement the Rank interface.

Variables

None

Methods

- **public int getTotalScore**() – Retrieves total rank for WebPageURL or occurrence for SearchTerms
    - Input – None
    - Output – score (integer)
- **public int getIndex**() – Retrieves index for WebPageURL or occurrence for SearchTerms
    - Input – None
    - Output – score (integer)
- **public void setRanking**(int r) – Set ranking used for URL objects
    - Input – ranking integer
    - Output – None.
- **public int getRanking**() – Get ranking for URL objects
    - Input – None
    - Output – ranking integer
- **default public int compareTo**(Rank other) – Compare two rank objects by total score
    - Input – Rank object
    - Output – integer showing less than, greater than, or equal to
- **public void printAttributes**() – Prints attributes of object
    - Input – None
    - Output – None; Prints attributes to screen.
- **public** String **getName**() – Retrieve URL name or search term name
    - Input – None
    - Output – Name: String of either search term or URL
- **public void increase**(**int** value) – Increase value of object (either total rank or search occurrence)

- Input – value: integer of how much want to increase object value by.
- Output – None.

WebPageURL Class implements Rank

The purpose of the WebPageURL class is to store the URL string as well as four rankings of the site which include frequency the link has been visited, site age, how many times the link has been referenced, and amount of money paid to sponsor the site and bring it to the top of the search results. Also included are the original index and current ranking in terms of other URL objects.

Variables

- **frequency** – private integer variable of frequency a site is visited
- **siteAge** – private integer variable of age of the site
- **linkReference** – private integer variable of number of times site has been referenced on other webpages
- **moneyPaid** – private integer variable of amount of money paid to increase rank
- **URL** – private string variable of URL string
- **Index** – private integer that stores the original index the result query was received at.
- **Ranking** – private integer storing URL rank in relation to other URLs

Methods

- **public WebPageURL**() – Basic generic constructor that creates object without any input fields that assigns four minimum values for frequency, siteAge, linkReference, and moneyPaid. Used when creating space for a new node in HeapSort class.
  - Input – None
  - Output – None
  - Algorithm – Generates four minimum values for variables using Integer.*MIN_VALUE*
  - Complexity – O(1)
- **public WebPageURL**(String uRL, int index) – Constructor given URL that creates object and generates four random values for variables.
  - Input – uRL: String of url that is given by results of web crawler (FunnyCrawler)
  - Output – None
  - Algorithm – Uses Math.random() to generate four values for frequency, siteAge, linkReference, and moneyPaid.
  - Complexity – O(1)
- **public WebPageURL**(**int** key) – Constructor given key that is set to moneyPaid variable. Defaults to generic string for URL and assigns four random values.
  - Input – key: integer that is set to moneyPaid variable of object
  - Output – None
  - Algorithm - Uses Math.random() to generate three values for frequency, siteAge, linkReference but assigns key to moneyPaid.

    - Complexity – O(1)
- **public int getFrequency**() - Retrieves value of private frequency variable
    - Input – None
    - Output – frequency: integer that is of private frequency variable
    - Algorithm – returns frequency site shows up in searches
    - Complexity – O(1)
- **public int getSiteAge**() - Retrieves value of private siteAge variable
    - Input – None
    - Output – siteAge: integer that is of private siteAge variable
    - Algorithm – returns age of site
    - Complexity – O(1)
- **public int getLinkReference**() - Retrieves value of private linkReference variable
    - Input – None
    - Output – linkReference: integer that is of private linkReference variable
    - Algorithm – returns number of times link has been referenced
    - Complexity – O(1)
- **public int getMoneyPaid**() - Retrieves value of private moneyPaid variable
    - Input – None
    - Output – moneyPaid: integer that is of private moneyPaid variable
    - Algorithm – returns amount of money paid
    - Complexity – O(1)
- **public int getIndex**() - Retrieves value of private index variable
    - Input – None
    - Output – index: integer that is of private index variable
    - Algorithm – returns index
    - Complexity – O(1)
- **public void setRanking**(int r) – Sets private ranking variable
    - Input – r: integer to set ranking
    - Output – None
    - Algorithm – sets input r to ranking variable
    - Complexity – O(1)
- **public int getRanking**() - Retrieves value of private ranking variable
    - Input – None
    - Output – ranking: integer that is of private ranking variable
    - Algorithm – returns ranking
    - Complexity – O(1)
- **public void increase**(**int** value) – Increased value of money paid to value inputted
    - Input – value: integer of amount of money being paid
    - Output – None
    - Algorithm – Sets the inputted value to the moneyPaid variable

    - Complexity – O(1)
- **public int getTotalScore**() – retrieves page rank by summating frequency, siteAge, linkReference, and moneyPaid.
    - Input – None
    - Output – Rank: summation of four ranking variables
    - Algorithm – returns the sum of frequency, siteAge, linkReference, and moneyPaid.
    - Complexity – O(1)
- **public String getName**() - Retrieves value of private URL variable
    - Input – None
    - Output – URL: String that is of private URL variable
    - Algorithm – returns URL
    - Complexity – O(1)
- **public void printAttributes**() – Prints all attributes of object including page rank
    - Input – None
    - Output – None
    - Algorithm – Prints URL, frequency, siteAge, linkReference, moneyPaid, and page rank.
    - Complexity – O(1)

SearchTerms Class implements Rank

The purpose of the SearchTerms class is to provide an object view of each search term that is queried in the application.

Variables

- **searchTerm** – private string variable that holds search term
- **numTimesSearched** – private integer variable that holds the count of number of times the search term has been queried.
- **Ranking –** equivalent to numTimesSearched variable

Methods

- **public SearchTerms**(String searchTerm) – Constructor of object that takes inputted search term.
    - Input – searchTerm: String that the user had inputted
    - Output – None
    - Algorithm – Sets the searchTerm to object's searchTerm value and sets numTimesSearched to 1.
    - Complexity – O(1)
- **public int getName**() - Retrieves value of private searchTerm variable
    - Input – None
    - Output – searchTerm: String that is contained in searchTerm variable
    - Algorithm – returns the search term

    - Complexity – O(1)
- **public int getTotalScore**() - Retrieves value of private numTimesSearched variable
    - Input – None
    - Output – numTimesSearched: Integer that holds number of times word has been searched.
    - Algorithm – returns the numTimesSearched variable value.
    - Complexity – O(1)
- **public void increase**(**int** value) – Increases value of numTimesSearched by adding value
    - Input – value: integer of number of additional times word was searched
    - Output – None.
    - Algorithm – Adds value to the numTimesSearched variable.
    - Complexity – O(1)
- **public boolean equals**(Object o) – Override function that returns if two objects are the same
    - Input – o: Object that you want to compare another object to
    - Output – Boolean that says if two objects are equal.
    - Algorithm – If inputted object is of type SearchTerms, checks to see if two have the same name (searchTerm). If inputted object is a String, checks if name of object and string are the same. Or else returns false.
    - Complexity – O(3)
- **public void printAttributes**() – Prints the two attributes, searchTerm and numTimesSearched
    - Input – None.
    - Output – None; prints to the screen.
    - Algorithm – Prints "searchTerm : numTimesSearched"
    - Complexity – O(1)
- **public int getIndex**() – Returns total score
    - Input – None
    - Output – total score: integer calculated from getTotalScore()
    - Algorithm – Calls getTotalScore()
    - Complexity – O(1)
- **public void setRanking**(int r) – Sets private ranking variable
    - Input – r: integer to set ranking
    - Output – None
    - Algorithm – sets input r to ranking variable
    - Complexity – O(1)
- **public int getRanking**() - Retrieves value of private ranking variable
    - Input – None
    - Output – ranking: integer that is of private ranking variable
    - Algorithm – returns ranking

- Complexity – O(1)

BuildApp Class

The purpose of the BuildApp class is to connect all elements of HeapSort, SearchTerms, WebPageURL, and FunnyCrawler to create a stable, clean interface for the user to complete any of the available functionality.

Variables

- **URLObjects** – ArrayList<Rank> Stores the complete list of search results from query
- **urlBST** – BST to manipulate search results
- **searchOccurrences** - ArrayList<Rank> Stores all search terms queried in application as SearchTerm objects
- **top10searchTerms** - ArrayList<Rank> Stores the top 10 searches from the heap generated by searchOccurrences
- **bucketSort –** BucketSort object for first search term URLs
- **firstSearchResults** - ArrayList<Rank> Stores first search term queried in application as WebPageURL objects

Methods

- **private** ArrayList<Rank> **search**(String search) – Private method to instantiate web crawler and retrieve complete set of results from query
  - Input – search: String the user has queried.
  - Output – ArrayList<Rank>: List of WebPageURL objects created from results.
  - Algorithm – Creates WebPageURL objects for every search result in the set returned from the web crawler.
  - Complexity – O(n)
- **public void enterSearch**(String searchTerm) – public method to retrieves search string from user, calls search and retrieves results, then builds BST, and stores the results if first time being searched.
  - Input – searchTerm: String the user has queried.
  - Output – None.
  - Algorithm – Add search term to searchOccurrences ArrayList, retrieve list of URLObjects and build a BST. Stores the URLObjects if first time searched.
  - Complexity – O(n lg n)
- **public void quicksortResults**() – Quicksort original search results from URLObjects by Total score
  - Input – None
  - Output – None.
  - Algorithm – Call quicksort on URLObjects.
  - Complexity – O(1)
- **public void printHeader**() – Print header columns
  - Input – None

    - Output – None.
    - Algorithm – Print header columns.
    - Complexity – O(1)
- **public void print**(ArrayList<Rank> arrlist) – For any ArrayList of Rank objects, call its printAttributes method
    - Input – arrlist: ArrayList<Rank> Stores ArrayList of Rank objects
    - Output – None.
    - Algorithm – For every object in the ArrayList, call its printAttributes method.
    - Complexity – O(1)
- **public int getIndexByname**(String searchTerm) – Used to see if search term has been already searched and retrieve its index in searchOccurrences
    - Input – searchTerm: String the user has queried.
    - Output – Index of where searchTerm is located in searchOccurrences.
    - Algorithm – Goes through searchOccurrences and uses SearchTerms class's equals method to determine is searchTerm is in searchOccurrences. If not there, returns -1.
    - Complexity – O(n)
- **private void addSearchTerm**(String searchTerm) – Increments searchTerm if exists or creates new SearchTerms object and add it to searchOccurrences. Called by enterSearch.
    - Input – searchTerm: String the user has queried.
    - Output – None.
    - Algorithm – Retrieves index from getIndexByname and increments value by one or adds new SearchTerms object to searchOccurrences.
    - Complexity – O(1)
- **public void getTop10searches**() – Sorts searchOccurrences and prints top 10 searches.
    - Input – None.
    - Output – None.
    - Algorithm – Builds BST for top searches using searchOccurrences, searches for top 10 in BST and adds them to ArrayList, and prints the top 10 searches in order.
    - Complexity – O(n)
- **public void userInterface**() – Holds all the options the user can do, validates user input, and calls respective methods of user's choice.
    - Input – None.
    - Output – None.
    - Algorithm – Displays options, asks user to pick, calls methods based on option choice and loops back until the user selects Quit.
    - Complexity – O(1)
- **public static void main**(String[] args) – Front point of the application
    - Input – None.
    - Output – None.

- Algorithm – Instantiates BuildApp object and calls userInterface.
- Complexity – O(1)

Problems Encountered During Implementation

One of the first problems I encountered was deciding whether to make the Binary Search Tree class implement generics and how to do so. I realized by implementing generics, the code became repetitive and fundamentally didn't make sense.

The next problem I encountered was trying to figure out how to implement the Binary Search Tree search method as I did not have a rank variable in my WebPageURL object. I ended up having to add a new variable and a few getter and setter methods to the Rank interface so that the search would work properly. I also created a helper function for search called select, where all the textbook pseudocode is primarily.

An issue that arose due to creating this member variable was how to update the rankings of each object whenever a Node was inserted or deleted from the Binary Search Tree. To overcome this issue, I created a method called updateRankings() that used the search function to iterate from zero to the size of the Binary Search Tree, and when it found the Node for that rank, it would update the Node's key which was a Rank object.

The fourth problem encountered was with BucketSort and trying to figure out how to map the values to the correct buckets and get the correct hostname while disregarding "http" or "www". To overcome this issue, after doing some online research, I found that by creating a URI object, you can retrieve the hostname of a website.

The final problem I encountered arose when using the Scanner object's nextInt() method to read user input. After using that method, the next time the options were displayed, the code would automatically say there was an error with the choice selected and would print the options twice. To fix this, I decided to read everything in as nextLine() and use a try-catch block to check user input.

Lessons Learned

While there were many foci of the application from Binary Search Trees, Quicksort, and BucketSort, this project taught me a lot more than just the implementation of those three structures and methods.

Another important finding was the importance of keeping most variables and methods as private to protect your code and help eliminate the possibility of errors. If you are trying to reference a private method in another class, the method won't even show up, which eliminates the possibility of calling the wrong method in your code and spending several hours debugging only to realize you called the wrong method.

Java's Uniform Resource Identifier was another cool object I used for the first time. By creating this object with a URL string, I was able to find the hostname of a site and could parse it to exclude "http" or "www".