Google Search Simulation 2.0: More Methods of Sorting and Manipulation

Uyen Nguyen

CS 146: Data Structures and Algorithms

Professor Mike Wu, Section 6, San Jose State University

About the Design and Implementation

Because the assignment requires the program be able to perform quicksort, BST methods, and bucket sort, these three factors are separated into their own classes. Only quicksort and the insertion sort part of bucket sort have been previously made and altered to fit the requirements while BST and the rest of bucket sort are based off the algorithm provided from the textbook; this is done so it would be easier for implementation and debugging since three different algorithms serving three different purposes will be used.

The assignment is based off the simulator used in Programming Assignment I, where the program allows the user to sort and manipulate data through user input and the scores are based on the total sum of the four factors chosen by random (for the sake of simplicity for a simple simulator). The only difference is a major modification of the simulator from the previous assignment, and the use of three algorithms compared to only one (Heapsort). User input is used to determine which algorithm (quick sort, bucket sort, or inorder of BST) they want to use or how they want to manipulate the data (insertion, deletion, transplant, search, etc using BST) through the use of a scanner and switches; this is done by choosing from the menu and selecting the objects by inputting the number that is associated to the task the user wants to do. All the information is used via a web crawler that collects; the web crawler works using an external jar file called jsoup to collect URLs and store them into a data structure for future use.

The assignment overall is about showing three algorithms that were taught in class prior to the assignment as a way to show how each one either sorts or manipulates data, while also showing how these algorithms can be used in real life applications.

The Classes and Their Methods

- Keyword Class: this class is made of five parameters that will be given for each keyword found by the webcrawler class. The five parameters include the name as a string and the factors - the frequency of the name on the page, time the webpage existed, the amount of associated webpages, and the amount paid by the page to increase advertisement - as ints. Two variables also exist: the index and the score. The score itself is the sum of all int parameters, and can be recalculated when one of those parameters change value. The index is an int used to help find and set the index of the keyword at a faster time and for the sake of the simulator also help with debugging if needed.
    - Variables:
        - private String name: the name of the keyword
        - private int f: the score for frequency of keyword on the page
        - private int t: the score for time the page existed
        - private int o: the score for the amount of other pages that reference the same keyword
        - private int p: the score for the amount of money paid by the page to increase their placement
        - private int score: the sum of all the factors
        - private int index: the index of where the keyword is in the arraylist. Initialized at 0.

- Methods:
  - Constructor: public Keyword(String name, int f, int t, int o, int p) {}

    The constructor of the keyword found by webcrawler. Initializes parameters and sets score to the sum of parameters
  - Get methods (public String getName(), public int getScore(), public int getF() [frequency], public int getT() [time on internet], public int getO() [other website associated with it], public int getP() [payment for exposure], public int getIndex())

    Simple get methods of the class to retrieve values needed.
  - Set methods (public void setName(String theName), public void setF(int aF) [frequency], public void setT(int aT) [time on internet], public void setO(int aO) [other website associated with it], public void setP(int aP) [payment for exposure], public void setIndex(int aIndex))

    Simple set methods of the class to change data.
  - public void calculateScore(int af, int at, int ao, int ap)

    Calculates the score given the factors.

- QuickSort Class: This class serves to sort the list of keywords using the quicksort algorithm. The code is based on the one used in the previous homework assignment and altered to sort the keywords by score.
  - Methods
    - public static ArrayList<Keyword> quickS(ArrayList<Keyword> k, int p, int r)

The method is responsible for recursively sorting the ArrayList, doing so from left to right. It takes the parameters k, p, and r; k represents the arraylist being sorted, p represents the leftmost index of k, and r represents the rightmost index of k. The method calls another method to find the partition that will be used to help sort k.

- public static int partition(ArrayList<Keyword> k, int p, int r)

  This method is called by the quickS method and serves to find the pivot for the quicksort algorithm. The method returns the index of the partition after sorting the arraylist based on the index's keyword score.

- static void printArrayQ(ArrayList<Keyword> k)

  This method is used to simply print the list of items along with their rank, score, and the index they are placed in. The method is similar to the method used in bucket sort when returning the values needed.

- BucketSort Class: This class serves to sort the list of keywords using the bucket sort algorithm. The code is based on the textbook's algorithms and altered to sort the keywords by name.
  - Methods:
    - public static Keyword[] bucketS(Keyword[] k)

      The method is responsible for sorting the array, doing so from left to right. It takes the parameters k; k represents the array being sorted. The method calls another method to sort the individual buckets created to help sort k.

    - public static Keyword[] insertS(Keyword[] k)

This method is called by the bucketS method and serves to sort each bucket by insertion sort. The insertion sort compares strings so that it will be sorted accordingly.

- static void printArrayB(Keyword[] k, int x)

  This method is used to simply print the list of items along with their rank, score, and the index they are placed in. The method is similar to the method used in quick sort when returning the values needed. Some differences include the method being implemented for an array and changes on how to present the list of how many items to return.

- BinarySearchTree Class: This class serves to manipulate the list of keywords using methods of the binary search tree data structure. The code is based on the textbook's algorithm and altered to manipulate the keywords based on their scores. Because BST is easier to implement using nodes, a node class is created to help manipulate data.
  - Node class: The node class is based on the Keyword class and is exclusively used by the BST class. Like the Keyword class, the Node class has similar get and set methods along with another original method that converts the keyword into the node class.
    - Variables:
      - public int key: holds the score of Keyword
      - public String URL: holds the URL string of Keyword
      - public Node left, right, p: pointers of the node (left, right, p)
      - public int index = 0: index of the node, initialized to 0;

- Methods
  - Constructor: public Node(String URL, int score)

    The constructor of the node based on Keyword. Because it is based on score, the parameters of the Keyword Type is not needed.
  - Get methods (public String getURL() , public int getKey(), getIndex())

    simple get methods of the class to retrieve values needed.
  - Set methods: (public void setURL(String u), public void setKey(int k), public void setIndex(int aIndex))

    simple set methods of the class to change data.
  - public static Node keywordToNode(String word, int s)

    This is used to convert the Keyword type into the Node type so it can be used for the BST.

- BST Variables:
  - private Node root: the first node. The root of the BST. Initialized as null to represent an empty BST.
- BST Methods
  - Constructor: public BinarySearchTree():

    Sets the root to null.
  - public static Node Search(Node x, int k):

    Recursively searches for the node of the given score. Starts the search at root.

- public static Node Iterative_Search(Node x, int k):

  Iteratively searches for the node of a the given score. Stars the search at root and does not use recursions.

- public static Node Minimum(Node x):

  Returns the node with the smallest key in the BST. Starts the search at the root.

- public static Node Maximum(Node x):

  Returns the node with the biggest key in the BST. Starts the search at the root.

- public static Node Successor(Node x):

  Finds the successor of the given node (the node that contains the next biggest key after the provided node).

- public static void Transplant(BinarySearchTree B, Node u, Node v):

  Changes the location of the two given nodes in the given tree by swapping the positions of the two nodes.

- public static void Insert(BinarySearchTree B, Node z):

  Inserts the given node into the given tree while maintaining BST properties.

- public static void Delete(BinarySearchTree B, Node z):

  Deletes the given node from the given tree while maintaining BST properties

- public static void inOrder(Node x):

Returns the list of Nodes sorted in order.

- Web crawler classes: The webcrawler class is the same web crawler used in the last assignment with a few altercations (changed the URL to something more specific and does not take keyword input so that it can really scrape the internet for any possible URLs)
    - Spider class: the class is used to find the pages and store them into a needed data structure so that it can be used and manipulated. The class works with the SpiderLeg class to find and collect the needed pages given a URL to scrape.
        - Variables:
            - private static final int MAX_PAGES_TO_SEARCH = 30: puts a max on how many pages to retrieve to avoid infinite loop.
            - private Set<String> pagesVisited = new HashSet<String>(): used to store the pages that have been visited and will be returned when needed
            - private List<String> pagesToVisit = new LinkedList<String>(): Used to keep track of how many pages need to be found.
        - Methods:
            - Get methods (public Set<String> getPages(), public ArrayList<String> getList()):

              Simple get methods of the class to retrieve values needed.
            - public void search(String url):

              Searches for the URLs using the SpiderLeg class

- private String nextUrl()

  Retrieves the next URL to be searched and helps keep track of the URLs.

- SpiderLeg class: The SpiderLeg class does most of the work in searching and scraping the internet for URL links and is used by the Spider class. To use the class, an external jar called jsoup is imported. In order to work, a fake robot is used (provided by the original code) to look through the internet.
  - Variables:
    - private static final String USER_AGENT = "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.112 Safari/535.1": The robot used so that any web browser listed (ex. Mozilla, Chrome, Safari) can be used.
    - private List<String> links = new LinkedList<String>(): the list of links found
    - private Document htmlDocument: the HTML document that will be looked into.
  - Methods:
    - public boolean crawl(String url):

      This is where most of the work is done. Using try and catch, this method asks for HTTP request and then stores the links that
    - public List<String> getLinks(): returns the links that have been found by the crawl method

- Google Simulator Class: The simulator class is where the user can perform the assignment requirements and is supposedly based off the previous assignment's own simulator. The simulator class takes in user input via a scanner and uses switches to take in the scanner input to perform the function needed. Quick sort simply needed to sort the data by score. With BST, another switch is put in to ask the user what sort of data manipulation they would like to do. Depending on their choice, they will be asked to input more data or simply return what they are asking for. Bucket sort simply needs another user input to ask how many values the user wants to see (top ten, top 30, etc). Prior to this, the webcrawler will run and the simulator will assign the URLs as type Keyword, using the URL as a string, and assigning random numbers for each factor so they could be calculated. To ensure that each sort and manipulation does not use the same data, a break is put into the case so that when the program runs again, new values will be used.

  For BST, the cases print the initial order of the tree as a way to debug and also show that the method works. Because of time constraints, most of the cases ask for just the score since it is easier to implement in such a short amount of time. Due to a bug that is unable to be fixed, users need to input a random letter before the case runs. (see problems and challenges).

Screenshot Results

Beginning of simulator (Web crawler working and starting menu):

```
**Visiting** Received web page at https://hangouts.google.com/
Found (3) links

**Visiting** Received web page at https://keep.google.com/
Found (11) links

**Visiting** Received web page at https://www.google.com/save
Found (11) links

**Visiting** Received web page at https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/tre
Found (11) links

**Visiting** Received web page at https://whatbrowser.org
Found (6) links

**Visiting** Received web page at https://www.google.com/chrome
Found (57) links

**Visiting** Received web page at https://www.google.com/about/
Found (49) links

**Visiting** Received web page at https://www.google.com/about/products/
Found (325) links

**Visiting** Received web page at https://www.google.com/about/our-commitments/
Found (75) links

**Visiting** Received web page at https://www.google.com/about/stories/
Found (96) links

**Done** Visited 30 web page(s)
Please enter your choice:
1        Sort By Score
2        Change the Data
3        Sort by Name
```

Input does not match (default switch):

```
**Visiting** Received web page at https://www.google.com/about/stories/
Found (96) links

**Done** Visited 30 web page(s)
Please enter your choice:
1        Sort By Score
2        Change the Data
3        Sort by Name
4
Sorry, that was the wrong input.
```

Quick sort run: (Number 20 index and score is 11 and 21 respectively)

```
Please enter your choice:
1        Sort By Score
2        Change the Data
3        Sort by Name
1
Top URLS Sorted By Score:
1. https://play.google.com/?hl=en (Index: 8 Total Score: 34)
2. https://www.blogger.com/ (Index: 22 Total Score: 33)
3. https://translate.google.com/?hl=en (Index: 9 Total Score: 33)
4. https://hangouts.google.com/ (Index: 7 Total Score: 31)
5. https://trends.google.com/trends/trendingsearches/daily?geo=US (Index: 25 Total Score: 28)
6. https://www.google.com/calendar (Index: 18 Total Score: 27)
7. https://www.google.com/webhp (Index: 15 Total Score: 26)
8. https://whatbrowser.org (Index: 0 Total Score: 26)
9. https://www.google.com/about/stories/ (Index: 1 Total Score: 25)
10. https://maps.google.com/maps?hl=en (Index: 28 Total Score: 25)
11. https://docs.google.com/document/?usp=docs_alc (Index: 12 Total Score: 25)
12. https://www.google.com/save (Index: 16 Total Score: 24)
13. https://keep.google.com/ (Index: 26 Total Score: 24)
14. https://www.google.com/chrome (Index: 19 Total Score: 24)
15. https://www.google.com/about/ (Index: 21 Total Score: 24)
16. https://news.google.com/nwshp?hl=en (Index: 13 Total Score: 23)
17. https://www.google.com/finance (Index: 5 Total Score: 23)
18. https://www.youtube.com/?gl=US (Index: 14 Total Score: 23)
19. https://books.google.com/bkshp?hl=en (Index: 24 Total Score: 22)
20. https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/trendingsearches/daily?geo%3DUS&foll
21. https://plus.google.com/?gpsrc=ogpy0 (Index: 27 Total Score: 20)
22. https://www.google.com/about/our-commitments/ (Index: 23 Total Score: 18)
23. https://myaccount.google.com/?utm_source=OGB&utm_medium=app (Index: 17 Total Score: 18)
24. https://mail.google.com/mail/ (Index: 6 Total Score: 17)
25. http://www.google.com/shopping?hl=en (Index: 4 Total Score: 17)
26. https://drive.google.com/ (Index: 20 Total Score: 16)
27. https://www.google.com/intl/en/options/ (Index: 10 Total Score: 16)
28. https://photos.google.com/?pageId=none (Index: 2 Total Score: 16)
29. https://contacts.google.com/?hl=en (Index: 29 Total Score: 11)
30. https://www.google.com/about/products/ (Index: 3 Total Score: 10)
```

BST:

BST menu

```
**Done** Visited 30 web page(s)
Please enter your choice:
1        Sort By Score
2        Change the Data
3        Sort by Name
2
Please Enter Your Choice:
1        Add a URL
2        Remove a URL
3        Sort Inorder
5        Find a URL
6        Return the URL with the smallest score
7        Return the URL with the biggest score
8        Return the successor of a URL
```

Insert a URL (fake.com)

```
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/trendingsearches/daily?geo%3DUS&followup=https://trends.googl
https://books.google.com/bkshp?hl=en (30)
What would you like to input Please enter its score
17
New Order of URLs:
https://www.youtube.com/?gl=US (12)
https://www.google.com/about/our-commitments/ (14)
https://www.google.com/about/stories/ (15)
https://news.google.com/nwshp?hl=en (16)
fake.com (17)
https://www.google.com/finance (18)
https://myaccount.google.com/?utm_source=OGB&utm_medium=app (18)
https://translate.google.com/?hl=en (19)
https://www.google.com/intl/en/options/ (19)
https://www.google.com/calendar (19)
https://www.google.com/about/products/ (20)
https://www.google.com/save (20)
https://drive.google.com/ (21)
http://www.google.com/shopping?hl=en (23)
https://play.google.com/?hl=en (23)
https://www.google.com/about/ (23)
https://www.google.com/chrome (24)
https://trends.google.com/trends/trendingsearches/daily?geo=US (24)
https://maps.google.com/maps?hl=en (24)
https://whatbrowser.org (25)
https://photos.google.com/?pageId=none (25)
https://docs.google.com/document/?usp=docs_alc (25)
https://keep.google.com/ (25)
https://hangouts.google.com/ (26)
https://www.google.com/webhp (26)
https://www.blogger.com/ (26)
https://mail.google.com/mail/ (28)
https://plus.google.com/?gpsrc=ogpy0 (28)
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/trendingsearches/daily?geo%3DUS&followup=https://trends.googl
https://books.google.com/bkshp?hl=en (30)
```

Remove a URL (fake.com)

```
https://trends.google.com/trends/trendingsearches/daily?geo=US (19)
fake.com (20)
https://www.google.com/about/stories/ (21)
https://play.google.com/?hl=en (21)
https://www.google.com/finance (22)
https://docs.google.com/document/?usp=docs_alc (24)
https://www.google.com/save (24)
https://myaccount.google.com/?utm_source=OGB&utm_medium=app (24)
https://www.google.com/chrome (24)
https://photos.google.com/?pageId=none (26)
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.goo
https://news.google.com/nwshp?hl=en (26)
https://www.youtube.com/?gl=US (26)
https://www.blogger.com/ (26)
https://www.google.com/calendar (28)
https://keep.google.com/ (28)
https://www.google.com/about/products/ (29)
https://www.google.com/about/our-commitments/ (32)
https://plus.google.com/?gpsrc=ogpy0 (32)
What would you like to remove. Please enter its url
fake.com
New Order of URLs:
https://mail.google.com/mail/ (7)
https://www.google.com/about/ (12)
https://www.google.com/intl/en/options/ (13)
https://translate.google.com/?hl=en (17)
https://drive.google.com/ (17)
https://whatbrowser.org (18)
http://www.google.com/shopping?hl=en (18)
https://hangouts.google.com/ (18)
https://www.google.com/webhp (18)
https://maps.google.com/maps?hl=en (18)
https://books.google.com/bkshp?hl=en (19)
https://trends.google.com/trends/trendingsearches/daily?geo=US (19)
https://www.google.com/about/stories/ (21)
https://play.google.com/?hl=en (21)
https://www.google.com/finance (22)
```

In order sort:

```
Order of URLs:
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/trendingsearches/daily?geo%3DUS&followup=https
https://www.youtube.com/?gl=US (4)
https://photos.google.com/?pageId=none (8)
https://www.google.com/about/products/ (8)
https://www.google.com/finance (8)
https://plus.google.com/?gpsrc=ogpy0 (8)
https://maps.google.com/maps?hl=en (8)
https://hangouts.google.com/ (12)
https://news.google.com/nwshp?hl=en (12)
https://whatbrowser.org (16)
https://mail.google.com/mail/ (16)
https://www.google.com/calendar (16)
https://keep.google.com/ (16)
https://translate.google.com/?hl=en (20)
https://myaccount.google.com/?utm_source=OGB&utm_medium=app (20)
https://drive.google.com/ (20)
https://play.google.com/?hl=en (24)
https://www.google.com/chrome (24)
https://www.google.com/save (28)
https://www.google.com/about/ (28)
https://www.blogger.com/ (28)
https://books.google.com/bkshp?hl=en (28)
https://www.google.com/intl/en/options/ (32)
https://docs.google.com/document/?usp=docs_alc (32)
https://www.google.com/about/our-commitments/ (32)
https://www.google.com/about/stories/ (36)
http://www.google.com/shopping?hl=en (40)
https://www.google.com/webhp (40)
https://trends.google.com/trends/trendingsearches/daily?geo=US (40)
```

Find URL:

```
List of URLs:
https://www.google.com/intl/en/options/ (10)
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/trendingsearches/daily?geo%3DUS&followup
https://maps.google.com/maps?hl=en (14)
https://www.google.com/about/stories/ (15)
https://www.google.com/about/products/ (15)
https://mail.google.com/mail/ (15)
https://books.google.com/bkshp?hl=en (15)
https://docs.google.com/document/?usp=docs_alc (16)
https://www.google.com/calendar (16)
https://translate.google.com/?hl=en (17)
https://myaccount.google.com/?utm_source=OGB&utm_medium=app (17)
https://news.google.com/nwshp?hl=en (18)
https://www.google.com/webhp (18)
https://www.google.com/finance (19)
https://www.google.com/about/ (20)
https://whatbrowser.org (21)
https://www.google.com/chrome (21)
https://plus.google.com/?gpsrc=ogpy0 (21)
https://www.google.com/about/our-commitments/ (22)
https://trends.google.com/trends/trendingsearches/daily?geo=US (22)
https://photos.google.com/?pageId=none (23)
http://www.google.com/shopping?hl=en (24)
https://www.youtube.com/?gl=US (25)
https://www.google.com/save (25)
https://www.blogger.com/ (25)
https://play.google.com/?hl=en (26)
https://hangouts.google.com/ (27)
https://drive.google.com/ (29)
https://keep.google.com/ (30)
What score would you like to find or try to find?
20
Our results found this:
Regular search: https://www.google.com/about/ at index 21
Iterative search: https://www.google.com/about/ at index 21
```

Minimum:

```
URLs:
https://www.google.com/save (6)
https://www.blogger.com/ (8)
https://books.google.com/bkshp?hl=en (11)
https://maps.google.com/maps?hl=en (13)
https://www.google.com/webhp (14)
https://mail.google.com/mail/ (15)
https://photos.google.com/?pageId=none (16)
https://docs.google.com/document/?usp=docs_alc (18)
https://www.google.com/finance (20)
https://keep.google.com/ (20)
https://whatbrowser.org (21)
https://trends.google.com/trends/trendingsearches/daily?geo=US (21)
https://myaccount.google.com/?utm_source=OGB&utm_medium=app (22)
https://www.google.com/intl/en/options/ (24)
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.goog
https://www.google.com/about/ (25)
https://www.google.com/about/stories/ (26)
https://www.google.com/about/products/ (26)
http://www.google.com/shopping?hl=en (26)
https://hangouts.google.com/ (26)
https://play.google.com/?hl=en (26)
https://translate.google.com/?hl=en (26)
https://www.youtube.com/?gl=US (26)
https://news.google.com/nwshp?hl=en (27)
https://www.google.com/chrome (27)
https://drive.google.com/ (27)
https://www.google.com/about/our-commitments/ (27)
https://www.google.com/calendar (32)
https://plus.google.com/?gpsrc=ogpy0 (36)
The smallest score is https://www.google.com/save with a score of 6 at index 16
```

Maximum:

```
URLs:
https://myaccount.google.com/?utm_source=OGB&utm_medium=app (6)
https://play.google.com/?hl=en (9)
https://www.google.com/chrome (9)
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/trendingsearches/dail
https://drive.google.com/ (15)
https://maps.google.com/maps?hl=en (15)
http://www.google.com/shopping?hl=en (16)
https://docs.google.com/document/?usp=docs_alc (16)
https://www.google.com/finance (18)
https://www.google.com/about/ (18)
https://whatbrowser.org (19)
https://www.google.com/about/stories/ (19)
https://news.google.com/nwshp?hl=en (19)
https://www.youtube.com/?gl=US (19)
https://www.blogger.com/ (19)
https://translate.google.com/?hl=en (22)
https://www.google.com/intl/en/options/ (22)
https://www.google.com/webhp (23)
https://keep.google.com/ (23)
https://trends.google.com/trends/trendingsearches/daily?geo=US (24)
https://www.google.com/about/products/ (25)
https://www.google.com/save (25)
https://www.google.com/about/our-commitments/ (25)
https://books.google.com/bkshp?hl=en (25)
https://mail.google.com/mail/ (26)
https://hangouts.google.com/ (26)
https://www.google.com/calendar (27)
https://plus.google.com/?gpsrc=ogpy0 (27)
https://photos.google.com/?pageId=none (31)
The largest score is https://photos.google.com/?pageId=none with a score of 31 at index 2
```

Successor (first finds node with score):

```
Order of URLs:
https://books.google.com/bkshp?hl=en (13)
https://hangouts.google.com/ (15)
https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/trends/trendingsearches/daily?geo%3DUS&t
https://keep.google.com/ (15)
https://www.google.com/webhp (16)
https://www.google.com/save (18)
https://www.blogger.com/ (18)
https://www.google.com/intl/en/options/ (19)
http://www.google.com/shopping?hl=en (20)
https://play.google.com/?hl=en (21)
https://www.google.com/about/ (21)
https://photos.google.com/?pageId=none (23)
https://news.google.com/nwshp?hl=en (23)
https://www.youtube.com/?gl=US (23)
https://plus.google.com/?gpsrc=ogpy0 (23)
https://www.google.com/about/products/ (24)
https://myaccount.google.com/?utm_source=OGB&utm_medium=app (24)
https://www.google.com/about/our-commitments/ (24)
https://whatbrowser.org (25)
https://mail.google.com/mail/ (25)
https://trends.google.com/trends/trendingsearches/daily?geo=US (25)
https://translate.google.com/?hl=en (26)
https://maps.google.com/maps?hl=en (26)
https://drive.google.com/ (27)
https://docs.google.com/document/?usp=docs_alc (29)
https://www.google.com/calendar (30)
https://www.google.com/finance (31)
https://www.google.com/about/stories/ (35)
https://www.google.com/chrome (38)
What would you like to find the sucessor of? Please enter its score
35
The successor is https://www.google.com/chrome at index19 with a score of 38
```

Invalid input:

```
**Done** Visited 30 web page(s)
Please enter your choice:
1        Sort By Score
2        Change the Data
3        Sort by Name
2
Please Enter Your Choice:
1        Add a URL
2        Remove a URL
3        Sort Inorder
4        Find a URL
5        Return the URL with the smallest score
6        Return the URL with the biggest score
7        Return the successor of a URL
9
a
Sorry, that was the wrong input.
```

Bucket Sort (Top 10):

```
How many would you like to see? (Max is 30)
10
Top 10 URLS Sorted By Score:
1. http://www.google.com/shopping?hl=en (Index: 0 Total Score: 29)
2. https://accounts.google.com/ServiceLogin?passive=1209600&continue=https://trends.google.com/tren
3. https://books.google.com/bkshp?hl=en (Index: 2 Total Score: 22)
4. https://contacts.google.com/?hl=en (Index: 3 Total Score: 9)
5. https://docs.google.com/document/?usp=docs_alc (Index: 4 Total Score: 10)
6. https://drive.google.com/ (Index: 5 Total Score: 31)
7. https://hangouts.google.com/ (Index: 6 Total Score: 33)
8. https://keep.google.com/ (Index: 7 Total Score: 16)
9. https://mail.google.com/mail/ (Index: 8 Total Score: 22)
10. https://maps.google.com/maps?hl=en (Index: 9 Total Score: 19)
```

How to Run Application

1. After unzipping the file by extracting them to your desired destination, run them on the command prompt using the line: java -jar C:\Users\(Name on PC)\(Location of Folder)\Uyen_Nguyen-PA2\Assignment-2.jar.
    a. This is based off the command prompt line I used for my computer. For reference, I used:

```
C:\Users\Carolyn>java -jar C:\Users\Carolyn\Documents\Uyen_Nguyen-PA2\Assignment-2.jar
```

    I put the file inside my documents file.
    b. Just in case the project needs to be run on an IDE, copy and paste the provided .java files and import the provided jsoup .jar file.
2. The simulator will start by searching for the links before returning three options and prompting you to choose based on the number.
    a. Option 1: Will return the 30 URLs sorted by score. The order of each line printed is Rank #. URL (Index: Total Score:)
    b. Option 2: Will provide a menu list of how you want to manipulate the data. All options will initially print out the order of the Binary Search Tree inorder to give a reference for the user since this is merely a simulator. The order of each line printed is Rank #. URL (Index: _ Total Score: _)
        i. Option 1: Inserting. The insert performance requires the user to simply insert a score; to make it easier for the user to see their URL implemented, it only needs that score. The program will then return the Binary Search

Tree inorder with the included URL labeled as “fake.com” with the score provided.

ii. Option 2: Removing. The URL labeled “fake.com” is included into the BST and will serve to show how the remove option is used. If revisited, the program will be altered so that any URL can be removed (it is also tedious to manually write down every url)

iii. Option 3: Sort in order. The program will simply return the BST sorted in order.

iv. Option 4: Find a URL. The program will show the BST inorder and will ask you for the score you are looking for. The program will return the regular search value and the iterative search value. They should return the same.

v. Option 5: Find minimum. The program will show the BST inorder as reference and should return the first value of the order.

vi. Option 6: Find maximum. The program will show the BST inorder as reference and should return the last value of the order.

vii. Option 7: Find successor. The program will show the BST inorder as reference. The user will be asked to input the score of the URL and the program will return the successor of that URL.

c. Will first ask how many values you want to see, with the max amount being 30. It will then bucket sort the items by name and return only as much as you want to see.

3. The simulator will only run once. To test other features, the program needs to run again (I was unable to implement the code needed so it will continue before the deadline).
4. If an invalid value for the menu is inputted, the program will simply say the value is invalid and stop the program.

Problems and Challenges Encountered

1. Bucket sort implementations:

   Honestly, the bucket sort pseudocode was difficult to translate into code that can be used for the assignment. It was the translation that confused me, which was the part of the program besides creating the simulator that took the longest. This could be from my lack of knowledge on some implementations and syntax that could help me on this, and the fact that the algorithm is used to order by string compared to typically by number makes it sort of intimidating to approach.

   I ended up slightly deterring from the algorithm because of this, but it still ran as it should; possibly when I return to improve the program I can make it actually follow the algorithm.

2. Webcrawler modification:

   Initially, the webcrawler would return a list of URLs that contain a given keyword. Because the assignment requires a list of keywords, initially I tried to find a better web crawler that could do that. After not being able to find one that suited my needs, I turned back to modifying the original one I had. I thought initially that I can simply take the substring of the URL after .com/ and that would be my keyword; however, while running the code prior to implementing this I noticed the URLs are not all the same. I decided to treat the URLs as the keywords themselves, and should there be more time I could figure out how to get the keywords instead.

3. Creating a better simulator:

   In my case, it was creating one from the ground up.

   In the last programming assignment, the simulator created was technically not a simulator at all. It wasn't able to show the programming requirements needed for the assignment. Since this assignment requires altering the first and I technically didn't have a first, that means the simulator for this assignment has to be recreated from scratch. It's not 100 percent from complete scratch, as the part where URLs are assigned as a type Keyword and put into an ArrayList to be used was from the previous assignment. It took some drawing and planning as to how the simulator is to be presented but for simplicity I went with a simple design of a plain menu and numbered options.

4. BST in simulator:

   For some reason, the simulator when using BST will not work after inputting the chosen number unless a random letter is entered afterwards. This problem didn't rise until implementing the insertion option (prior to this, most of the options do not require additional input from the user such as finding the maximum and giving the list of URLs in order) and those that did (successor) did not run into this problem. When insertion was put in, the problem arose, but then it happened to all the other methods, even the ones that do not require additional user input. It doesn't seem too much of a problem for now, but it is something that I wasn't able to fix or find the root of the problem.

5. The use of nodes in addition to the Keyword class:

   Because BST required the use of nodes, I initially created a node class and made it generic so that it can be used by the already made Keyword class. However, it didn't

work as planned, so I created a node class where the nodes can be made based off the already Keyword type items and then can be manipulated. I realized it there is some more efficient way of doing this; but because I already implemented the Keyword class in the two algorithms, it seems more reasonable to make a class that can be altered from Keyword type to Node.

Lessons Learned

Despite the challenges that did come with the assignment, overall the program seems to do as expected and, compared to the previous assignment, performs better. With the program Quick sort, Bucket sort, and using BST to change data can be seen and used on a real life application. Personally, it had challenges that may not be similar to that of others, since the program requires me to create a new simulator compared to those who simply have to alter theirs and to generally take what I know and figure out how to apply them to the given situation. I honestly still think that it can still be improved on and there are still some bits that need to be fixed, but overall the program seems to do what it should do.

There were many lessons learned from the assignment but one really stuck out for me: the thing I struggle with the most when it comes to programming is what to do when there is a problem with my code (exceptions, bugs, implementation problems). I get frustrated and feel like I should give up then and there, which makes me less motivated to code, leading me to not enjoy myself, not doing a good job, and ultimately leave me struggling to finish the assignment and do it to the best of my ability. For this assignment, I tried something else with this assignment where I decided to actually calm myself down and take my time to figure out what is wrong with my code; if I'm still frustrated, I work on something else (in this case, I worked on finishing parts of the report needed so that I can try and finish the assignment on time without worrying too much about figuring out what to finish when). I ended up figuring out the problems and how to fix them, and gradually as I started finishing up it became a lot faster to spot the causes and fix them. I think it's because of this lesson I felt like I actually was enjoying myself and putting in the care and effort for this assignment.

The program and lessons that are learned personally meant a lot being that it showed me I can improve on myself and I should believe in myself more when given a challenge or assignment that comes my way. Overall, I'm really happy with how much progress I made and hope I can continue to apply the lessons I learned in the future.