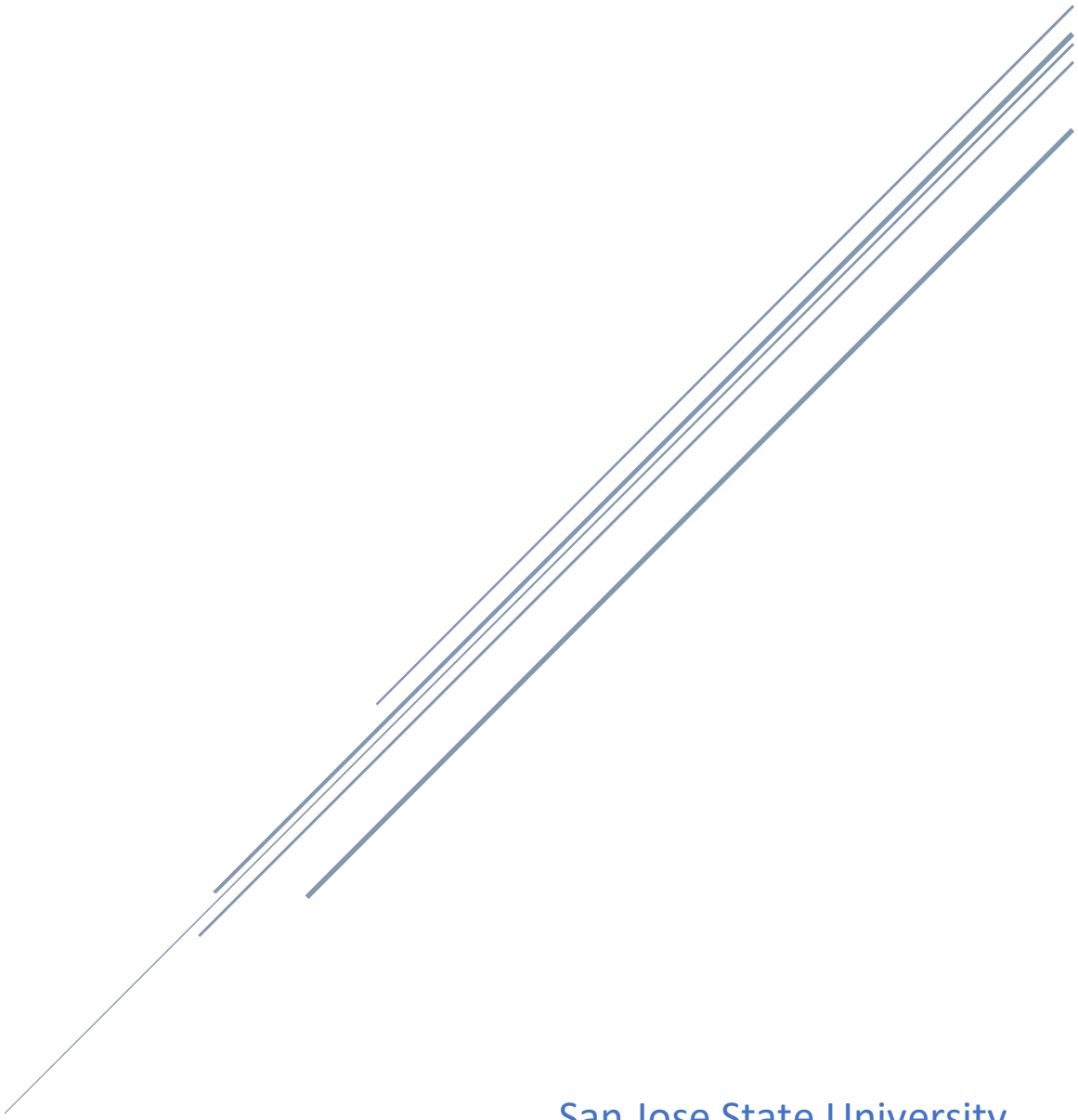
GOOGLE SEARCH ENGINE SIMULATOR USING HEAP SORT

Programming Assignment 1

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 four main sub functions. Once the search is submitted, a list of the top 10 results are displayed to the user along with options on what to do next. These options include:

1. View the complete search list of 30 URLs
2. Pay to raise the rankings of a site (only available for top 10 URLs)
3. View the top 10 searches of the day
4. Run another search

The Simulator consists of five java classes and one interface: HeapSort, FunnyCrawler, BuildApp, WebPageURL, SearchTerms, and Rank Interface.

The BuildApp class uses all classes (HeapSort, FunnyCrawler, WebPageURL, and SearchTerms).
HeapSort uses Rank Interface for its ArrayList.

Test Cases

1. Display the top 10 results for the user query

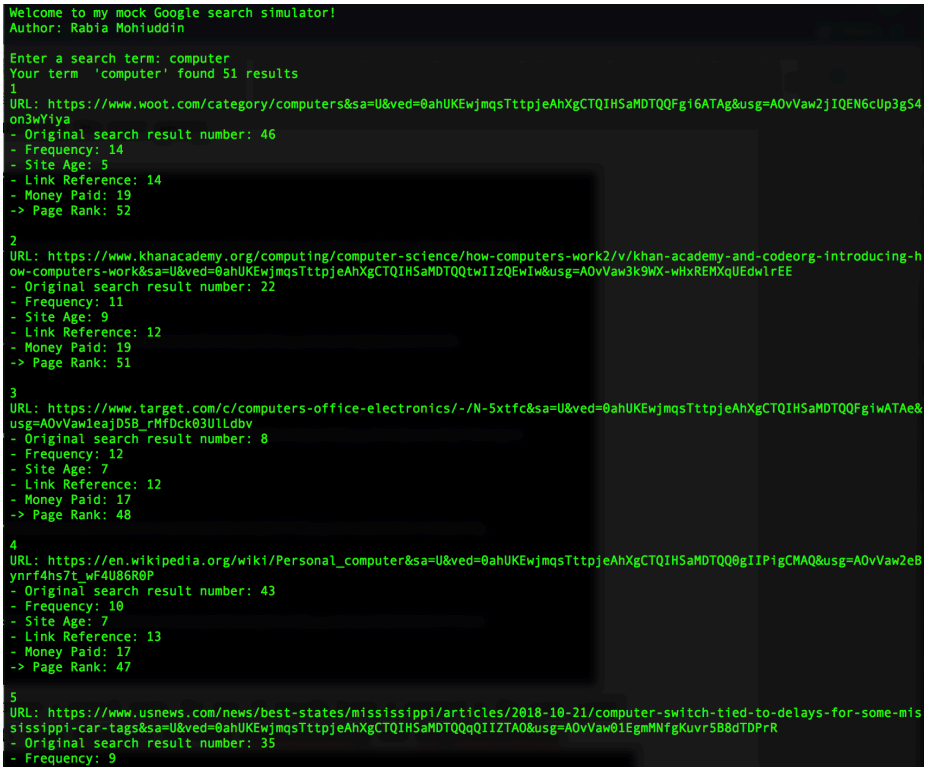


Figure 1: When the user enters a search term, the top 10 results heap is printed to the user. (user scroll for 10). Functionality from HeapSort includes buildMaxHeap and heapExtractMax

```
33      /**
34       * public search method that retrieves search string from user, calls search and retrieves results, then builds heap, extracts top 10 res
35       * top 10 heap, and prints top 10 results
36       *
37       * @param searchTerm
38       *          String -> search term
39       * @return none
40       */
41      public void enterSearch(String searchTerm) {
42          addSearchTerm(searchTerm);          // Given search term user entered, call method to add term to the search term heap
43
44          URLObjects.clear();             // Clear the public URLObjects ArrayList to ensure no old results left behind
45
46          URLObjects = search(searchTerm);        // Get new search results and store them in public ArrayList
47
48          HeapSort hSort = new HeapSort();        // Create HeapSort object to sort all search results
49          hSort.buildMaxHeap(URLObjects);         // Build max heap using the ArrayList of URLObjects retrieved from search results
50          System.out.println("Your term '" + searchTerm + "' found " + URLObjects.size() + " results");       // Print # of results to user
51
52          top10urls.clear();          // Clear the public top10urls ArrayList to ensure no old results left behind
53          for (int i = 0; i < 10 && i < URLObjects.size(); i++) {          // Extract top 10 highest ranked URLs
54                  top10urls.add(hSort.heapExtractMax(URLObjects));            // Add each max node from results to top10urls ArrayList
55          }
56
57          top10heap.buildMaxHeap(top10urls);          // Build max heap using top10urls ArrayList
58          print(top10urls);           // Print top 10 urls and their attributes to user
59
60      }
61
```

Figure 2: The above code produces the output in Figure 1. BuildMaxHeap and heapExtractMax are used.

2. Increase key value

```
10
URL: https://timesofsandiego.com/politics/2018/10/20/dmw-voter-registration-program-botched-by-makeshift-computer-system
/&sa=U&ved=0ahUKEwjmqsTttpjeAhXgCTQIHSaMDTQQqQIIZzAP&usg=AOvVaw3z5uStgpuv5cwxA1rVUaej
- Original search result number: 36
- Frequency: 2
- Site Age: 8
- Link Reference: 12
- Money Paid: 17
-> Page Rank: 39

What would you like to do next?
1. View top 10 URLs from search
2. View the complete search list of 30 URLs
3. Pay to raise the rankings of a site (only available for top 10 URLs)
4. View the top 10 searches of the day
5. Run another search
0. Quit

Option: 3
Which number URL do you want to pay for? 10
Enter dollars you will pay: $ 40
What would you like to do next?
1. View top 10 URLs from search
2. View the complete search list of 30 URLs
3. Pay to raise the rankings of a site (only available for top 10 URLs)
4. View the top 10 searches of the day
5. Run another search
0. Quit

Option: 1
1
URL: https://timesofsandiego.com/politics/2018/10/20/dmw-voter-registration-program-botched-by-makeshift-computer-system
/&sa=U&ved=0ahUKEwjmqsTttpjeAhXgCTQIHSaMDTQQqQIIZzAP&usg=AOvVaw3z5uStgpuv5cwxA1rVUaej
- Original search result number: 36
- Frequency: 2
- Site Age: 8
- Link Reference: 12
- Money Paid: 40
-> Page Rank: 62
```

Figure 3: The search result's last element, result 10 was the last element with Page Rank 32. After paying $100, it is not at the top of the top 10 search results with a Page Rank of 124.

```
144     /**
145      * Increases key value of given index and reorders tree
146      *
147      * @param A
148      *          ArrayList<Rank> -> ArrayList of Rank objects that hold nodes
149      *          index int -> index of node want to change key of key int -> new key value of node
150      * @return void -> newly sorted max heap
151      */
152     public void heapIncreaseKey(ArrayList<Rank> A, int index, int key) {
153         try {       // Try used for catching exception thrown in if statement
154             if (key < A.get(index).getRank()) {     // Check if value of new key is less than current key value
155                 throw new IOException("New key is smaller than current key");       // If less than current value, throw exception
156             }
157         } catch (IOException ex) {      // Catch the exception thrown
158             System.out.println(ex.getMessage());        // Print exception error message
159         }
160
161         if (A.get(index) instanceof WebPageURL) {
162             A.get(index).increase(key);
163         } else {
164             A.get(index).increase(1);
165         }
166
167         while (index > 0 && A.get(parent(index)).getRank() < A.get(index).getRank()) {      // Heapify elements to check if parent's index is les
168             // than child's index
169             Collections.swap(A, index, parent(index));          // Swap values of parent and child
170             index = parent(index);      // Set new index to be parent's index
171         }
172     }
```

Figure 4: The code in HeapSort shows that given an ArrayList of WebPageURL objects, you can increase the value of the amount paid. Below is the implementation of the increase function

```java
    /**
     * Sets new value of moneyPaid variable
     *
     * @param moneyPaid
     *          int -> new value of moneyPaid variable
     * @return none
     */
    public void increase(int value) {
        this.moneyPaid = value;       // Set moneyPaid member variable to input parameter given
    }
```

Figure 5: Implementation of increase function in WebPageURL class

3. View top 10 searches of the day

```
What would you like to do next?
1. View top 10 URLs from search
2. View the complete search list of 30 URLs
3. Pay to raise the rankings of a site (only available for top 10 URLs)
4. View the top 10 searches of the day
5. Run another search
0. Quit

Option: 4
Top 10 search terms today:
1
computer : 2 times

2
jack : 1 times
```

Figure 6: The word computer was searched twice, and jack once, therefore displaying computer first as it was searches more times than jack.

```java
    /**
     * 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

        HeapSort heapSort = new HeapSort();          // Create heapsort object to sort searchOccurrences
        heapSort.buildMaxHeap(searchOccurrences);      // Build max heap using searchOccurrences

        top10searches.clear();
        ArrayList<Rank> searchOccCOPY = new ArrayList<Rank>();
        for (Rank obj: searchOccurrences) {
            searchOccCOPY.add(obj);
        }

        for (int i = 0; i < 10 && i < searchOccurrences.size(); i++) {       // Iterate 10 times or size of searchOccurrences to extract top 10
            top10searches.add(heapSort.heapExtractMax(searchOccCOPY));       // Extract max nodes from searchOccurrences
                                                                             // And add to top10searches ArrayList
        }

        top10searchesHeap.heapSort(top10searches);        // Use public top10searchesHeap to sort top10searches
        Collections.reverse(top10searches);        // Heapsort sorts in ascending order so reverse to sort in descending order
        print(top10searches);            // Print top 10 searches and their attributes to the user

    }
```

Figure 7: The above code in BuildApp shows how the top 10 searches of the day are displayed using heapSort.

Classes and Methods

HeapSort Class

The purpose of the HeapSort class is to provide functionality for sorting an ArrayList of objects of type Rank. HeapSort implements all heap required methods but instead of being dependent on integers, nodes are filled with Rank objects.

Variables

- **heapSize** – private integer that keeps track of current heap size. Especially used for maxHeapify, buildMaxHeap, and heapExtractMax functions

Methods

- **public int parent**(**int** index) – Finds the parent index of passed in index.
  - Input – index as integer who's parent want to find
  - Output – Index of parent
  - Algorithm – Heap properties state that the parent of an index is located at the floor of index divided by 2.
  - Complexity – O(1)
- **public int left**(**int** index) – Finds the index of the passed in index's left child
  - Input – index as integer who's left child want to find
  - Output – Index of left child
  - Algorithm – Heap properties state that the left child of an index is located at index times 2.
  - Complexity – O(1)
- **public int right**(**int** index) - Finds the index of the passed in index's right child
  - Input – index as integer who's right child want to find
  - Output – Index of right child
  - Algorithm – Heap properties state that the left child of an index is located at index times 2, plus 1.
  - Complexity – O(1)
- **public void maxHeapify**(ArrayList<Rank> A, **int** currentIndex) - Reorders heap to maintain max-heap property by comparing parent, left, and right nodes
  - Input – ArrayList<Rank>: ArrayList of Rank objects that will be put into heap, currentIndex: index who's heap you want to heapify.
  - Output – none, ArrayList<Rank> will be sorted as a max heap
  - Algorithm – The left and right child of the current index will be checked to see if they are within the heap size and if they are greater than the current index. If either of them are larger than the current index, the greater value's index will be swapped with the current index, which is its parent node. The method will recursively call maxHeapify on the largest index until the max heap property is obtained.
  - Complexity – O(lg n)

- **public void buildMaxHeap**(ArrayList<Rank> A) – Build a max heap using the given ArrayList of Rank objects and ensure all parent nodes satisfy the max heap property.
    - Input - ArrayList<Rank>: ArrayList of Rank objects that will be put into heap
    - Output - none, ArrayList<Rank> will be sorted as a max heap
    - Algorithm – Set the heap size to the current size of the ArrayList and calls maxHeapify on all nodes without children.
    - Complexity – O(n lg n)
- **public void heapSort**(ArrayList<Rank> A) – Sorts ArrayList in ascending order
    - Input - ArrayList<Rank>: ArrayList of Rank objects that will be put into heap
    - Output - none, ArrayList<Rank> will be sorted as a max heap
    - Algorithm – First build a max heap using the ArrayList, then swap the first and last elements, decrease the heap size, call maxHeapify on the first element to put every node in its proper place and because the largest node is put at the end, it will no longer be considered part of the heap when maxHeapify is called.
    - Complexity – O(n lg n)
- **public** Rank **heapMaximum**(ArrayList<Rank> A) – Peek function that returns the largest element in the heap
    - Input - ArrayList<Rank>: ArrayList of Rank objects that will be put into heap
    - Output – Rank object that contains data of largest element in heap
    - Algorithm – Returns element at index 0 of max heap. No element is deleted
    - Complexity – O(1)
- **public** Rank **heapExtractMax**(ArrayList<Rank> A) – Extracts largest node in max heap
    - Input – ArrayList <Rank>: ArrayList of Rank objects that will be put into heap
    - Output - Rank object that contains data of largest element in heap
    - Algorithm – Returns element at index 0 of max heap and removes it from the heap.
    - Complexity – O(1)
- **public void heapIncreaseKey**(ArrayList<Rank> A, **int** index, **int** key) – Increases key value of given index and reorders tree
    - Input – ArrayList <Rank>: ArrayList of Rank objects that will be put into heap, index: which element you want to increase, key: new value of the index
    - Output - none
    - Algorithm – Checks if key entered is less than current value of node, if so gives error. Otherwise, updates object with new value and checks if still satisfies max heap property by comparing its new value with its parent.
    - Complexity – O(lg n)
- **public void maxHeapInsert**(ArrayList<Rank> A, **int** key) – Insert new element in max heap
    - Input – ArrayList <Rank>: ArrayList of Rank objects that will be put into heap, key: value of the object you want to add

- Output - none
- Algorithm – Increases the heap size, then creates space for the new node and sets its value by calling heapIncreaseKey
- Complexity – O(lg 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 getRank**() – Retrieves total rank for WebPageURL or occurrence for SearchTerms
  - Input – None
  - Output – Rank: either summation of four ranking variables or occurrence.
- **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.

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.

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 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 getRank**() – 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 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.

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 getRank**() - 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)

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
- **top10heap** – HeapSort object to sort the top 10 URLs
- **top10urls** – ArrayList<Rank> Stores the top 10 URLs from the heap generated from URLObjects.
- **searchOccurrences** - ArrayList<Rank> Stores all search terms queried in application as SearchTerm objects
- **top10searchesHeap** – HeapSort object to sort the top 10 searches
- **top10searches** - ArrayList<Rank> Stores the top 10 searches from the heap generated by searchOccurrences

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 add search term to list of terms searched in application and create top 10 results heap and print.
  - Input – searchTerm: String the user has queried.
  - Output – None.
  - Algorithm – Add search term to searchOccurrences ArrayList, retrieve list of URLObjects and build a heap, then extract the top 10 elements and build a top 10 heap and print to the user.
  - Complexity – O(n lg n)
- **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 Max heap using searchOccurrences, then creates a copy of searchOccurrences ArrayList, extracts the top 10 searches from the copy, creates a max heap, and prints the top 10 searches in order.
    - Complexity – O(n lg 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 when I implemented my WebPageURL class with HeapSort is that I decided to change HeapSort's input ArrayList to object type WebPageURL instead of Integer. This was fairly simple as wherever I needed to compare two nodes, I would add a call to the function getRank() so it would compare integer values. What I did not realize was that I would be using HeapSort for the top 10 searches as well because I had initially implemented this as a sorted map (or dictionary). When I changed the top 10 implementation to create nodes of SearchTerms, I had to figure out a way to make HeapSort take a generic type so that it could be used for WebPageURL and SearchTerms. To do this, I implemented an interface called Rank that had some basic methods that could be incorporated into my WebPageURL and SearchTerms classes and changed my HeapSort methods to take an ArrayList of Rank objects. This caused small issues with the naming conventions in the two classes, but with some renaming so the method name would be a little more generic, the issue was resolved.

The second problem I encountered was when trying to create a max heap for the top 10 searches, I was extracting from my searchOccurrences ArrayList which was messing up the my top 10 results. If I tried to view the top 10 results after searching something and then searched the same term after, it would say that the term was only searched once. To fix this, I ended up creating a copy of searchOccurrences when the user wanted to view the top 10 searches and creating a max heap from the copy, so that the values would not be extracted and mess up the top 10 searches.

The third problem I encountered was when implementing my web crawler with the code, the web crawler I found had only been retrieving the domain names of websites. However, the requirements of this programming assignment needed the full link. To bypass this, I took out the getDomainName function from the web crawler and simply stored the full link in my results set.

The final major problem was due to the change in HeapSort to take a Rank object, I had to define what heapIncreaseKey meant for each of the SearchTerms and WebPageURL objects. What I ended up doing was checking if the objects stored in the ArrayList were an instance of WebPageURL, then increase the index's value to the key. If it was an instance of SearchTerms, I would increase the index's value by 1.

Lessons Learned

While the essence of this project was HeapSort, this project taught me a lot more than just the implementation of all HeapSort methods. I learned to implement interfaces in order to create a generic type for HeapSort's methods to work on multiple objects. This also helped me learn how to make my code and methods a little more flexible so I could use methods on multiple objects.

The Introduction to Algorithms book also used negative infinity to create a minimum empty node. I learned that there is no negative infinity but that you can reference the smallest integer value by using Integer.*MIN_VALUE*.

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 Collections also has some very important and useful methods. I used Collections.reverse() when retrieving my ArrayList after calling HeapSort so that way for the top 10 searches, instead of displaying in ascending order, the top 10 would be displayed in descending order.