About the Simulator and Implementation:

In this simulator, Heap Sort and a priority queue are used to sort a list of URLs based on a score calculated and assigned to them; these scores are based on the average of the URL's rank in four categories: frequency of keyword found in the URL, the amount of time the URL page has existed on the internet, the amount of other pages that reference the URL, and the amount of money paid by the owner of the URL to increase in priority. To retrieve the URLs, a webcrawler is used in addition to the other classes needed. The webcrawler was retrieved online and adjusted to suit the program.

Four classes were made by hand: KeyWord, Simulator, SimulatorTest1, and SimulatorTest2. KeyWord is used to implement the URL with two parameters. Simulator, on the other hand, is the actual sorting and queuing. An ArrayList is used to store the URLs that will be sorted and eventually return sorted. SimulatorTest1 and SimulatorTest2 are two separate tests done on the program. The actual simulator can be done on SimulatorTest2.

Classes and Calls:

Spider Class and SpiderLeg Class

The Spider Class and SpiderLeg class are two classes used as a webcrawler for the assignment. According to the creator of the two classes, the Spider class serves to make sure all the websites that have been found are accounted for and are not looked up again. The other class, SpiderLeg, serves to actually "crawl" through the web. In order for the code to work, an external .JAR file called jsoup must be imported.

The Spider class returns the list of URL links in a HashSet. It will eventually be turned into an ArrayList in order to be sorted. The Spider class takes on two parameters: the Webpage URL as a String, and the keyword being searched as a String. For simplicity, google.com is used for the simulator. The SpiderLeg has no constructor or parameters.

More information about how the two classes are implemented can be found in the link below

- Both classes were found online at: http://www.netinstructions.com/how-to-make-a-simple-web-crawler-in-java/

KeyWord Class

The KeyWord Class is used to store the URLS found by the Spider and SpiderLeg class. KeyWord has two parameters that are listed below, and these parameters are used to identify the URL and assign a score that will be used to compare different URLs with one another. The score is based on the average of the ranking given to each parameter; for simplicity sake, the number assigned to each rank is random.

- public KeyWord(String word, int score)
    - The constructor of the KeyWord class with its parameters.
    - Parameters:
        - word: The URL itself
        - score: This will be the key used to compare with other keywords. The score is the average of the parameter scores rounded down.
- public String getWord(), public int getScore()

    - All the “get” methods of the class. Returns whatever is needed based on the method called.
- public void setWord(String word), public void setScore(int score)
    - All the “set” methods of the class. Sets the variable as the parameter listed in the method.

Simulator Class

The Simulator Class contain methods that aid in performing heapsort and creating priority queues. The class takes in an Arraylist of KeyWord and uses the methods in both Simulator and KeyWord to implement the methods needed.

- Instance:
    - private ArrayList<KeyWord> keywords;
    - private int heapSize;
    - private boolean isSorted
- public Simulator(int capacity)
    - The constructor of the KeyWord class with its parameters.
    - Sets the ArrayList keywords about to be sorted
    - Sets the boolean as false so that it can return true when sorted.
    - Parameter:
        - capacity: the initial size of the heap and Array
- private int parent(int index), private int left(int index), private int right(int index)
    - These methods serve to provide the location of different nodes based on the index of a given node. The method takes the index as the parameter and returns accordingly depending on what method is called.
    - parent(int index) will return the parent of the node belonging to ‘index’ by dividing the index by two.
    - left(int index) will return the left child of the node belonging to ‘index’ by multiplying the index by two.
    - right(int index) will return the right child of the node belonging to ‘index’ by multiplying the index by two before adding one.
- public KeyWord Maximum(ArrayList<KeyWord> A)
    - Is part of the priority queue. Returns the maximum of the heap
- public void ExtractMax(ArrayList<KeyWord> A)
    - Is part of the priority queue. Takes out the maximum value of the heap and adjusts the heap accordingly afterwards
- public void Insert(ArrayList<KeyWord> A, KeyWord e, int f, int t, int o, int p)
    - Is part of the priority queue. Inserts a KeyWord into the heap and then adjusts the heap accordingly afterwards.
    - The method allows the user to input in the scores for each rank (labeled in order as frequency, time, appearance on other pages, and payment) as parameters.
- public void Heapsort(ArrayList<KeyWord> A)
    - Actual sorting.
- public void IncreaseKey(ArrayList<KeyWord> A, int i, KeyWord key, int f, int t, int o, int p)

    - Is part of the priority queue. Used to change the score of the key and maintain heap properties.
- public void BuildMaxHeap(ArrayList<KeyWord> A)
    - Takes the Array and builds a MaxHeap
- public void maxHeapify(ArrayList<KeyWord> A, int i)
    - Used to make sure MaxHeap properties are still in place by making sure that the children of the node is not bigger than itself. Uses recursion to continuosly switch until MaxHeap properties are restored.

SimulatorTester1 and SimulatorTester2 Class

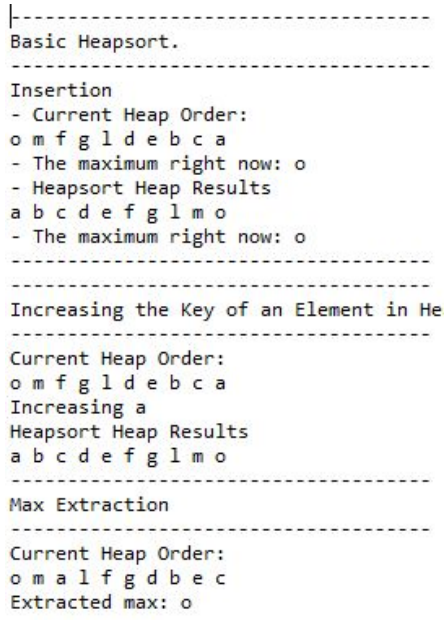
These two classes serve as tester classes for the program. I created two so that I can make a good assumption that I will have similar results in another class. The screenshots in this report are from both classes.

SimulatorTest1 was created to simply see if the sort works and so that the actual program would not look cluttered with multiple tests. It's straight to the point and inputs are letters and the scores are based on their ranking in the alphabet, so the order is correct if it is in alphabetical order. Some methods aren't really tested (maxHeapify,parent, left, right) since they are used in the many other methods that are tested, so it is safe to assume that they also work because the methods that are tested are able to run well.

SimulatorTest2 is the actual program. SimulatorTest2 uses Spider and SpiderLeg to get the webpages needed by asking the user to input a keyword the user wants to use; the results are then converted into an ArrayList. In both SimulatorTests, the variable for the KeyWord is initialized as 0. This doesn't create a problem as it is simply put there and will be calculated and changed by the user while the program runs. User input is done by scanner, and appears once the webpages are retrieved. The user will input the scores for each URL found, and the program will return the top ten scored links with the highest score on top. The program runs on a loop to represent a typical day of searching, and stores the keyword that had been inputted after each run. After a while, the program will then run a some code that will store the keywords and order them by uniqueness (uniqueness based on order in which it was entered).

Screenshots

- SimulatorTest1



```
------------------------------------
Basic Heapsort.
------------------------------------
Insertion
- Current Heap Order:
o m f g l d e b c a
- The maximum right now: o
- Heapsort Heap Results
a b c d e f g l m o
- The maximum right now: o
------------------------------------
------------------------------------
Increasing the Key of an Element in He
------------------------------------
Current Heap Order:
o m f g l d e b c a
Increasing a
Heapsort Heap Results
a b c d e f g l m o
------------------------------------
Max Extraction
------------------------------------
Current Heap Order:
o m a l f g d b e c
Extracted max: o
```

(Simple method check)

- SimulatorTest2

```
**Visiting** Received web page at https://
Found (362) links
Searching for the word hello...

**Visiting** Received web page at https://
Found (11) links
Searching for the word hello...

**Visiting** Received web page at https://
Found (48) links
Searching for the word hello...

**Visiting** Received web page at https://
Found (11) links
Searching for the word hello...

**Visiting** Received web page at https://
Found (199) links
Searching for the word hello...

**Visiting** Received web page at https://
Found (39) links
Searching for the word hello...

**Visiting** Received web page at https://
Found (18) links
Searching for the word hello...

**Visiting** Received web page at http://w
Found (39) links
Searching for the word hello...

**Done** Visited 30 web page(s)
For https://play.google.com/?hl=en&tab=w8
Enter the URL's frequency score
```

(Webcrawler storing 30 search results)

```
Searching for the word so...

**Visiting** Received web page at https://www.google.com/intl/en/about/?fg=1&utm_source=google.com&utm_medium=referral&utm_campaign=hp-header
Found (50) links
Searching for the word so...
**Success** Word so found at https://www.google.com/intl/en/about/?fg=1&utm_source=google.com&utm_medium=referral&utm_campaign=hp-header

**Done** Visited 2 web page(s)
For https://www.google.com/intl/en/about/?fg=1&utm_source=google.com&utm_medium=referral&utm_campaign=hp-header
Enter the URL's frequency score
1
Enter the URL's time score
1
Enter the URL's other webpages score
1
Enter the URL's payment score
1
URL #1's total score has been calculated.
----------------------------------------
For https://google.com/
Enter the URL's frequency score
1
Enter the URL's time score
1
Enter the URL's other webpages score
1
Enter the URL's payment score
1
URL #2's total score has been calculated.
----------------------------------------
The maximum: https://www.google.com/intl/en/about/?fg=1&utm_source=google.com&utm_medium=referral&utm_campaign=hp-header with a score of 1
Top Priority Searches
Rank #1: https://www.google.com/intl/en/about/?fg=1&utm_source=google.com&utm_medium=referral&utm_campaign=hp-header (score: 1)
Rank #2: https://google.com/ (score: 1)
1
Please enter a keyword you would like to look up:
```

(User input)

```
Enter the URL's frequency score
1
Enter the URL's time score
9
Enter the URL's other webpages score
2
Enter the URL's payment score
8
URL #9's total score has been calculated.
----------------------------------------
For https://google.com/
Enter the URL's frequency score
1
Enter the URL's time score
5
Enter the URL's other webpages score
3
Enter the URL's payment score
6
URL #10's total score has been calculated.
----------------------------------------
The maximum: https://www.google.com/imghp?hl=en&tab=wi with a score of 6
Top Priority Searches
Rank #1: https://www.google.com/imghp?hl=en&tab=wi (score: 6)
Rank #2: https://accounts.google.com/ServiceLogin?hl=en&passive=true&continue=https://www.google.com/ (score: 6)
Rank #3: https://store.google.com/?utm_source=hp_header&utm_medium=google_oo&utm_campaign=GS100042 (score: 5)
Rank #4: https://www.google.com/intl/en/about/?fg=1&utm_source=google.com&utm_medium=referral&utm_campaign=hp-header (score: 4)
Rank #5: https://www.google.com/intl/en_us/policies/privacy/?fg=1 (score: 4)
Rank #6: https://mail.google.com/mail/?tab=wm (score: 4)
Rank #7: https://www.google.com/# (score: 3)
Rank #8: https://google.com/ (score: 3)
Rank #9: https://www.google.com/intl/en/options/ (score: 2)
Rank #10: https://www.google.com/intl/en_us/policies/terms/?fg=1 (score: 2)
```

Priority search results

```
Rank #5: https://mail.google.com/mail/?tab=wm (score: 1)
10
Most Unique Searches Today:
two hi bye google 1 so you so so you
```

Top 10 “unique” searches (input order: so, two, you, hi, so bye, google, so, 1, you, ten)

Unzipping, Installing, and Running the Program

To run this program, simply unzip the folder by extracting it to whatever location you desire. Because the program requires the Jsoup .jar in order to run, the file includes a Jsoup .jar file that can imported to the library. Importing the .jars simply involve adding the jars into the library of the project.

When running, the console will ask for the keyword you are looking for. You will then be allowed to enter the scores of each URL when prompted to. A list of the top 10 highest scored links (priority search) will appear along with their rank and score. For now, it is best to have any word used as a keyword or term lowercase and for the score to be ints. The Spider class has the Webpage default as google.com due to time constraints.

The program will run for a while, to represent a typical day going by. When it end, it will return a list of the keywords looked up and ordered by uniqueness. Uniqueness is based on order of which it is found and then whether the search was already done beforehand or not.

Problems Encountered during the implementation

Sentinel and Indexing

- Figuring out how to implement the sentinel value found in the method Insert(ArrayList<KeyWord> A, KeyWord e, int f, int t, int o, int p) was difficult, as the value that would represent infinity is hard to understand and implement. It is also difficult since some of the methods in the Simulator class involve using the method. It didn't really help too much when I have to remember that the pseudocode's indices do not match the computer's implementation. It took a while before I figured out what I was suppose to do. Indexing in general is quite difficult to keep track of when the code itself has to be based from zero.

Testing

- Testing turned out to be difficult than I thought. When I made two test, I made it with the assumption that if it passed the "theoretical" SimulatorTest1, it can pass SimulatorTest2 with minor changes. However, it turns out the minor changes were pretty difficult to implement as the code would calculate the scores incorrectly if done a certain way. Because the program requires user implementation, I had to keep that in mind when implementing the program.

Lessons Learned

Even when I planned out the entire time to finish this programming assignment I still did not have enough time to do it. If I had to do this again, I would have actually make it more neater and cleaner for the user to implement. I would improve on my program by allowing the user to search whatever website they want and possibly make it appear better (though my GUI skills are terrible). I think there is still some errors and bugs that haven't been found yet, since I only tested the code with a general case and have not considered any possible corner case. Still, I did learn how a data structure can be reorganized through a sorting algorithm.