

Google Search Engine Simulation

Programming Assignment 1

March 24, 2019

# 

# 

# By: Arielle Shnayder

CS146-07

San Jose State University

Table of Contents

Design and Implementation………………...…….……………..……………………………..1

Classes and Methods………………………………………………………………………...2-5

1. HeapSort Class……………………………………………..……………………..…2-3
2. WebCrawler Class…………………………………………………………....…..……4
3. PageURL………………………………………………………………………....….4-5

Screenshots…………………………………………………………………………………6-12

Procedure to Unzip……………………………………………………………………...……13

Encountered Problems…………………………………………………………………..……13

Lessons Learned…………………………………………………………………….………..14

# 

# Design and Implementation

My Google Search Engine Simulation allows a user to search for a keyword using google.com.

Using only three classes, the user is allowed to enter a keyword, view top 30 URL links, and see the

top 30 links sorted using heapsort. The search engine consists of the following classes: WebCrawler,

HeapSort(main), and PageURL. The main method uses an ArrayList with type PageURL named

ArrayList to store the URL links from WebCrawler class. The PageURL class has methods that set

rank numbers randomly as well as a print method which print the URL along with the page rank.

Main → creates webCrawler object → stores 30 URL’s in array list → sorts by page rank → extracts

max 20 → allows user to pay to increase a site → allows user to check the max url

# 

# Classes and Methods

**HeapSort Class-** Includes main and all heap sort algorithms.

Variables-

1. HeapSize: Main passes an integer value through the constructor and heapSize is set to that value.

Methods-

1. **public** **static** **void** outputLinks(ArrayList<PageURL> A): prints first 30 elements in the ArrayList using printeverything method from PageURL.
2. **public** **static** **void** outputLinks1(ArrayList<PageURL> A): prints 20 elements using the PriorityQueue ArrayList using printeverything method from PageURL.
3. **public** **static** **void** main(String[] args): Creates two ArrayLists with type PageURL. Uses scanner input to ask the user if they want to see top 30 URLs. Creates a WebCrawler object named webby. Using webby search method, and using a for loop the URL’s copy to my ArrayList. Creates a HeapSort object passing 31 for the heapSize. Then builds max heap and sorts the ranks by heapsort method.
4. **public** **int** Parent(**int** i): Passes through index i and returns the floor of index/2
5. **public** **int** Left(**int** i): Passes through index i and returns 2\*index
6. **public** **int** Right(**int** i): Passes through index i and returns 2\*index+1
7. **public** **void** HeapSize(**int** size): Passes size and sets HeapSize to that size that was passed.
8. **public** **void** MaxHeapify(ArrayList<PageURL> A,**int** i): Passes ArrayList and index. This helps order the heap so that it organizes it using heap property.
9. **public** **void** BuildMaxHeap(ArrayList<PageURL> A): Passes ArrayList and builds a max heap; also ensures that all parent nodes are greater than their children. Calls MaxHeapify recursively.
10. **public** **void** Heapsort(ArrayList<PageURL> A): Passes ArrayList and calls BuildMaxHeap then it sorts elements is ascending order.
11. **public** **void** MaxHeapInsert(ArrayList<PageURL> A, **int** key): Priority Queue that passes ArrayList and a key. Allows the user to insert an element to the max heap.
12. **public** PageURL HeapExtractMax(ArrayList<PageURL> A): Passes ArrayList andreturns the largest element in the max heap.
13. **public** **void** HeapIncreaseKey(ArrayList<PageURL> A,**int** i,**int** key): Passes ArrayList, index i, and key. This function increases the key value of the given index and after reorders the heap tree.
14. **public** PageURL HeapMaximum(ArrayList<PageURL> A): This passes ArrayList and returns the largest element in the heap.

**WebCrawler Class-** This class was given by Professor Wu. Allows user to input a keyword then it searches google.com for it and stores all URL links found in a hashset.

Methods-

1. **public** **void** search(): This method starts the search using the keyword and calls the method crawl. Then it prints out a success message if the word was found.
2. **public** String getDomainName(String url): This method will extract the domain and returns the url that has been cut to only 27 characters.
3. **public** Set<String> getUrls(): This method gets the set of URL links in the hash set.
4. **public** **boolean** crawl(String url): This method passes a string and goes through the web and puts the links in a hashset.
5. **public** **boolean** searchForWord(String searchWord): This method checks if the website contains the keyword.

**PageURL Class-**

Variables-

1. moneyPaid: integer for the amount of money someone pays
2. siteAge: integer for the amount of years a site has been on google
3. Frequency: integer for the amount of times a keyword is searched
4. linkRef: integer for the reference of the link
5. String URL: String for the URL
6. i: integer for the index
7. getRank: integer that stores the total rank number

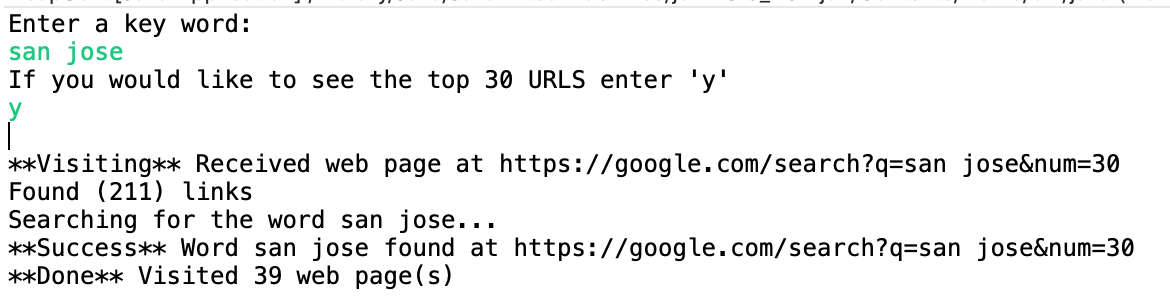
Methods-

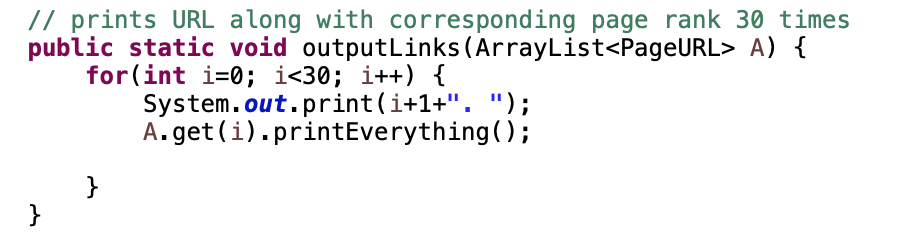
1. **public** PageURL(): This method initializes moneyPaid, siteAge, frequency, and linkRef to zero.
2. **public** PageURL(String URL): This method is also a constructor that only passes the string URL and sets moneyPaid, siteAge, frequency, and linkRef to random integer values.
3. **public** **void** add(**int** i): Passes index i and sets moneyPaid to that index.
4. **public** PageURL(**int** key): Passes a key and sets moneyPaid to that key.
5. **public** **double** getMoneyPaid(): This method returns the value of moneyPaid.
6. **public** **double** getsiteAge(): This method returns the value of siteAge.
7. **public** **double** getFrequency(): This method returns the value of frequency.
8. **public** **double** getLinkRef(): This method returns the value of the linkRef.
9. **public** String getURLName(): This method returns the URL string.
10. **public** **double** getRank(): This method returns the sum of moneyPaid, siteAge, frequency, and linkRef.
11. **public** **double** getRank(int i): This method returns the sum from above and adds the new price the user entered.
12. **public** **void** printEverything(): This method prints the URL link and the page rank below it.

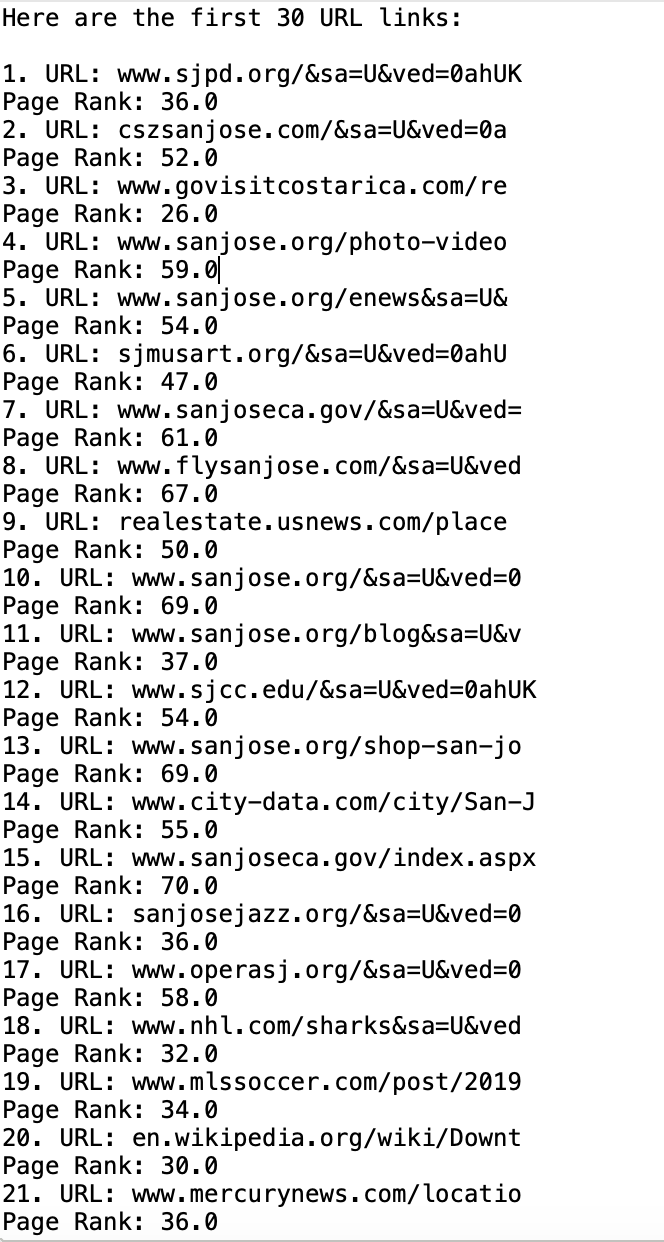
# 

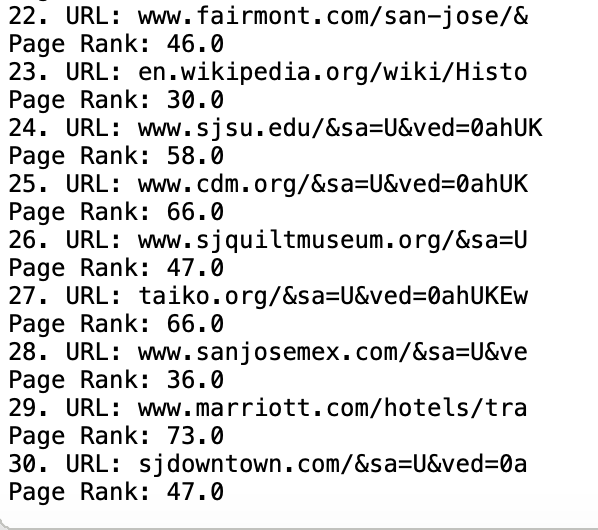
# Screenshots

**Screenshot 1:** Allows user to enter keyword and passes it to webCrawler object as shown below in screenshot 2.

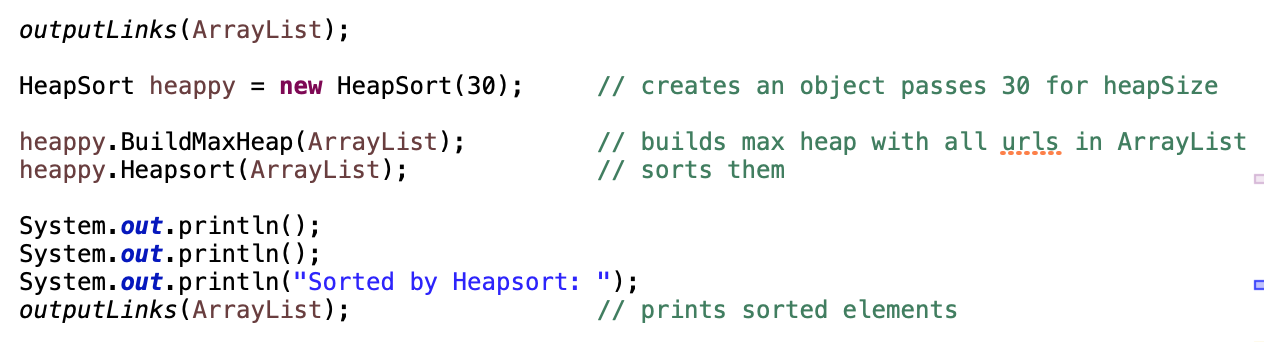
**Screenshot 2.** Allows user to enter their own keyword. Asks the user to enter ‘y’ if they want to display the top 30 URL’s. Then it displays how many links were found and how many web pages it has visited. 



**Screenshot 3:** Calls this function and passes in ArrayList to print the URL along with the page ranks. 

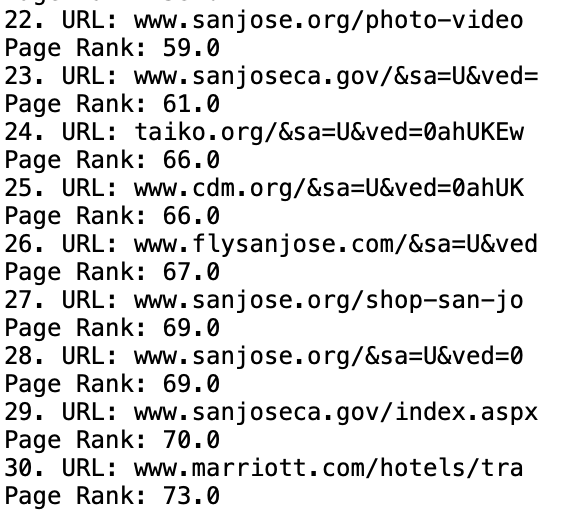


**Screenshot 3:** Displays the 30 URL’s with their corresponding page rank number in unsorted order.

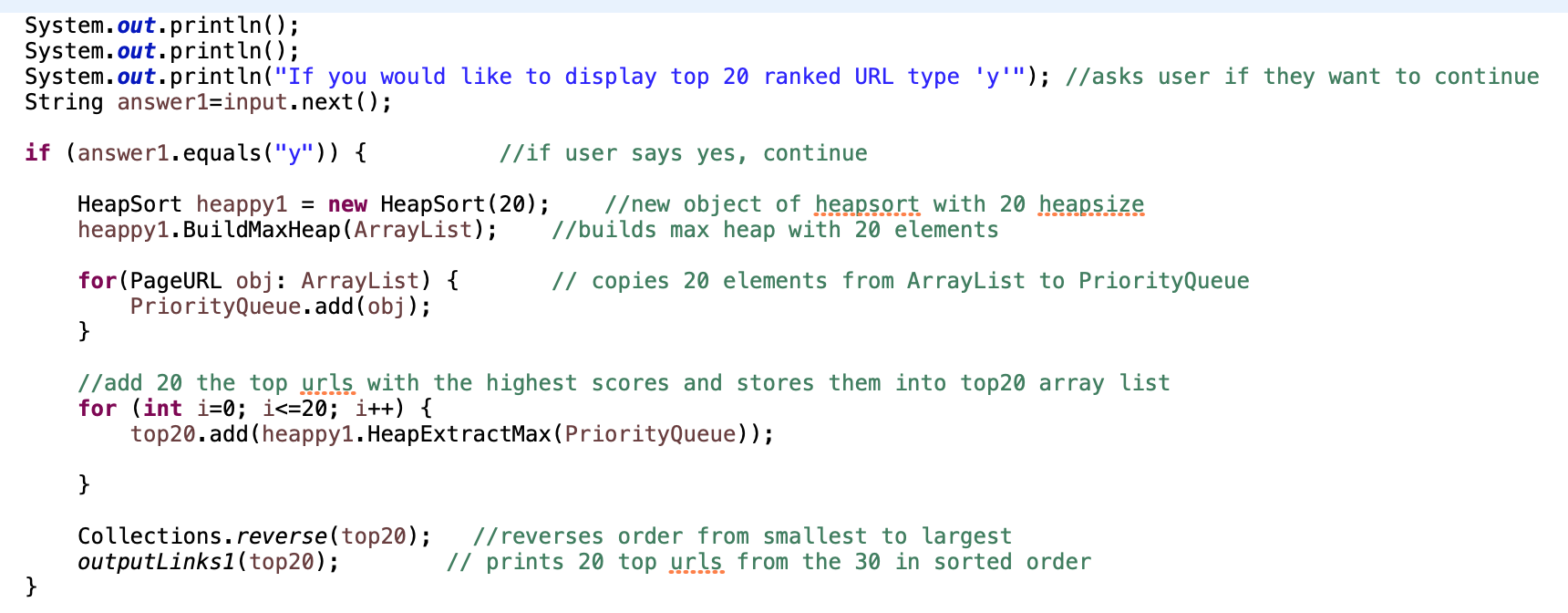


**Screenshot 4:** This is the code for printing the sorted heapsort as shown below in screenshot 5.



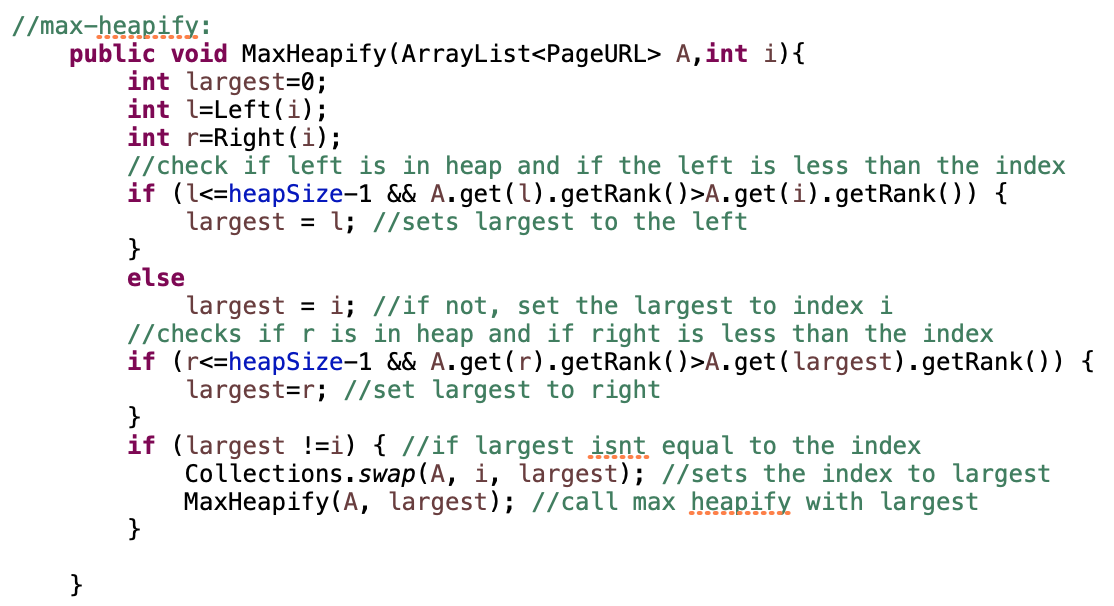


**Screenshot 5:** Shows the same 30 URL’s in sorted order after heapsort is called.

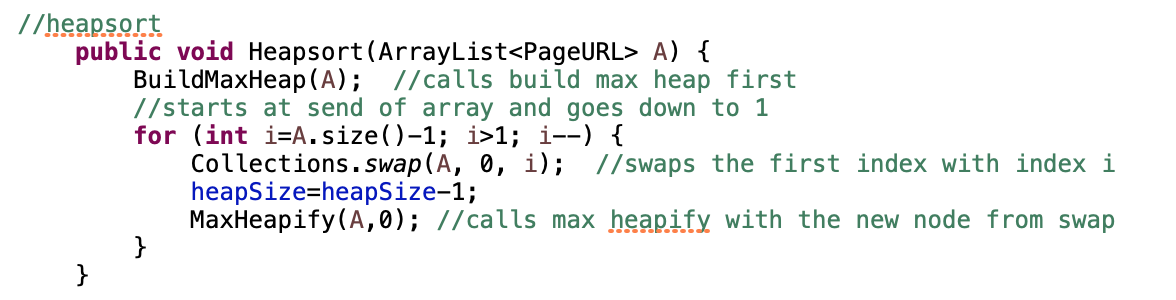


**Screenshot 6:** This code copies the top 20 URL’s from ArrayList to top20 array list. As shown below.

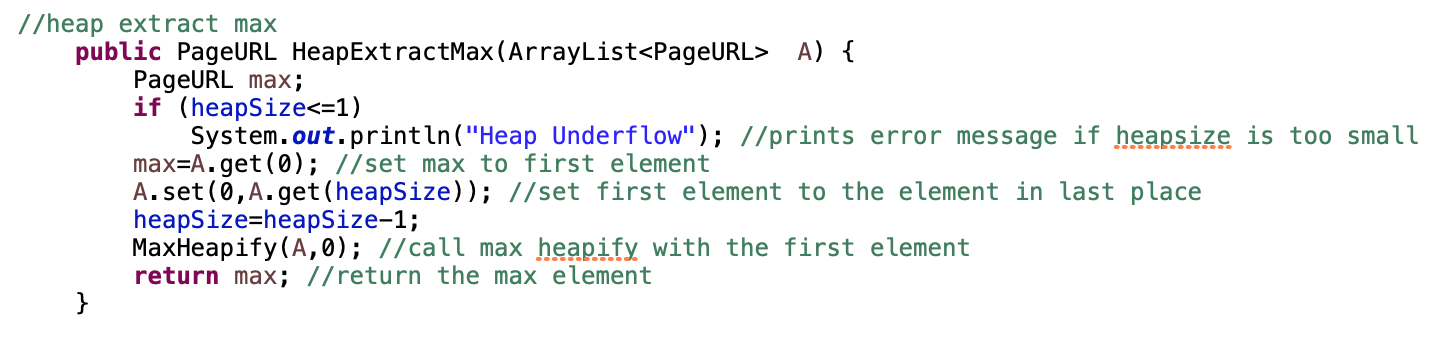
**Screenshot 7:** Displays the top 20 URL’s from the 30 sorted elements. Has highest(page rank) 20 elements.



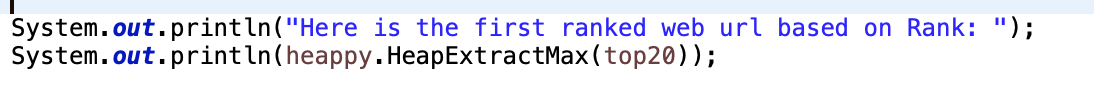
**Screenshot 8:** Max Heapify Method



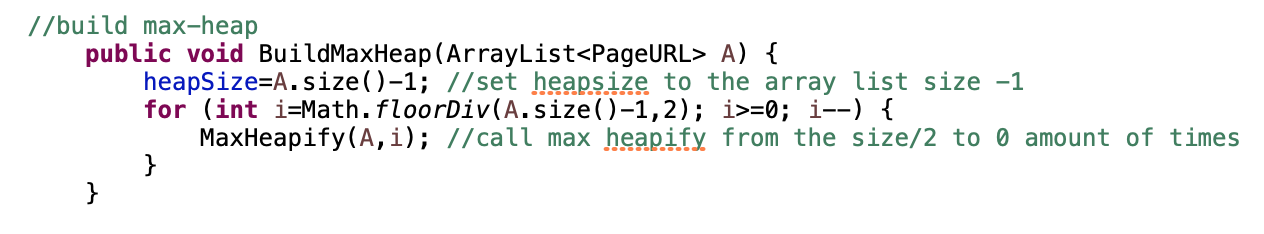
**Screenshot 9:** Heap Sort Method; calls max heapify above

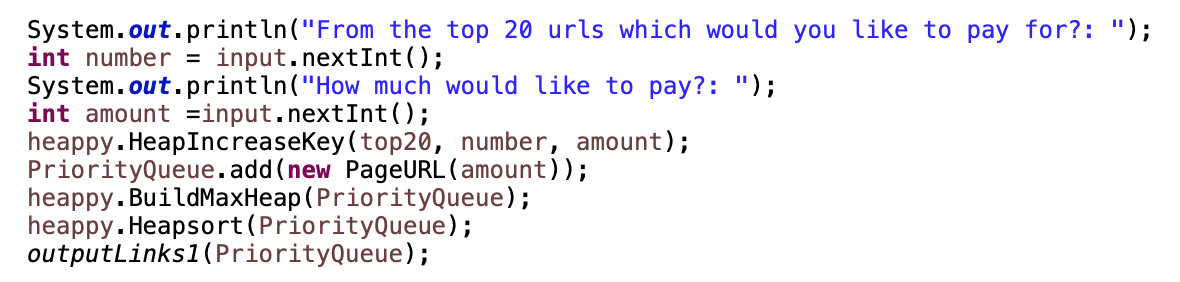


**Screenshot 10:** Heap Extract Max that is used below in screenshot 11.

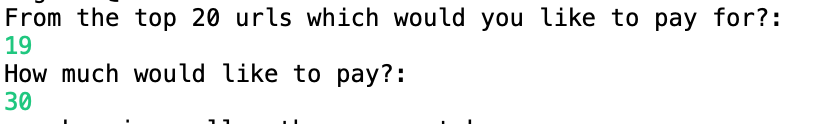


**Screenshot 11:** Extracts max from the 20 urls and prints it.

**Screenshot 12:** Builds Max Heap used in main before sorting. 



**Screenshot 13:** In main, allows user to enter which url they would like to increase and by how much.



**Screenshot 14:** Allows user to enter.

# 

# Screenshot 15: This prints out the largest URL based on rank.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Procedure to Unzip

1. Unzip folder
2. Open terminal
3. Change to the directory of the project
4. Type “ls” to make sure HeapSort.jar is inside that directory
5. Type “java -jar HeapSort.jar”
6. Follow instructions from the project

# 

# Problems Encountered

I encountered many problems throughout the process of building this simulator. The first problem I

encountered was implementing an array list instead of an array. This was challenging because all my

heap sort methods passed through an array not arraylist. In order to fix this I figured out how to make

an array list using my PageURL as the type. I also had to change the heap sort methods inside to

get the rank of the arraylist index. Another problem I encountered was storing the first 30 links in

my array list. I figured out that in order to do this you had to call the webCrawler object and

make an if statement to check when the arraylist size is equal to 30.

The second major problem I had was using the heap max extract method to print only the top

20 URL’s because I would get index out of bounds errors. When I tried running the max heap extract,

my output for the sorted 30 would get messed up. In order to fix this, I created a new heapSort object

and passed 20 for the heap size. I then had a for loop and added the urls using the extract max to my

new array list I made. Then I used the reverse call from Collections to sort them in ascending order.

After I finished part a of number two in the assignment, I tried implementing heap increase key

To allow the user to pay for a specific site of their choice. I added print statements and accepted the

site number as well as how much they wanted to pay. Then, I called HeapIncreaseKey passing my

Array list that had top 20 urls, the site number the user entered, and lastly the amount of which

the user wants to pay for that site. I ran build max heap and heapsort after that so re organizes with

the new page rank number.

# 

# Lessons Learned

While building this project, I learned many things about programming and about myself. First thing

I learned was overall how heap sort works with all its methods working together and how some methods

work recursively. As well, I understood how to call different classes using objects. I also learned many

methods that were from the import such as Collections, I used the reverse. Also, I learned how to

implement an array list instead of using an array. I had to use the arraylist.get whatever. In PageURL I

learned that it is better to initialize variables in the constructor and then have get methods for each

page rank; such as get linkRef or siteAge. At the end of that class I figured out that it would be best

to have a print statement that prints the URL along with the page rank it calculated in the same class.

I was unable to implement some of the requirements, however, I did try my best to try out different

code to see if something would work and that led me to understanding more about the over all code I had.

This project was very interesting to me because I saw the progress I was making in each step and it was

something I had not done in the past. I know have a better understanding about how a google search

engine works.