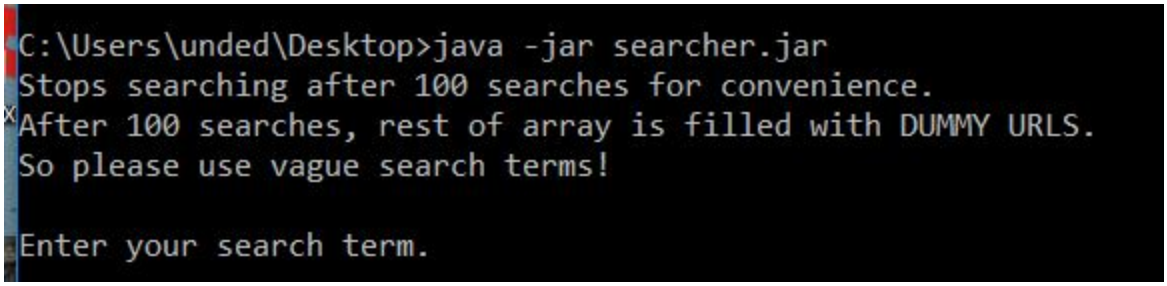
TABLE OF CONTENTS

Compressing ruins the quality of screenshots. Here is the source document for higher quality.

https://docs.google.com/document/d/1V7_6txj3qSxJ5hqtNTFYf0rR8bvutLWz9TPCu0wAtdM/edit?usp=sharing

HOW TO START:

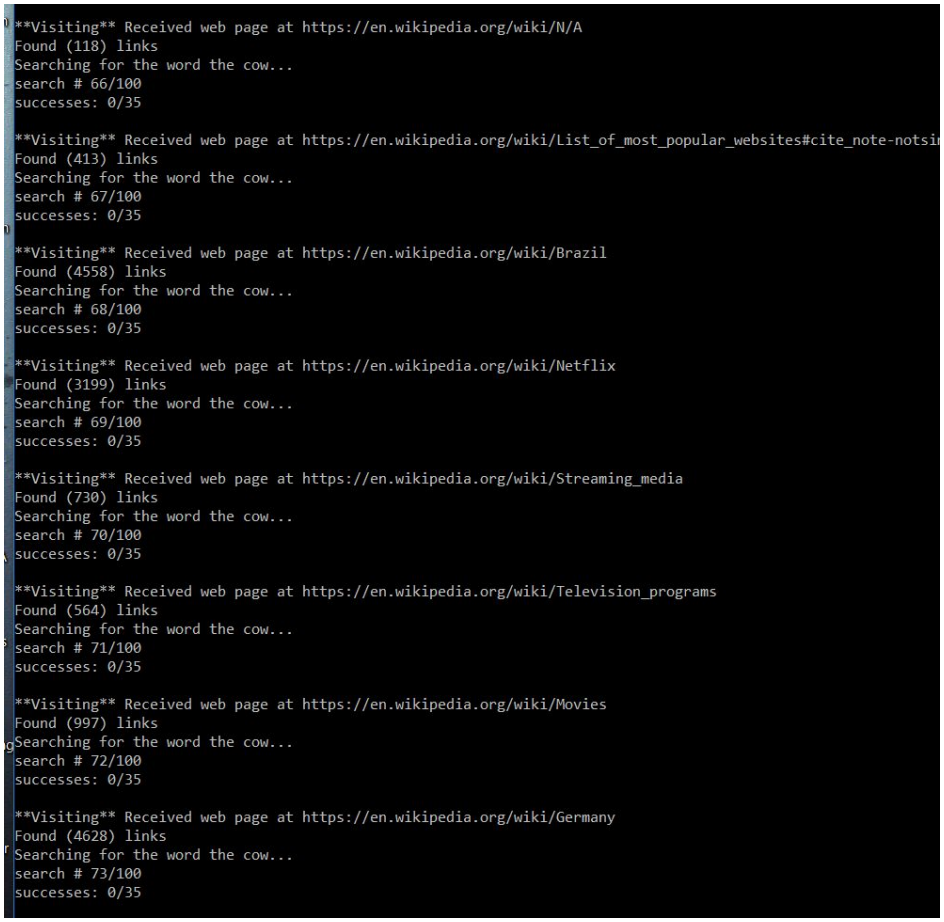
I use winrar to extract and zip my files.



1) Extract the files (With winrar, right click the zip and click “extract here”)
2) Open command prompt
3) Navigate to directory where searcher.jar is. I held it in my desktop
4) Type “**java -jar searcher.jar**”
5) The program will start in your command prompt

…

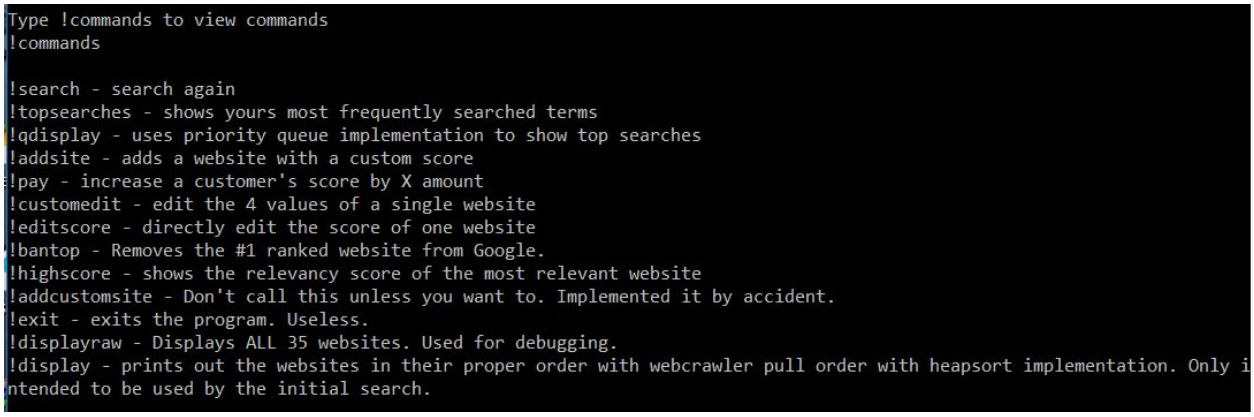
1) Enter your search term, as prompted.



^ If you did it right, you should see that.

Note: If the searcher goes through 100 searches and does not find 35 websites with the term you want, it will fill the array up with dummy URLs pulled from the unsuccessful websites. This is to save time instead of looping forever.

Type "!commands" for a list of commands



!display, !displayraw, and !addcustomsite are optional and not recommended
If a prompt says enter an INTEGER, do not enter a String.

Examples of how to use the program can be found in the test cases where I test my program within the command prompt. It will be on the first page of the testing chapter, since those are the most important tests. Please look there for examples.

Note: There are two implementations of displaying the top 10 results. One uses heapsort and prints from the sorted array, another uses heap_extract_max 10 times and puts it into a priority queue, which it is printed from.

!qdisplay uses the queue implementation and is the recommended command for displaying. It only displays the google pagerank followed by the name.
!display uses heapsort to sort the array and prints from the sorted array. It will display the order it entered the webcrawler, google pagerank, url, and score

IMPLEMENTATION SUMMARY

Program runs → Enter your term → spider searches → spider returns an array string[] of successful searches → array string[] is converted to Url objects. Each Url is randomly assigned the variable values that go into calculating their total score → Url objects are put into an array of Url[] → a heap is created with that array of Url[] → the heap manages the array → the commands you use change the array inside the heap class and display the changes you made.

For search tracking: Every new search you enter with !search is stored in an array. Calling !topsearches finds the frequency of every search, puts every unique search into a SearchFrequency object with the number of searches that that search has as a field variable, and then puts an array of SearchFrequencies into a heap to be sorted. The top 10 most popular results are displayed.

Note: I thought about another implementation where every time you make a search, it scans a map for that term. If found, increment its key by 1, if not found, add it to the map and set its key to 1. Even though that sounds like O(n), I'm not sure if the fact that it has to do it for every search throughout the day makes it technically O(n)^2.

CLASSES AND METHODS

Spider class

Imagine you entered a term. For the root website, it searches a wikipedia page and all the websites linked by it for that term. If the term is found on the website it is on, the website's URL is added to an array of Strings[]. If there are 100 searches (value can be changed by going into the code) and the array of successful Urls is not full, then it fills the remaining slots in the array with dummy Urls, pulled from the list of unsuccessful searches. I chose to do this instead of having it run forever because it saves time.

```
 */
public String[] search(String url, String searchWord) {
    while (this.pagesVisited.size() < MAX_PAGES_TO_SEARCH && breaker < MAX_PAGES_TO_SEARCH) {
        System.out.println("search # " + breaker + "/" + MAX_PAGES_TO_SEARCH);
        System.out.println("successes: " + successes + "/" + SUCCESSES_NEEDED);
        breaker++;
        String currentUrl;
        SpiderLeg leg = new SpiderLeg();
        if (this.pagesToVisit.isEmpty()) {
            currentUrl = url;
            this.pagesVisited.add(url);
        } else {
            currentUrl = this.nextUrl();
        }
        leg.crawl(currentUrl); // Lots of stuff happening here. Look at the crawl method in
                               // SpiderLeg
        boolean success = leg.searchForWord(searchWord);
        if (success) {
            System.out.println(String.format("**Success** Word %s found at %s", searchWord, currentUrl));

            // if success, add that successful URL to the URL you were searching.
            urlsFound[successes] = currentUrl;

            successes++; //self explanatory if statement
            if (successes == SUCCESSES_NEEDED) {
                break;
            }
        }
        this.pagesToVisit.addAll(leg.getLinks());
    }
    System.out.println("\n**Done** Visited " + this.pagesVisited.size() + " web page(s)");
```

^ I don't know what most of the Spider code does, but I realized that the person who coded it has an if(success) statement that applies when the term is found on the website. So I decided to take advantage of that and instead of just printing the currentUrl, add it to a list of successes and increment a "success" variable to keep track of the number of the number of successes. Once it reaches the number of successes it needs, it leaves the loop.

```
while (this.pagesVisited.size() < MAX_PAGES_TO_SEARCH && breaker < MAX_PAGES_TO_SEARCH) {
    System.out.println("search # " + breaker + "/" + MAX_PAGES_TO_SEARCH);
    System.out.println("successes: " + successes + "/" + SUCCESSES_NEEDED);
    breaker++;
    String currentUrl;
```

^ The "breaker" variable is to keep track of the number of websites searched. That's why it ends at 100 and not into infinity.

```
/**
 * Finished max # of searches
 * Enters "if" statement if you could not find the number
 * URLs you wanted (35) after the max attempts of searches (100-500)
 * Fills the rest of the array with Dummy URLs because term was too vague.
 */
if (successes < SUCCESSES_NEEDED) {
    //prevents error in the next for loop.

    /**
     * fills the rest of the array with dummy URLs
     */
    Iterator<String> iterator = pagesVisited.iterator();
    for (int i = successes; i < urlsFound.length; i++) {

        String dummyUrl = "null2";

        //(gives it a base URL to make dummy from?)
        if (iterator.hasNext()) {
            dummyUrl = iterator.next();
        }
        dummyUrl = "DUMMY_" + i + "_" +dummyUrl;
        urlsFound[i] = dummyUrl;
    }
}
```

^ If the array of successes is not full, add dummy URLs to fill up the rest of the array. I get the dummy URLs from the list of unsuccessful websites. Dummyurl is initiated as "null2" so that if a website URL is named null2, I have an idea of where the error is from.

```
Google PageRank: 5
PageRank BEFORE being sorted: 16
DUMMY_15_https://en.wikipedia.org/wiki/E-commerce_payment_system
Score: 337

Google PageRank: 6
PageRank BEFORE being sorted: 27
DUMMY_26_https://en.wikipedia.org/wiki/XVideos
Score: 336

Google PageRank: 7
PageRank BEFORE being sorted: 32
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
Score: 328
```

^examples of dummy urls.

Url class

It is a simple class but the function is important. From spider class, search() returns an array of successful Strings. When the program is running, it will take that array of strings and turn every string into a Url object with a random score assigned to it. This array of Urls is passed to a UrlHeap object to be organized.

```
public class Url implements Comparable<Url> {
    int rankBeforeSort = 0;
    int score = 0;

    int years = 0;
    int links = 0;
    int keywords = 0;
    int paid = 0;

    public String name = "";

    /**
     * Makes Url object with randomized values.
     * @param name - the URL
     */
    public Url(String name) {
        years = (int) (Math.random()*100);
        links = (int) (Math.random()*100);
        keywords = (int) (Math.random()*200);
        paid = (int) (Math.random()*100);

        score = years + links + keywords + paid;
        this.name = name;
    }
```

^ The most important constructor, supply a website and turn it into a Url with a score.

UrlHeap class

To save time, I will only be talking about the methods I made edits to. Please go into the source code to view the rest, as they were taken almost directly from the textbook.
UrlHeap takes an array of Urls in the constructor to add to the object's fields so that it can organize it. It is a very normal heap, except for some functions I changed, shown below

```
public class UrlHeap {

    Url[] arr;
    int size = 0;
    int originSize = 0; //not needed?
    UrlHeap heap; //i don't know why I have this but I'm too scared to delete it

    /**
     * Makes an un-heapified heap. Call Build_Max_Heap() to turn it into a heapified
     * heap.
     *
     * @param The
     *            array you want to apply the heap functions to.
     */
    public UrlHeap(Url[] array) {
        this.size = array.length;
        this.originSize = size; //not needed?
        arr = array;
    }
```

```
public void Max_Heap_Insert(String name, int key) {
    size++;
    // this means make a new array with +1 size... :')

    // newArr[size-1] = Integer.MIN_VALUE;
    Url[] newArr = new Url[size];
    Url dummy = new Url(name, Integer.MIN_VALUE); // ensures it goes last?
    newArr[size - 1] = dummy;

    for (int i = 0; i < size - 1; i++) {
        newArr[i] = arr[i];
    }
    this.arr = newArr;

    Heap_Increase_Key(size - 1, key); // sets whatever is at size - 1 to have that new score.

}
```

^ For example, in the method above, I did not know how to handle increasing the size of an array, so I made a new array and copied all the values from the old array to the new array. Size is just a variable I keep track of in the object's fields so that I can keep adding and subtracting sizes through loops where I need to add or take away several items.

```
public int Heap_Extract_Max() {
    if (size < 1) {
        System.out.println("heap underflow @ Heap_Extract_Max");
    }
    int max = arr[0].getScore();
    arr[0] = arr[size - 1];
    size--;
    // make new array for new size
    Url[] newArr = new Url[size];
    for (int i = 0; i < size; i++) {
        newArr[i] = arr[i];
    }
    this.arr = newArr;

    Max_Heapify(0);
    return max;
}
```

^ Similarly, I make a new array instead of decreasing any heap size.
^^ For the two methods above, I have copies of them but for Urls instead of integers.
Notice that many of my UrlHeap functions deal with Urls and not integers. That is the other change I made, which is the only thing that sets it apart from a normal integer heap. Urls are compared by their score as the integer.

SortTracker, SearchHeap, and SearchFrequency classes

They all work together so that I could implement the function of keeping track of your searches and displaying the top 10 most frequent searches.

SortTracker class

When the Program is running, it puts all your searches into an ArrayList. When you call !topsearches, it passes that ArrayList into SortTracker...

```
 */
public class SortTracker {
    ArrayList<String> searches;

    public SortTracker(ArrayList<String> passed) {
        this.searches = new ArrayList<String>(passed);
    }
```

```
public SearchFrequency[] getSearchFrequencies() {
    //ties every search to a frequency. Hashmaps are sets.
    Map<String, Integer> map = new HashMap<>();
    for(String x : searches) {
        map.put(x,Collections.frequency(searches, x));
    }

    //SearchFrequency objects have the search and their frequency
    ArrayList<SearchFrequency> searches = new ArrayList<>();
    for(String x : map.keySet()) {
        SearchFrequency search = new SearchFrequency(x, map.get(x));
        searches.add(search); //I think this is pointless?
    }

    //im doing it this way because I don't know how to add from keySet to array
    //adds all items from ArrayList to Array
    SearchFrequency[] searchArr = new SearchFrequency[searches.size()];
    for(int x = 0; x < searches.size(); x++) {
        searchArr[x] = searches.get(x);
    }
    SearchHeap searchHeap = new SearchHeap(searchArr);
    searchHeap.HeapSort();
    return searchHeap.getArray();
```

^ The point of the code above is to turn the searches into an array of SearchFrequency objects. **SearchFrequency objects hold the search term and the amount of times that that term occurred.**

Java's collection has a method that returns the frequency of an item in the list it belongs to. First, a hashmap is created with the url string tied to the # of searches of that url. Then, it is converted to a SearchFrequency object, and stored in an array so that heapsort can organize it for the top 10 search terms to be printed. This is the purpose of the SearchHeap class.
The data structure that I used to sort the searches is therefore a heap.

Program class

```
public void start() {
    while (true) { //outer loop so that you can search infinite times
        initSearchAndHeap();
        while (true) {
            System.out.println("Type !commands to view commands");
            input = scan.nextLine();
            if (input.equals("!search")) { //typing !search breaks inner loop
                //and go back to searching.
                break;
            } else {
                runCommands(); //if not !search then check other commands.
            }
        }
    }
```

^I wrote the comments to explain what it does. The outer while loop keeps the user inside the program and the inner while loop repeatedly takes commands from the user. The inner while loop only exits if the user types !search, where initSearchAndHeap() prompts the user to search again, and since the inner loop exits, the heap and all the values in the heap are emptied for the new search.

```
public void initSearchAndHeap() {

    System.out.println("Enter your search term.");

    String term = scan.nextLine();
    searches.add(term);
    Spider spider = new Spider();
    String[] sites = spider.search("https://en.wikipedia.org/wiki/List_of_most_popular_websites", term);

    // takes all the sites successfully searched by spider and stores it in urls
    urls = new Url[sites.length];

    // take the urls and build class Url with scores
    for (int i = 0; i < urls.length; i++) {

        urls[i] = new Url(sites[i]);
        //raw pagerank
        urls[i].setRankBeforeSort(i);
    }
    heap = new UrlHeap(urls);
    heap.Build_Max_Heap();
    this.heapSortDisplay();
    //for some reason, if I put build max heap above heapsortdisplay, !bantop doesn't work...

    System.out.println();
    //bmh

}
```

^ We convert the string of sites into Url objects that are assigned random scores.
The UrlHeap object takes an array of urls to manage. So my for loop turns all the Strings into Url objects.

There are two implementations for displaying the top 10 results, one that uses the heap priority queue implementation, and another that uses heapsort and prints directly from the sorted array, while also printing the order it was added to the webcrawler.
!qdisplay is for the queue implementation display. **!display** is for the regular heapsort display with the order it came into the webcrawler from.

I use a clone method to get a copy of the original heap's array and add that to a second heap because extract max heap and heapsort change the size of the array and remove from the array. Since the user will keep wanting to use the array with other commands, I don't want to actually remove the values. I tried keeping track of the original size and recalling build_max_heap after doing heapsort but that made a small bug I wrote about in "problems encountered".

```java
public void qDisplay() {

    UrlHeap heap2 = new UrlHeap(heap.getArray().clone());

    PriorityQueue<Url> queue = new PriorityQueue<Url>();
    int size = heap2.getArray().length;

    if (size <= 10) {
        for (int i = 0; i < size; i++) {
            //h2.bmh
            Url extracted = heap2.Heap_Extract_Max_Url();
            queue.add(extracted);
        }
    } else if (size > 10) {
        for (int i = 0; i < 10; i++) {
            //h2.bmh
            Url extracted = heap2.Heap_Extract_Max_Url();
            queue.add(extracted);
        }
    }
    size = queue.size();
    if (size <= 10) {
        for (int i = 0; i < size; i++) {
            int rank = i + 1;
            System.out.println("Google PageRank: " + rank);
            queue.remove().printValues();
            System.out.println();

        }
    } else if (size > 10) {
        for (int i = 0; i < 10; i++) {
            int rank = i + 1;
            System.out.println("Google PageRank: " + rank);
            queue.remove().printValues();
            System.out.println();
        }
    }
```

^ !qdisplay

Priority queue implementation of printing the top 10 values with extract max instead of heapsort. There are 4 if loops instead of 2 to prevent array out of bounds error because we are displaying the top 10 searches, but sometimes there are less than 10 in the array. If you see code similar to this without explanation, assume this is the explanation.

```
public void heapSortDisplay() {
    System.out.println("calling heapSortDisplay()");
    // we use heap2 because the heapsort messes up the size variable in UrlHeaps...
    UrlHeap heap2 = new UrlHeap(heap.getArray().clone());
    heap2.HeapSort();
    Url[] sorted = heap2.getArray();
    int x = 1;

    if (sorted.length >= 10) {
        for (int i = sorted.length - 1; i > sorted.length - 11; i--) {
            System.out.println("Google PageRank: " + x);
            x++;
            // retrieves the order it came into the webcrawler
            int beforeSorted = sorted[i].getRankBeforeSort() + 1; // arrays start at 0.
            System.out.println("PageRank BEFORE being sorted: " + beforeSorted);

            heap2.printValue(i);
            System.out.println();

        }
    }

    else if (sorted.length < 10) {
        for (int i = sorted.length - 1; i >= 0; i--) {
            System.out.println("Google PageRank: " + x);
            x++;
            int beforeSorted = sorted[i].getRankBeforeSort() + 1; // arrays start at 0.
            System.out.println("PageRank BEFORE being sorted: " + beforeSorted);
            heap2.printValue(i);
            System.out.println();
        }
    }
```

^!display

this sorts a copy of the array with heapsort and displays the top 10 results with the highest Google pagerank followed by the order they were put into the webcrawler. This is used in the !search function.

```
public void runCommands() {

    if (input.equals("!highscore")) {
        highscore();
    }

    else if (input.equals("!bantop")) {
        banTop();
    } else if (input.equals("!editscore")) {
        editScore();
    } else if (input.equals("!customedit")) {
        customEdit();
    }

    else if (input.equals("!qdisplay")) {
        qDisplay();
    }

    else if (input.equals("!addsite")) {
        addSite();
    }

    else if (input.equals("!displayraw")) {
        displayRaw();
    } else if (input.equals("!pay")) {
        pay();
    } else if (input.equals("!addcustomsite")) {
        addCustomSite();
    }

    else if (input.equals("!exit")) {
        System.exit(0);
    }

    else if (input.equals("!commands")) {
        printCommands();
```

^This is how I check for which command you entered.

```
private void addSite() {
    // h.bmh
    System.out.println("Enter the URL of the new website. (STRING)");
    String url = scan.nextLine();
    System.out.println("Enter the amount of money they paid to the nearest (INTEGER)");
    int paid = scan.nextInt();
    heap.Max_Heap_Insert(url, paid);

}
```

^Adds a site by using Max_Heap_Insert

```
 */
private void addCustomSite() {
    // h.bmh
    System.out.println("Enter the URL of the new website. (STRING)");
    String url = scan.nextLine();
    System.out.println("Enter the amount of money they paid to the nearest (INTEGER)");
    int paid = scan.nextInt();
    System.out.println("Enter the number of years old they are (INTEGER)");
    int date = scan.nextInt();
    System.out.println("Enter the number of keywords this website has related to your search (INTEGER)");
    int keywords = scan.nextInt();
    System.out.println("Enter the number of links this website has to it (INTEGER)");
    int links = scan.nextInt();

    Url score = new Url(url, paid, date, keywords, links);
    heap.Max_Heap_Insert(url, score.getScore());

}
```

^ how a custom site is added.

```
3
4   public void displayTop() {
5       SortTracker tracker = new SortTracker(searches);
6       SearchHeap searchHeap = new SearchHeap(tracker.getSearchFrequencies());
7       searchHeap.HeapSort();
8       SearchFrequency[] uniques = searchHeap.getArray();
9
0       if (uniques.length <= 10) {
1           for (int i = uniques.length - 1; i >= 0; i--) {
2               System.out.println("Term: " + uniques[i].getSearch());
3               System.out.println("Frequency: " + uniques[i].getFrequency());
4               System.out.println();
5           }
6       } else if (uniques.length > 10) {
7           for (int i = uniques.length - 1; i > uniques.length - 11; i--) {
8               System.out.println("Term: " + uniques[i].getSearch());
9               System.out.println("Frequency: " + uniques[i].getFrequency());
0               System.out.println();
1           }
2       }
3
4   }
5
```

^ I explained this in my explanation of the tracker, searchHeap, and SearchFrequency classes. SortTracker takes the array of searches, converts that into an array of SearchFrequencies, which is sorted by heapsort and printed for the user.

```java
/**
 * adds x score to the score of the customer because they paid x amount
 */
public void pay() {
    // h.bmh
    this.heapSortDisplay(); // for convenience of tester
    // h.bmh

    for (int i = 0; i < heap.getArray().length; i++) {
        // h.bmh
        // id = index
        System.out.println("ID: " + i);
        heap.printValue(i);
        System.out.println();
    }
    System.out.println("Enter the ID of the customer who paid. (INTEGER)");
    int ID = scan.nextInt();
    // their new score will be old score + amount paid
    int score = heap.getScore(ID);
    System.out.println("The customer is " + heap.getOneName(ID));
    System.out.println("Enter the amount this customer paid, rounded to the nearest INTEGER");
    int paid = scan.nextInt();
    int newScore = score + paid;
    heap.Heap_Increase_Key(ID, newScore);
    System.out.println("Customer's score has been increased by " + paid);

}
```

^ Uses Heap_Increase_Key to allow you to increase the score of a website, but not decrease.

```java
/
public void customEdit() {
    // h.bmh
    // now print it out for the console viewer
    this.qDisplay(); // for convenience of tester.

    for (int i = 0; i < heap.getArray().length; i++) {
        // h.bmh
        System.out.println("ID: " + i);
        heap.printValue(i);
        System.out.println();
    }
    System.out.println("Enter the ID of the customer who you want to edit. (INTEGER)");
    int ID = scan.nextInt();
    Url recieved = heap.getUrl(ID);
    System.out.println("Customer recieved: " + recieved.getName());
    System.out.println("Score of customer: " + recieved.getScore());
    System.out.println();

    System.out.println("KEYWORDS value of this site: " + recieved.getKeywords());
    System.out.println("Enter the new KEYWORD value for this site (INTEGER).");
    int keywords = scan.nextInt();
    recieved.setKeywords(keywords);
    ;

    System.out.println("AGE value of this site: " + recieved.getYears());
    System.out.println("Enter the new AGE value for this site (INTEGER).");
    int years = scan.nextInt();
    recieved.setYears(years);

    System.out.println("LIKNKS value of this site: " + recieved.getLinks());
    System.out.println("Enter the new LINKS value for this site (INTEGER).");
    int links = scan.nextInt();
    recieved.setLinks(links);
```

^ uses getters and setters to allow you to edit the individual values of one website.
This is only half the method because it's long and redundant to screenshot.
Note: the end of this function uses build_max_heap instead of heap_increase_key because build max heap will maintain a max heap's integrity regardless of the value added to it. Now that I think about it, maybe I could have used insert_key but my insert_key takes O(n) anyways.

```
public void editScore() {
    // h.bmh
    qDisplay();

    for (int i = 0; i < heap.getArray().length; i++) {
        // h.bmh
        System.out.println("ID: " + i);
        heap.printValue(i);
        System.out.println();
    }
    System.out.println("Enter the ID of the customer who you want to edit. (INTEGER)");
    int ID = scan.nextInt();
    Url recieved = heap.getUrl(ID);
    System.out.println("Customer recieved: " + recieved.getName());
    System.out.println("Score of customer: " + recieved.getScore());
    System.out.println("Enter their new score");
    int newScore = scan.nextInt();
    recieved.setScore(newScore);
    heap.Build_Max_Heap(); // re heapifies. Can't use Increase key. Increase key only applies to bigger
                           // values.
    System.out.println("Their new score is " + recieved.getScore());
}
```

^ Lets you edit the score of a website, then rebuilds the heap so that it is placed in its necessary place. Similarly uses build_max_heap.

CALCULATIONS OF RUNNING TIMES

Spider class

Assume n = # of successful URLs added, since we use our heap on the successes.

```
                    // SpiderLeg
boolean success = leg.searchForWord(searchWord);
if (success) {
    System.out.println(String.format("**Success** Word %s found at %s", searchWord, currentUrl));

    // if success, add that successful URL to the URL you were searching.
    urlsFound[successes] = currentUrl;

    successes++; // self explanatory if statement
    if (successes == SUCCESSES_NEEDED) {
        break;
    }
}
```

^ adding all successes to list = O(n)
Worst case scenario is a flat # of searches (100) without success, in that case, runs code below

```
 */
if (successes < SUCCESSES_NEEDED) {
    // prevents error in the next for loop.

    /**
     * fills the rest of the array with dummy URLs
     */
    Iterator<String> iterator = pagesVisited.iterator();
    for (int i = successes; i < urlsFound.length; i++) {

        String dummyUrl = "null2";

        // (gives it a base URL to make dummy from?)
        if (iterator.hasNext()) {
            dummyUrl = iterator.next();
        }
        dummyUrl = "DUMMY_" + i + "_" + dummyUrl;
        urlsFound[i] = dummyUrl;
    }
```

^ worst case checks all 100 , finds zero successes, so uses O(n) time to fill list with successes.
Worst case of search() is 2n + c, which is **O(n)**

UrlHeap class

This is the heap, the time is given to us by the textbook, except some of the methods I modified. For example...

```
public int Heap_Extract_Max() {
    if (size < 1) {
        System.out.println("heap under
    }
    int max = arr[0].getScore();
    arr[0] = arr[size - 1];
    size--;
    // make new array for new size
    Url[] newArr = new Url[size];
    for (int i = 0; i < size; i++) {
        newArr[i] = arr[i];
    }
    this.arr = newArr;

    Max_Heapify(0);
    return max;
}
```

^Extract max makes a new array and copies the values from the old array. Heap extract max normally takes lg(n) time, but I'm realizing that my implementation makes it take O(n)[copies values to new array] + O(lg(n))[max heapify] + c = **O(n)** time.

```
public void Max_Heap_Insert(String name, int key) {
    size++;
    // this means make a new array with +1 size... :')

    // newArr[size-1] = Integer.MIN_VALUE;
    Url[] newArr = new Url[size];
    Url dummy = new Url(name, Integer.MIN_VALUE); // ensures it goes last?
    newArr[size - 1] = dummy;

    for (int i = 0; i < size - 1; i++) {
        newArr[i] = arr[i];
    }
    this.arr = newArr;

    Heap_Increase_Key(size - 1, key); // sets whatever is at size - 1 to have tha

}
```

^The same applies for Max Heap Insert. It normally takes lg(n) time, but since I copied it from an old array to a new array, it now takes O(n) + lg(n) + c = **O(n)** time.

The runtime of the rest of the heap methods are given to us by the textbook.
Searchheap class is just UrlHeap's code but modified.

SortTracker class

Assume n = # of searches

```
public SearchFrequency[] getSearchFrequencies() {
    //ties every search to a frequency. Hashmaps are sets.
    Map<String, Integer> map = new HashMap<>();
    for(String x : searches) {
        map.put(x,Collections.frequency(searches, x));
    }

    //SearchFrequency objects have the search and their frequency
    ArrayList<SearchFrequency> searches = new ArrayList<>();
    for(String x : map.keySet()) {
        SearchFrequency search = new SearchFrequency(x, map.get(x));
        searches.add(search);
    }

    //im doing it this way because I don't know how to add from keySet to array
    //adds all items from ArrayList to Array
    SearchFrequency[] searchArr = new SearchFrequency[searches.size()];
    for(int x = 0; x < searches.size(); x++) {
        searchArr[x] = searches.get(x);
    }
    SearchHeap searchHeap = new SearchHeap(searchArr);
    searchHeap.HeapSort();
    return searchHeap.getArray();

}
```

First, let's look at the first for loop. This is the source code for Java.Collections's Frequency() method.

```
public static int frequency(Collection<?> c, Object o) {
    int result = 0;
    if (o == null) {
        for (Object e : c)
            if (e == null)
                result++;
    } else {
        for (Object e : c)
            if (o.equals(e))
                result++;
    }
    return result;
}
```

The frequency method takes O(n) time.
For all the items in the arraylist of searches, I run a function that takes (O(n)) time, therefore, the first for loop takes O(n^2)) time.

The second for loop converts all the items in the newly built map into SearchFrequency objects, which takes O(n) time.

Third for loop passes all those items into an array so that it can be passed to a heap, O(n) time.

The total time of this method is O(n) + O(n) + O(n^2) + c. Which means the runtime is **O(n^2).** That is terrible.

Program class

```
public void initSearchAndHeap() {

    System.out.println("Enter your search term.");

    String term = scan.nextLine();
    searches.add(term);
    Spider spider = new Spider();
    String[] sites = spider.search("https://en.wikipedia.org/wiki/List_of_most_popular_websites", term);

    // takes all the sites successfully searched by spider and stores it in urls
    urls = new Url[sites.length];

    // take the urls and build class Url
    for (int i = 0; i < urls.length; i++) {

        urls[i] = new Url(sites[i]);
        // raw pagerank
        urls[i].setRankBeforeSort(i);
    }
    heap = new UrlHeap(urls);
    heap.Build_Max_Heap();
    this.heapSortDisplay();
    System.out.println();
    // bmh

}
```

^ We've already calculated that the search() function from spider takes O(n). The for loop takes 2n which is O(n). time is 2n + n + c = **O(n)**

```java
public void qDisplay() {

    UrlHeap heap2 = new UrlHeap(heap.getArray().clone());

    PriorityQueue<Url> queue = new PriorityQueue<Url>();
    int size = heap2.getArray().length;

    if (size <= 10) {
        for (int i = 0; i < size; i++) {
            // h2.bmh
            Url extracted = heap2.Heap_Extract_Max_Url();
            queue.add(extracted);
        }
    } else if (size > 10) {
        for (int i = 0; i < 10; i++) {
            // h2.bmh
            Url extracted = heap2.Heap_Extract_Max_Url();
            queue.add(extracted);
        }
    }
    size = queue.size();
    if (size <= 10) {
        for (int i = 0; i < size; i++) {
            int rank = i + 1;
            System.out.println("Google PageRank: " + rank);
            queue.remove().printValues();
            System.out.println();

        }
    } else if (size > 10) {
        for (int i = 0; i < 10; i++) {
            int rank = i + 1;
            System.out.println("Google PageRank: " + rank);
            queue.remove().printValues();
            System.out.println();
        }
    }
```

^ Only two for loops will be run. They both perform no more than 10 actions. They do not loop through the entire list. In the worst case, heap extract max is run 10 times. That's 10*n. It is 10*n and not 10*lg(n) because heap extract max makes a new array without extracted node in my implementation. Therefore, the runtime of this is **O(n).**

```java
 */
public void heapSortDisplay() {
    System.out.println("calling heapSortDisplay()");
    // we use heap2 because the heapsort messes up the size variable in UrlHeaps...
    UrlHeap heap2 = new UrlHeap(heap.getArray().clone());
    heap2.HeapSort();
    Url[] sorted = heap2.getArray();
    int x = 1;
    if (sorted.length >= 10) {
        for (int i = sorted.length - 1; i > sorted.length - 11; i--) {
            System.out.println("Google PageRank: " + x);
            x++;

            int beforeSorted = sorted[i].getRankBeforeSort() + 1; // arrays start at 0.
            System.out.println("PageRank BEFORE being sorted: " + beforeSorted);

            heap2.printValue(i);
            System.out.println();

        }
    }

    else if (sorted.length < 10) {
        for (int i = sorted.length - 1; i >= 0; i--) {
            System.out.println("Google PageRank: " + x);
            x++;
            int beforeSorted = sorted[i].getRankBeforeSort() + 1; // arrays start at 0.
            System.out.println("PageRank BEFORE being sorted: " + beforeSorted);
            heap2.printValue(i);
            System.out.println();
        }
    }
}
```

^ I think cloning takes (O(n)) time. Thankfully, heapsort takes (O(n(lg(n))) time, so I didn't make anything worse. The for loop only runs 10 times, so that's constant. The total time is nlgn + n + c, which is **O(nlg(n))**

```
public void displayTop() {
    SortTracker tracker = new SortTracker(searches);
    SearchHeap searchHeap = new SearchHeap(tracker.getSearchFrequencies());
    searchHeap.HeapSort();
    SearchFrequency[] uniques = searchHeap.getArray();

    if (uniques.length <= 10) {
        for (int i = uniques.length - 1; i >= 0; i--) {
            System.out.println("Term: " + uniques[i].getSearch());
            System.out.println("Frequency: " + uniques[i].getFrequency());
            System.out.println();
        }
    } else if (uniques.length > 10) {
        for (int i = uniques.length - 1; i > uniques.length - 11; i--) {
            System.out.println("Term: " + uniques[i].getSearch());
            System.out.println("Frequency: " + uniques[i].getFrequency());
            System.out.println();
        }
    }

}
```

^ Heapsort is nlgn. Tracker.getsearchfrequencies() was already calculated to be O(n)^2. The for loop runs no more than 10 times. nlgn + n^2 + c = **O(n^2)**.

```
public void pay() {
    // h.bmh
    this.heapSortDisplay(); // for convenience of tester
    // h.bmh

    for (int i = 0; i < heap.getArray().length; i++) {
        // h.bmh
        // id = index
        System.out.println("ID: " + i);
        heap.printValue(i);
        System.out.println();
    }
    System.out.println("Enter the ID of the customer who paid. (INTEGER)");
    int ID = scan.nextInt();
    // their new score will be old score + amount paid
    int score = heap.getScore(ID);
    System.out.println("The customer is " + heap.getOneName(ID));
    System.out.println("Enter the amount this customer paid, rounded to the nearest INTEGER");
    int paid = scan.nextInt();
    int newScore = score + paid;
    heap.Heap_Increase_Key(ID, newScore);
    System.out.println("Customer's score has been increased by " + paid);

}
```

^heapsortDisplay was already calculated to be nlgn. It is not needed. Heap increase key is lg(n). The for loop is O(n). nlgn + lgn + n = **O(nlgn)**. I could make it O(n) by deleting the unnecessary method.

I will only be posting the important parts of the next method

```
 */
public void customEdit() {
    // h.bmh
    // now print it out for the console viewer
    this.qDisplay(); // for convenience of tester.

    for (int i = 0; i < heap.getArray().length; i++) {
        // h.bmh
        System.out.println("ID: " + i);
        heap.printValue(i);
        System.out.println();
    }
```

^ qDisplay is O(n). At the end, it builds a max heap, which takes O(n). 2n + c = **O(n)**

```
public void editScore() {
    // h.bmh
    qDisplay();

    for (int i = 0; i < heap.getArray().length; i++) {
        // h.bmh
        System.out.println("ID: " + i);
        heap.printValue(i);
        System.out.println();
    }
    System.out.println("Enter the ID of the customer who you want to edit. (INTEGER)");
    int ID = scan.nextInt();
    Url recieved = heap.getUrl(ID);
    System.out.println("Customer recieved: " + recieved.getName());
    System.out.println("Score of customer: " + recieved.getScore());
    System.out.println("Enter their new score");
    int newScore = scan.nextInt();
    recieved.setScore(newScore);
    heap.Build_Max_Heap(); // re heapifies. Can't use Increase key. Increase key only applies to bigger
                           // values.
    System.out.println("Their new score is " + recieved.getScore());
}
}
```

^ again, qdisplay() is O(n). The for loop is O(n), and build max heap isis O(n). 3n + c = **O(n)**.

```
private void addCustomSite() {
    // h.bmh
    System.out.println("Enter the URL of the new website. (STRING)");
    String url = scan.nextLine();
    System.out.println("Enter the amount of money they paid to the nearest (+INTEGER)");
    int paid = scan.nextInt();
    System.out.println("Enter the number of years old they are (+INTEGER)");
    int date = scan.nextInt();
    System.out.println("Enter the number of keywords this website has related to your sear
    int keywords = scan.nextInt();
    System.out.println("Enter the number of links this website has to it (+INTEGER)");
    int links = scan.nextInt();

    Url score = new Url(url, paid, date, keywords, links);
    heap.Max_Heap_Insert(url, score.getScore());

}
```

^ max heap insert takes O(n) time instead of lg(n) because of my implementation. Therefore, this method is **O(n)**

Same reasoning applies for addsite() method.

```
private void banTop() {
    System.out.println("removed a website with progress " + heap.Heap_Extract_Max());

}
```

Only uses heap extract max, which takes O(n) instead lg(n) time because of my implementation. Therefore, **O(n)**.

PROBLEMS ENCOUNTERED

I wanted to provide an array of genuinely successful URLs to the user if I could, but that would mean searching forever. I solved this issue by adding dummy URLs to fill the list if nothing could be found after 100 searches.

Debugging is a process that makes me feel like the worst and best programmer in the world. For some reason, the Java debugger did not work if the webcrawler ran, and I would get an index OOB error if I did something that looked like this in a very weird order, like in the image below.

```
heap.Heap_Extract_Max();
heap.HeapSort();
heap.Build_Max_Heap();
heap.printValues();
heap.Heap_Extract_Max();
heap.HeapSort();
System.out.println();
heap.printValues();
```

Each individual function worked, but together, they gave me the error. I think I tried testing each individual function with the debugger, and eventually I found out that adding originsize = size in heapsort was causing the bug. I kept track of the original size because heapsort messed with the size of the heap and I wanted to restore the size when I was done so I could re-call build_max_heap() and re use the same array. One solution caused another problem, so to work around this, from now on, whenever I need to call heapsort, I make a copy of the array and pass it into a new heap. That's why in my Prorgram class, you will see me create a heap2 from a cloned array of heap1.

I needed a way to keep track of the score assigned to a URL. At first I thought about using a map, but I was too lazy to learn how to. Thankfully, it turns out that using a map is a bad idea because you can't edit the individual values that go into calculating the score. I ended up making a class called Url, which held the name and all the score values.

Recording the searches was the easy part, finding the frequencies of the searches was harder. At first, I wanted to get the frequency by checking all the items for every other item, but then I realized the Collections class had a frequency() method to do that for me. I used that to create objects of a class that held the name of a term and the frequency it was searched, and used a heap designed for that object to sort it. Multiple SearchFrequency objects for one term don't exist because I took advantage of how a hashmap is also a set. On the side, it taught me about the usefulness of classes to hold data.

I needed a gui and did not know how to make a gui, so I used scanner. The first time I tried to implement a command console, it was terrible. The double while loop may look messy, but it's better than my last idea.

I encountered four problems that I could not figure out how to solve. How to extract and insert into an array without having to increase and decrease the size to make it lg(n) instead of O(n) time and how to get the frequencies of a search without comparing every element to every other element for O(n^2) time.

LESSONS LEARNED

While looking at the runtimes of my code, I realized that the way I implemented my max heap extract and max heap insert caused them to be O(n) instead of lg(n). I suppose I need to be more cautious about whether my implementation of the data structure is causing it to take more time than it was originally intended. I also did not realize that the collections.frequency method took O(n) time until I went and looked at it, and that that caused my entire !topsearches function to be O(n)^2, which makes using heapsort a bit useless, so I suppose I also need to check the time of Java functions I use.

On the bright side, the fact that I copied a new array while using !display in heapsortdisplay() did not affect the big O runtime of the methods that used heapsort, since heapsort is nlgn, and copying an array only takes O(n).

A lot of my time was spent debugging. I think I learned some debugging skills. While playing around with the debugger, I learned about the ability to watch a variable, which was helpful for keeping track of a specific array that had its values changed in testing

I gained experience in editing somebody else's program for my own use. For the webcrawler, even though I did not understand how his program worked, by messing around with it, I figured out that the fact that he printed his successes meant that I could also add those successes to an array and use those as my Urls, and that since his searches ran on a while loop of max searches, I could add a statement to break the loop if my array of successes was full.

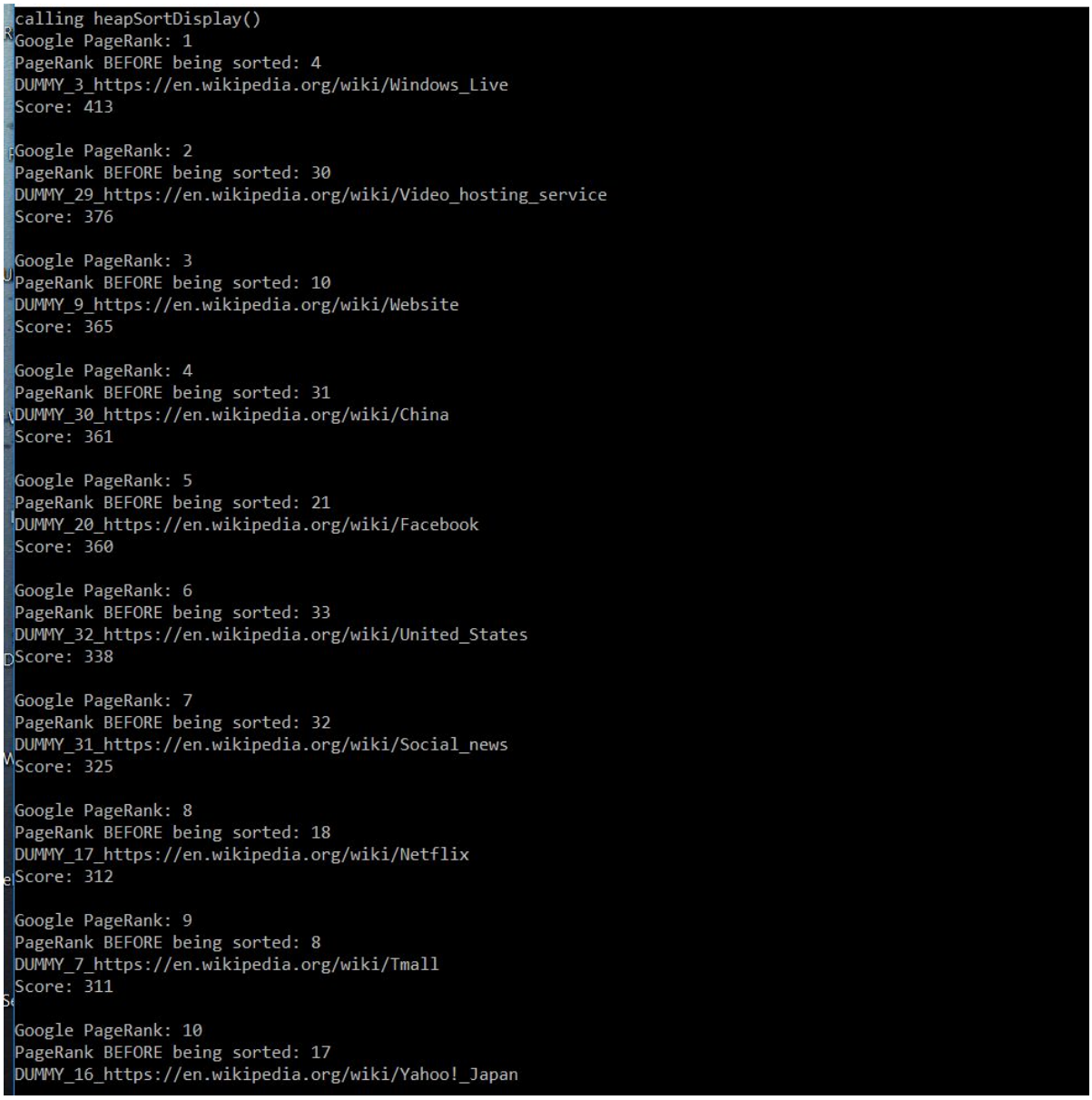
I also gained experience in using a data structure to organize and sort objects based off of their data instead of just integers. I always knew that I could do that if I implemented it correctly, but I have never done it before.

I think a queue implementation is supposed to be faster than printing from heapsort, since heapsort takes nlgn and when you use the queue, you only use 10*lg(n) with heap-extract-max and maybe a O(n) function to make a copy of the array to back it up. You probably don't even have to back up the whole array, just back up the 10 values you popped from it and add it back with the insert function without decreasing / increasing the array's size. That would make the queue implementation lg(n) ideally.

TESTING - FROM MOST IMPORTANT TO LEAST IMPORTANT

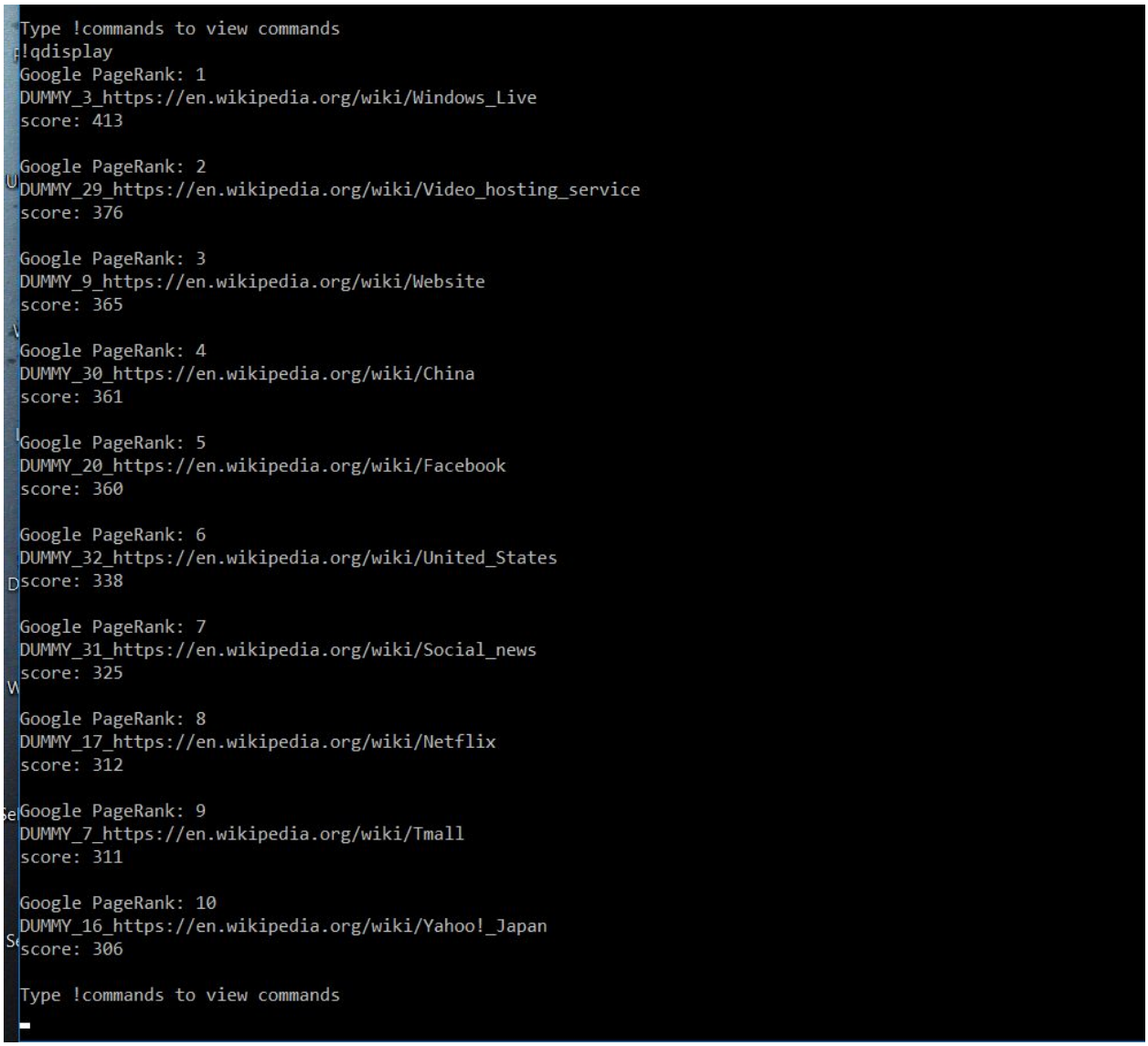
Testing from the console - the actual program

Enter “the cow jumped”

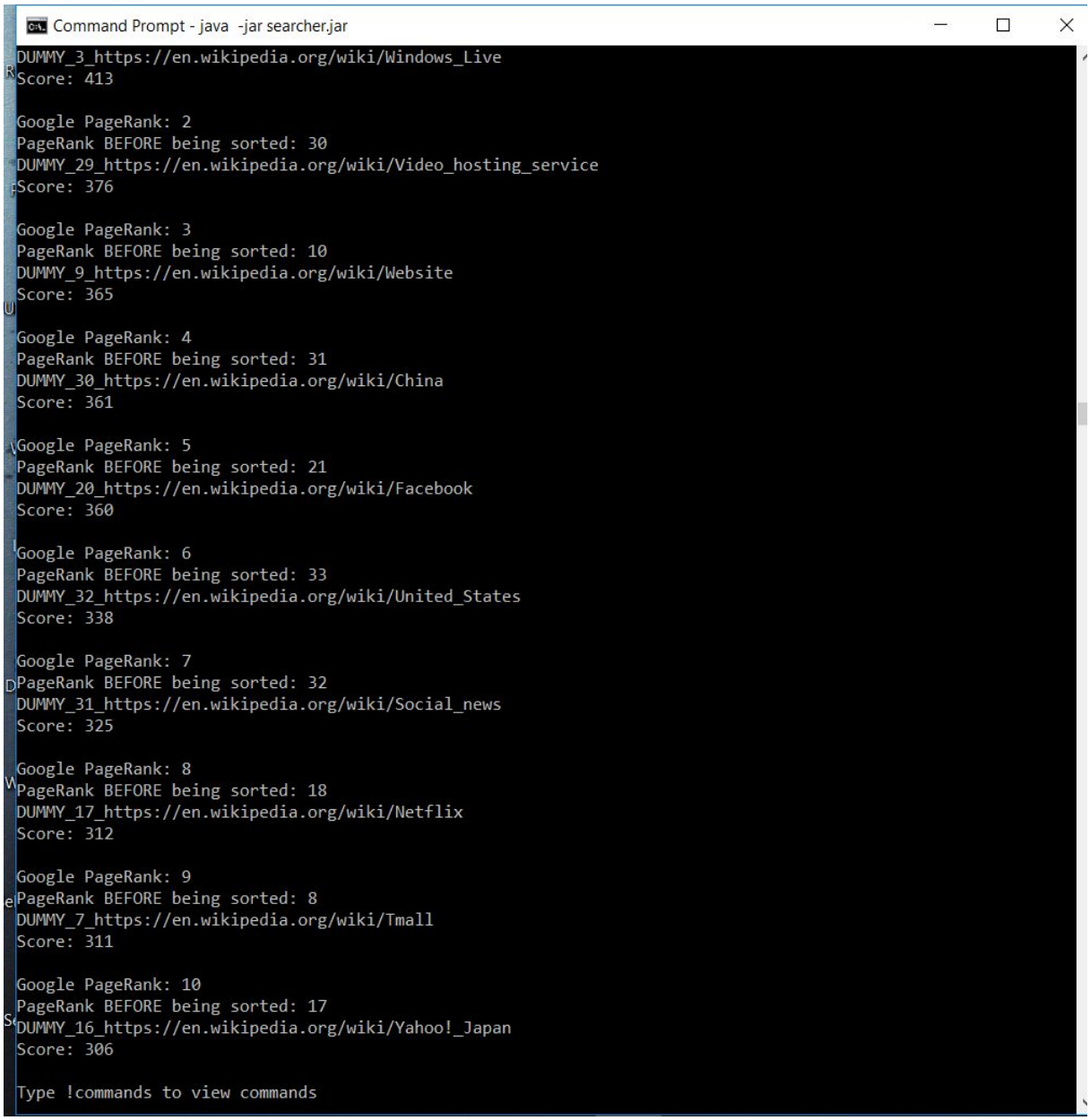


^ vague term = dummy urls

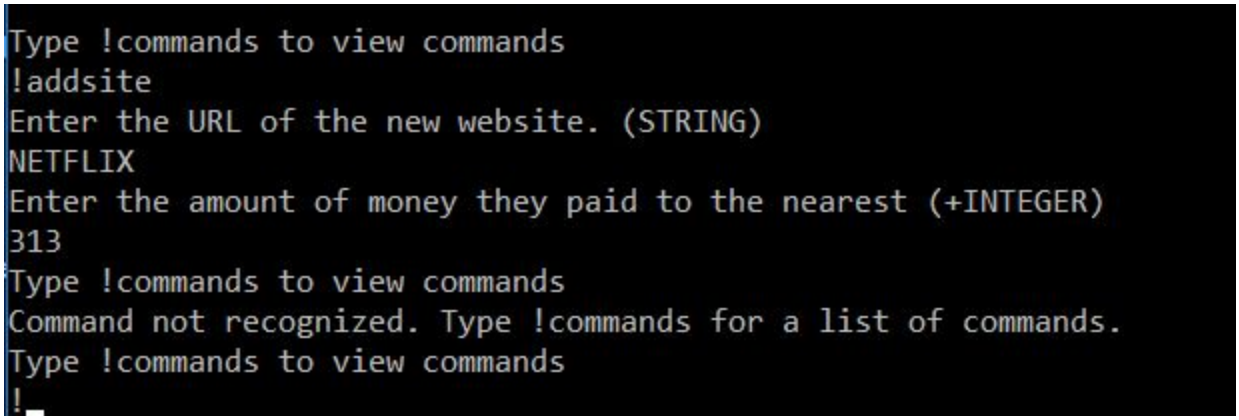
Enter “!qdisplay”



Enter “!display”

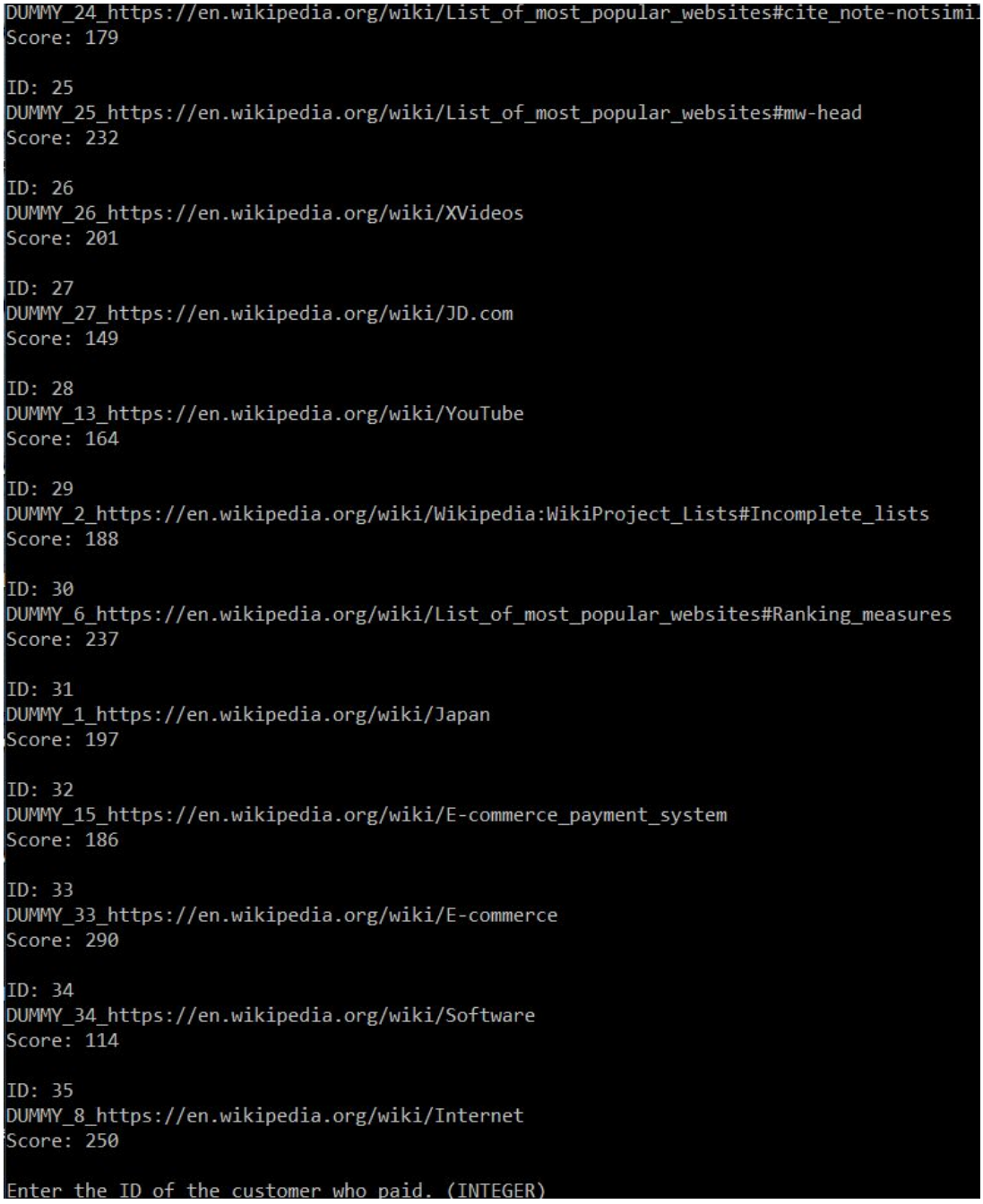


Enter “!addsite”, “NETFLIX”, “313”, “!qdisplay”

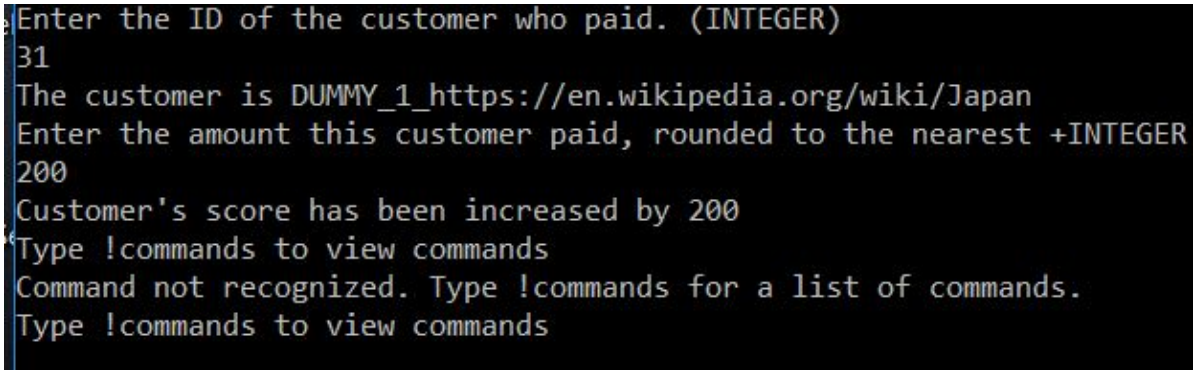

Type !commands to view commands
!addsite
Enter the URL of the new website. (STRING)
NETFLIX
Enter the amount of money they paid to the nearest (+INTEGER)
313
Type !commands to view commands
Command not recognized. Type !commands for a list of commands.
Type !commands to view commands
!

```
!qdisplay
Google PageRank: 1
DUMMY_3_https://en.wikipedia.org/wiki/Windows_Live
score: 413

Google PageRank: 2
DUMMY_29_https://en.wikipedia.org/wiki/Video_hosting_serv
score: 376

Google PageRank: 3
DUMMY_9_https://en.wikipedia.org/wiki/Website
score: 365

Google PageRank: 4
DUMMY_30_https://en.wikipedia.org/wiki/China
score: 361

Google PageRank: 5
DUMMY_20_https://en.wikipedia.org/wiki/Facebook
score: 360

Google PageRank: 6
DUMMY_32_https://en.wikipedia.org/wiki/United_States
score: 338

Google PageRank: 7
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
score: 325

Google PageRank: 8
NETFLIX
score: 313

Google PageRank: 9
DUMMY_17_https://en.wikipedia.org/wiki/Netflix
score: 312

Google PageRank: 10
DUMMY_7_https://en.wikipedia.org/wiki/Tmall
score: 311

Type !commands to view commands
```

^notice in the image above, NETFLIX was added as #8
Enter “!pay” to be prompted with this



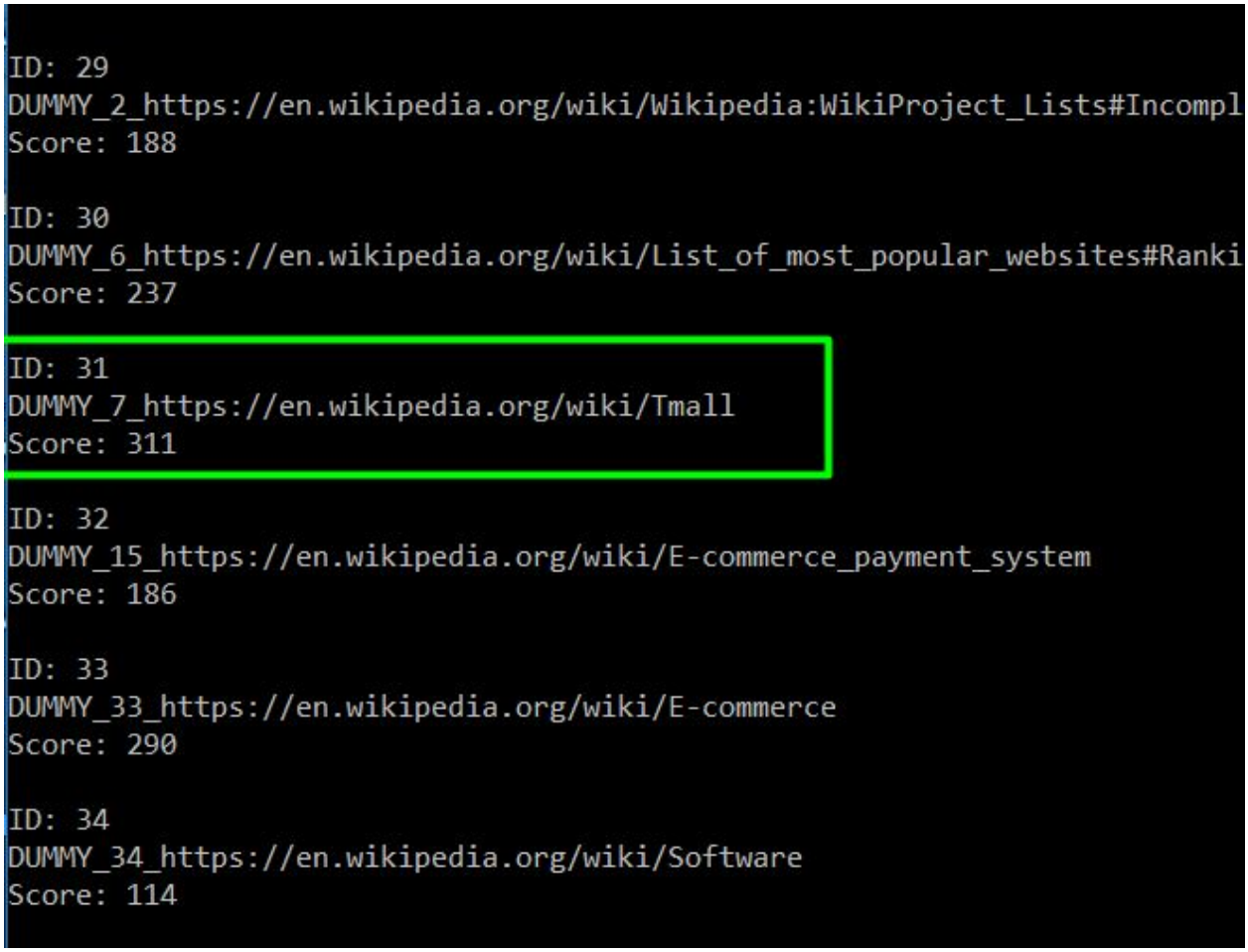
Enter “31” to select japan
Japan’s score is 197. Say they paid 200
Enter “200”
Note: 200 + 197 = 397.



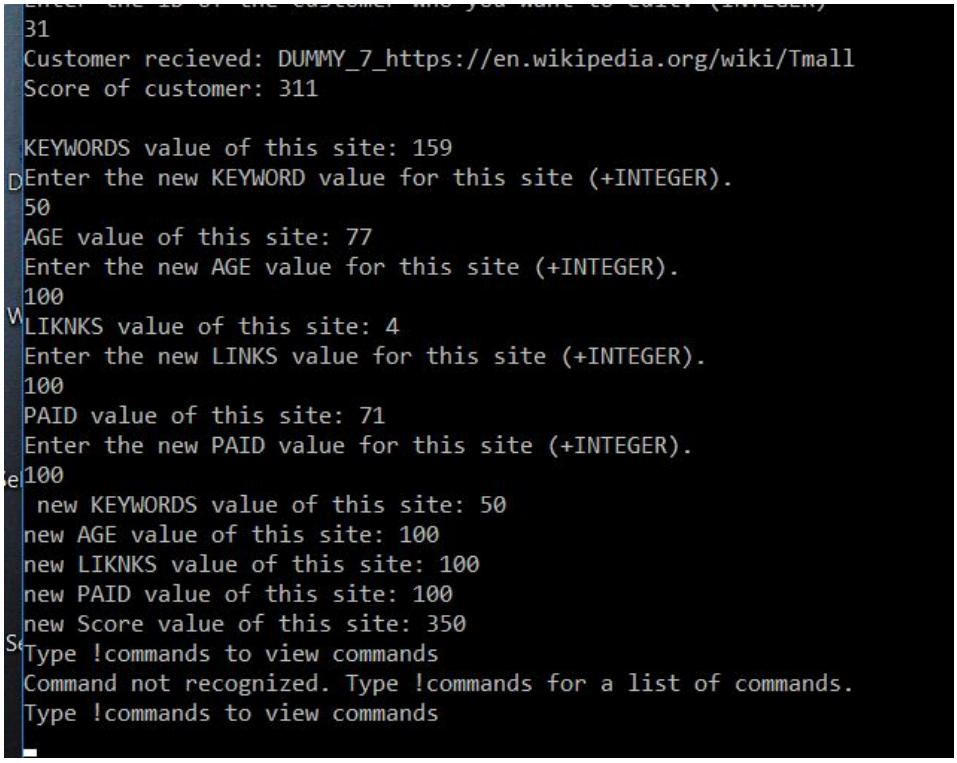
Enter “!qdisplay”

```
Type !commands to view commands
!qdisplay
Google PageRank: 1
DUMMY_3_https://en.wikipedia.org/wiki/Windows_Live
score: 413

Google PageRank: 2
DUMMY_1_https://en.wikipedia.org/wiki/Japan
score: 397

Google PageRank: 3
DUMMY_29_https://en.wikipedia.org/wiki/Video_hosting_service
score: 376

Google PageRank: 4
DUMMY_9_https://en.wikipedia.org/wiki/Website
score: 365

Google PageRank: 5
DUMMY_30_https://en.wikipedia.org/wiki/China
score: 361

Google PageRank: 6
DUMMY_20_https://en.wikipedia.org/wiki/Facebook
score: 360

Google PageRank: 7
DUMMY_32_https://en.wikipedia.org/wiki/United_States
score: 338

Google PageRank: 8
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
score: 325

Google PageRank: 9
NETFLIX
score: 313

Google PageRank: 10
DUMMY_17_https://en.wikipedia.org/wiki/Netflix
score: 312

Type !commands to view commands
```

^notice how japan is #2 at 397
Enter “!customedit”



Enter “31” to select Tmall
Enter “50” “100” “100” “100”



Notice how 159 + 77 + 4 + 71 = 311 (their old score). Means random score generator is working.

Enter “!qdisplay”

```
!qdisplay
Google PageRank: 1
DUMMY_3_https://en.wikipedia.org/wiki/Windows_Live
score: 413

Google PageRank: 2
DUMMY_1_https://en.wikipedia.org/wiki/Japan
score: 397

Google PageRank: 3
DUMMY_29_https://en.wikipedia.org/wiki/Video_hosting_service
score: 376

Google PageRank: 4
DUMMY_9_https://en.wikipedia.org/wiki/Website
score: 365

Google PageRank: 5
DUMMY_30_https://en.wikipedia.org/wiki/China
score: 361

Google PageRank: 6
DUMMY_20_https://en.wikipedia.org/wiki/Facebook
score: 360

Google PageRank: 7
DUMMY_7_https://en.wikipedia.org/wiki/Tmall
score: 350

Google PageRank: 8
DUMMY_32_https://en.wikipedia.org/wiki/United_States
score: 338

Google PageRank: 9
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
score: 325

Google PageRank: 10
NETFLIX
score: 313
```

^notice how tmall is now #7 with a score of 350
Enter “!editscore”

```
ID: 27
DUMMY_27_https://en.wikipedia.org/wiki/JD.com
Score: 149

ID: 28
DUMMY_13_https://en.wikipedia.org/wiki/YouTube
Score: 164

ID: 29
DUMMY_2_https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists
Score: 188

ID: 30
DUMMY_6_https://en.wikipedia.org/wiki/List_of_most_popular_websites#Ranking_measures
Score: 237

ID: 31
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
Score: 325
```

Enter “28” to select youtube
Enter “349” as their new score.

```
Enter the ID of the customer who you want to edit. (INTEGER)
28
Customer recieved: DUMMY_13_https://en.wikipedia.org/wiki/YouTube
Score of customer: 164
Enter their new score
349
Their new score is 349
```

Enter “!qdisplay”

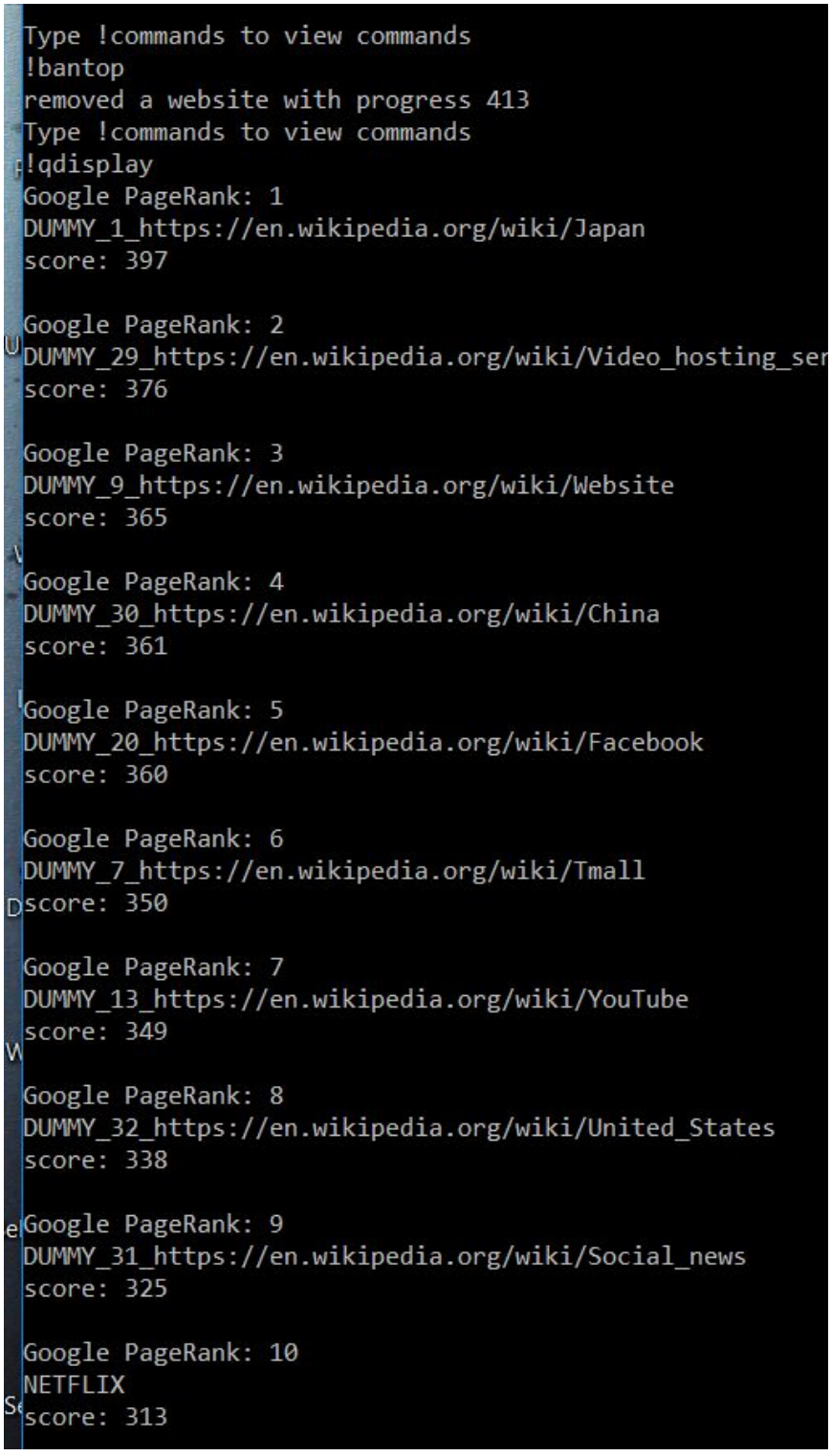
```
!qdisplay
Google PageRank: 1
DUMMY_3_https://en.wikipedia.org/wiki/Windows_Live
score: 413

Google PageRank: 2
DUMMY_1_https://en.wikipedia.org/wiki/Japan
score: 397

Google PageRank: 3
DUMMY_29_https://en.wikipedia.org/wiki/Video_hosting_service
score: 376

Google PageRank: 4
DUMMY_9_https://en.wikipedia.org/wiki/Website
score: 365

Google PageRank: 5
DUMMY_30_https://en.wikipedia.org/wiki/China
score: 361

Google PageRank: 6
DUMMY_20_https://en.wikipedia.org/wiki/Facebook
score: 360

Google PageRank: 7
DUMMY_7_https://en.wikipedia.org/wiki/Tmall
score: 350

Google PageRank: 8
DUMMY_13_https://en.wikipedia.org/wiki/YouTube
score: 349

Google PageRank: 9
DUMMY_32_https://en.wikipedia.org/wiki/United_States
score: 338

Google PageRank: 10
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
score: 325
```

^notice how youtube is #8 with a score of 349

Enter “!bantop” to remove top website

```
Type !commands to view commands
!bantop
removed a website with progress 413
Type !commands to view commands
```

Enter “!qdisplay”



Enter “!highscore” to view top site

```
Type !commands to view commands
!highscore
Top website is: DUMMY_1_https://en.wikipedia.org/wiki/Japan
Score: 397
Type !commands to view commands
```

Enter “!search”. Search for another or the same term
Enter “!search”. Search for another or the same term
Enter “!search”. Search for another or the same term.
Repeat as many times as you want. Even 10+ times of unique searches.
Enter “!topsearches”

```
Type !commands to view commands
!topsearches
Term: moo moo
Frequency: 3

Term: beep
Frequency: 2

Term: the cow jumped
Frequency: 1

Term: the
Frequency: 1
```

!display, !displayraw, and !addcustomsite are optional and not recommended

Original tests written earlier in the week - individual heap function tests

Sometimes things seem fine but later down the line I discover a bug that I end up fixing. All these test files were gathered from old test files I had laying around from throughout the week.

Heapsort

```
1  package heapsort_tester;
2
3  public class Test_Heapsort {
4
5      public static void main(String[] args) {
6          int[] arr = new int[] {16,4,10,14,7,9,3,2,8,11,1,1,1};
7          HeapArray heap = new HeapArray(arr);
8
9          heap.HeapSort(arr);
10
11         for (int x : arr) {
12             System.out.println(x);
13         }
14     }
15 }
16
```

Problems Javadoc Declaration Search Console

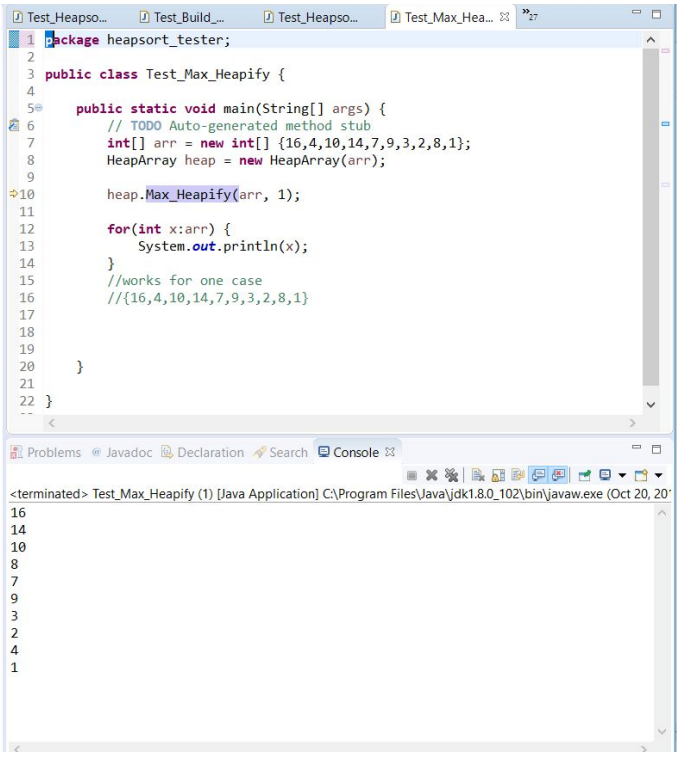
```
<terminated> Test_Heapsort (2) [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw
1
1
1
2
3
4
7
8
9
10
11
14
16
```

```
1  package heapsort_tester;
2
3  public class Test_Heapsort {
4
5      public static void main(String[] args) {
6          int[] arr = new int[] {16,4,10,14,7,9,3,2,8,1};
7          HeapArray heap = new HeapArray(arr);
8
9          heap.HeapSort(arr);
10
11         for (int x : arr) {
12             System.out.println(x);
13         }
14     }
15 }
16
```

Problems Javadoc Declaration Search Console

```
<terminated> Test_Heapsort (1) [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe (Oct 20, 2018,
1
2
3
4
7
8
9
10
14
16
```

Max heapify (I think I got array from textbook)



Build max heap

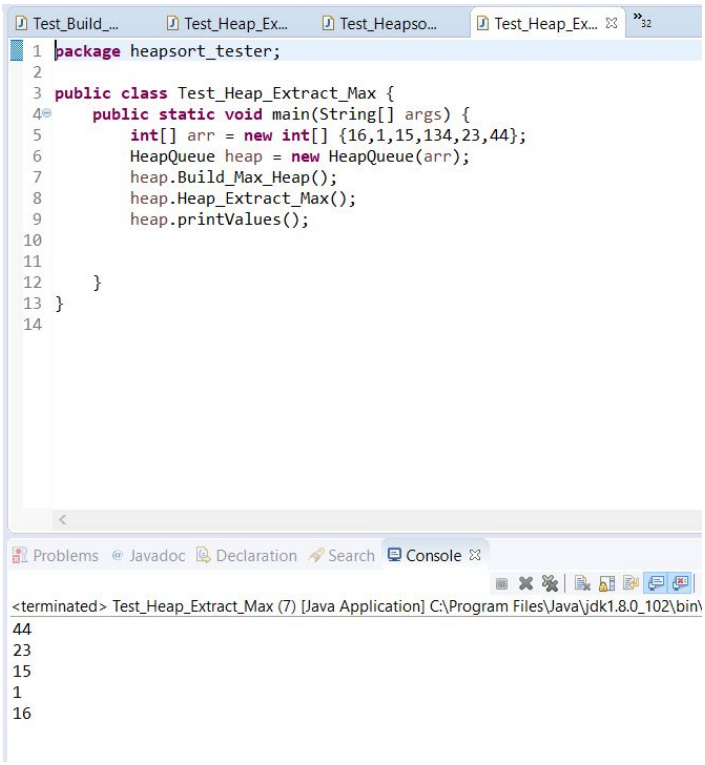
```java
package heapsort_tester;

public class Test_Build_Max_Heap {
    public static void main(String[] args) {
        int[] arr = new int[] {4,1,3,2,16,9,10,14,8,7};
        HeapArray heap = new HeapArray(arr);
        heap.Build_Max_Heap(arr);

        for(int x:arr) {
            System.out.println(x);
        }
    }
}
//16,14,10,8,7,9,3,2,4,1
//ok
```

Problems Javadoc Declaration Search Console

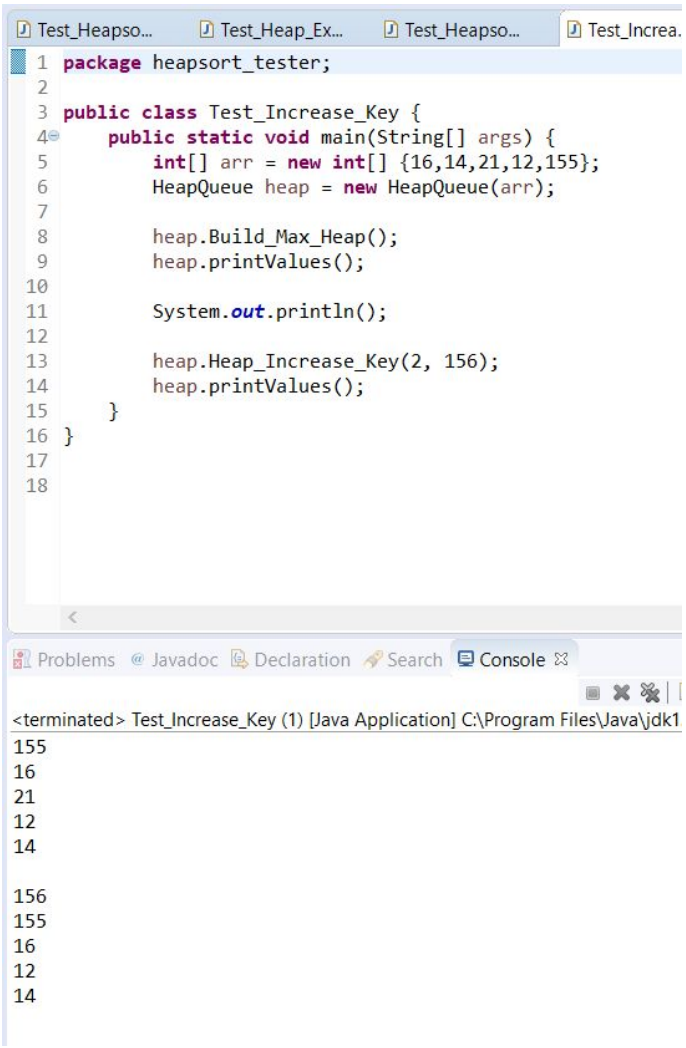
<terminated> Test_Build_Max_Heap (1) [Java Application] C:\Program Files\Java\jdk1.8

```
16
14
10
8
7
9
3
2
4
1
```

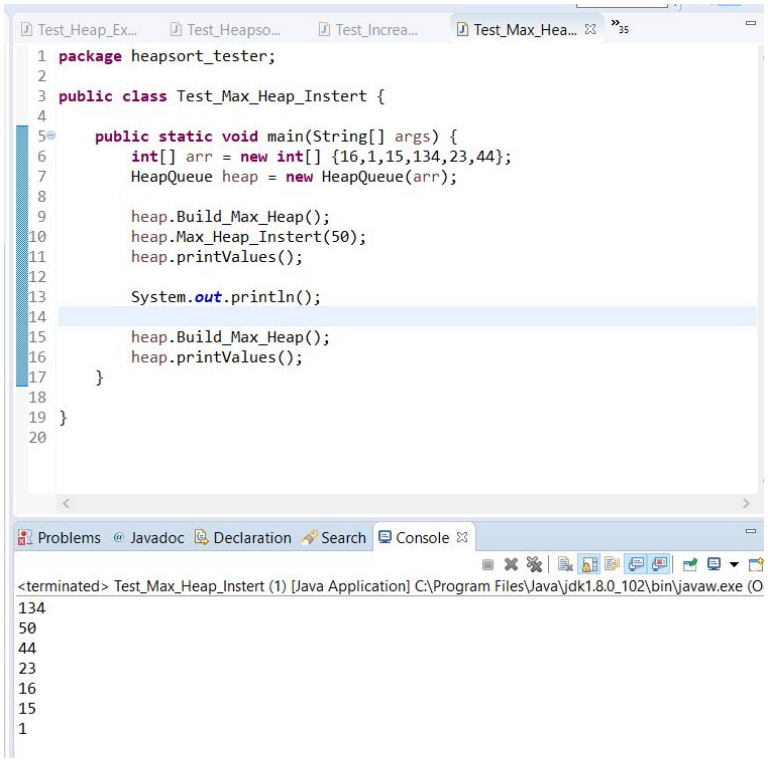
Heap extract max


Test_Build_...
Test_Heap_Ex...
Test_Heapso...
Test_Heap_Ex...
package heapsort_tester;
public class Test_Heap_Extract_Max {
public static void main(String[] args) {
int[] arr = new int[] {16,1,15,134,23,44};
HeapQueue heap = new HeapQueue(arr);
heap.Build_Max_Heap();
heap.Heap_Extract_Max();
heap.printValues();
}
}
Problems
Javadoc
Declaration
Search
Console
<terminated> Test_Heap_Extract_Max (7) [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\
44
23
15
1
16

Heap Increase Key



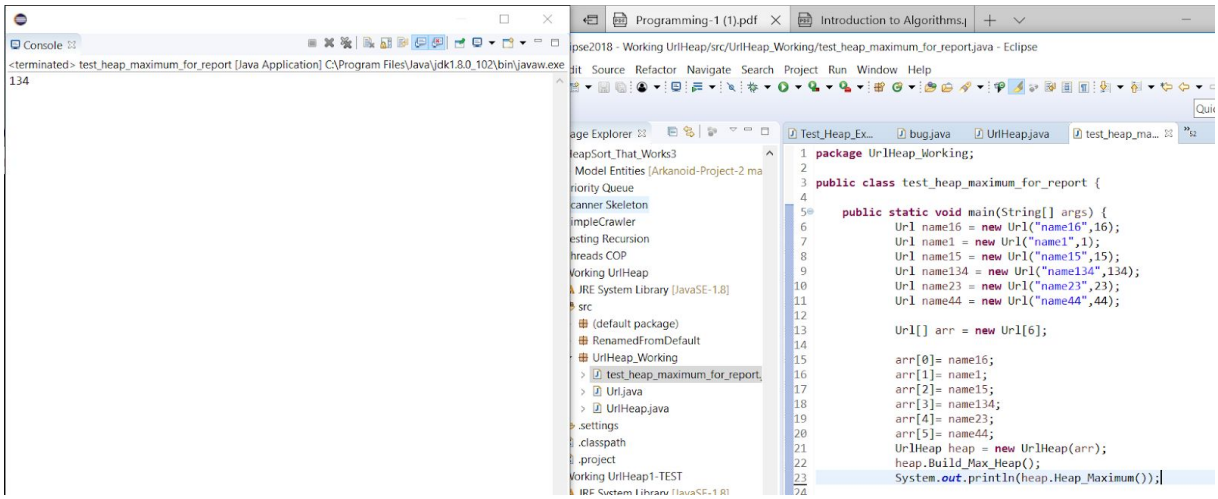
Max Heap insert



```
package heapsort_tester;

public class Test_Max_Heap_Instert {

    public static void main(String[] args) {
        int[] arr = new int[] {16,1,15,134,23,44};
        HeapQueue heap = new HeapQueue(arr);

        heap.Build_Max_Heap();
        heap.Max_Heap_Instert(50);
        heap.printValues();

        System.out.println();

        heap.Build_Max_Heap();
        heap.printValues();
    }

}
```

```
<terminated> Test_Max_Heap_Instert (1) [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe (O
134
50
44
23
16
15
1
```

Heap maximum() is impossible to mess up...



```
<terminated> test_heap_maximum_for_report [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe
134
```

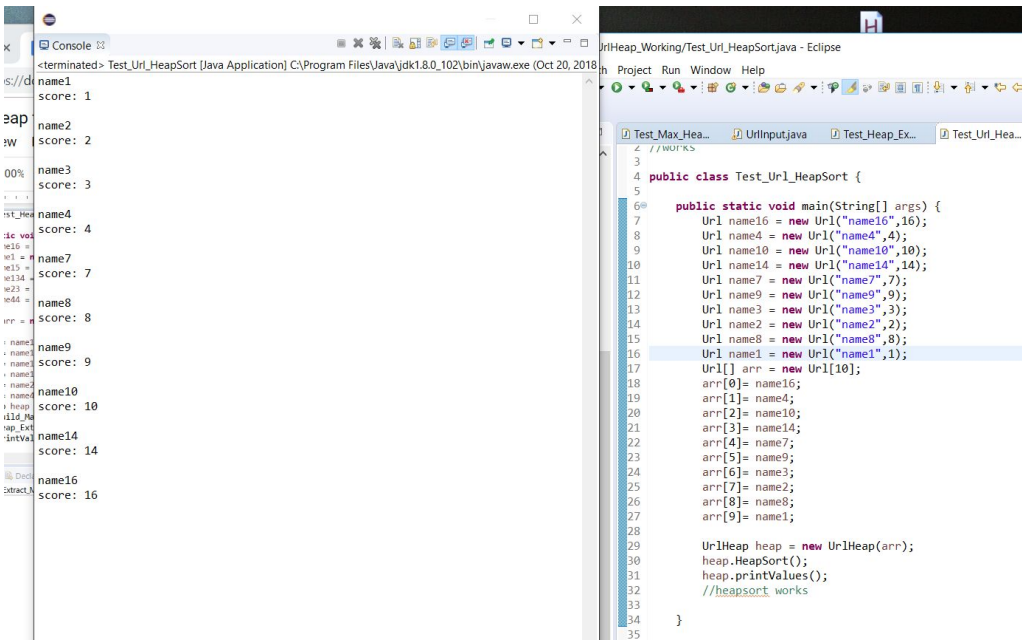
```
package UrlHeap_Working;

public class test_heap_maximum_for_report {

    public static void main(String[] args) {
            Url name16 = new Url("name16",16);
            Url name1 = new Url("name1",1);
            Url name15 = new Url("name15",15);
            Url name134 = new Url("name134",134);
            Url name23 = new Url("name23",23);
            Url name44 = new Url("name44",44);

            Url[] arr = new Url[6];

            arr[0]= name16;
            arr[1]= name1;
            arr[2]= name15;
            arr[3]= name134;
            arr[4]= name23;
            arr[5]= name44;
            UrlHeap heap = new UrlHeap(arr);
            heap.Build_Max_Heap();
            System.out.println(heap.Heap_Maximum());
```

```java
public class Test_Heap_Extract_Max {

    public static void main(String[] args) {
        Url name16 = new Url("name16",16);
        Url name1 = new Url("name1",1);
        Url name15 = new Url("name15",15);
        Url name134 = new Url("name134",134);
        Url name23 = new Url("name23",23);
        Url name44 = new Url("name44",44);

        Url[] arr = new Url[6];

        arr[0]= name16;
        arr[1]= name1;
        arr[2]= name15;
        arr[3]= name134;
        arr[4]= name23;
        arr[5]= name44;
        UrlHeap heap = new UrlHeap(arr);
        heap.Build_Max_Heap();
        heap.Heap_Extract_Max();
        heap.printValues();

```

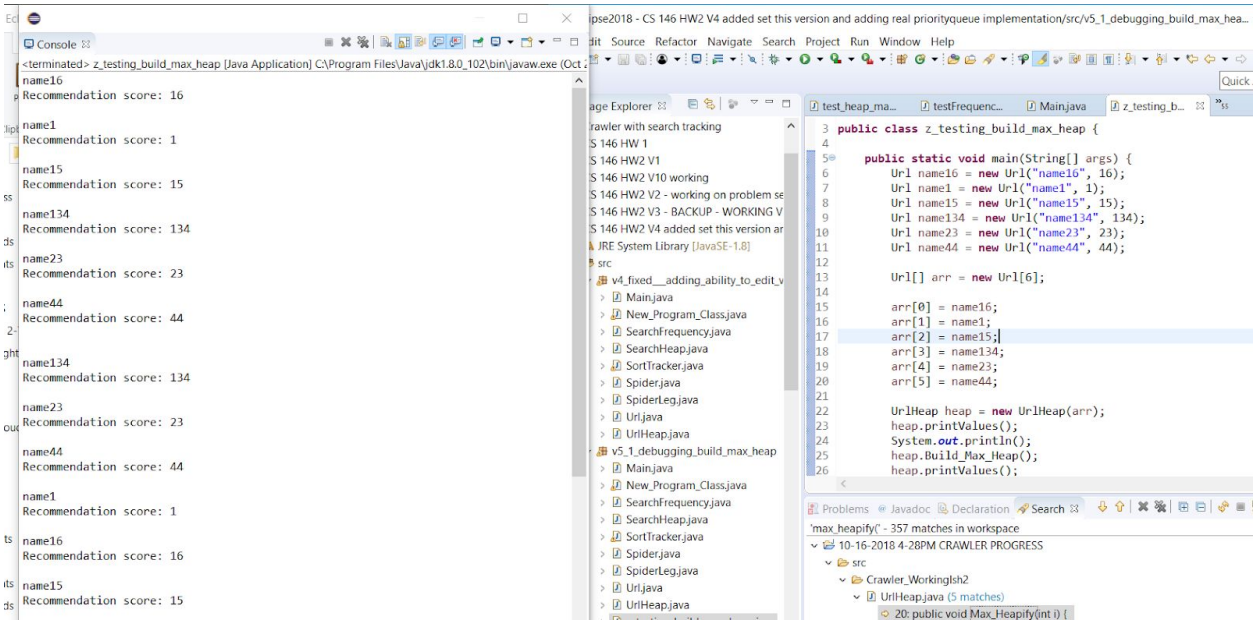
Problems Javadoc Declaration Console Search

```
<terminated> Test_Heap_Extract_Max (3) [Java Application] C:\Program Files\Java\jdk1.8.0_1(
name44
score: 44

name23
score: 23

name15
score: 15

name1
score: 1

name16
score: 16
```

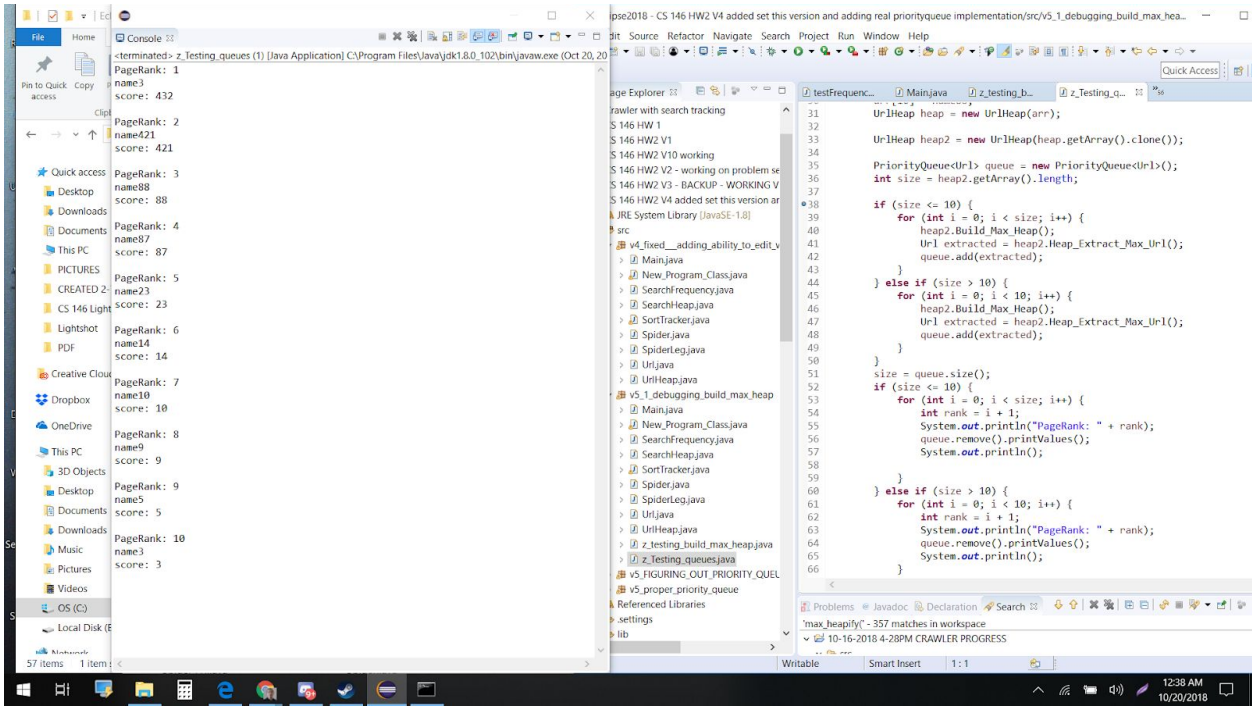
Heapsort with urls



Half of what I think is a build max heap test with urls



Queue implementation sort test



More old tests - !topsearches

You can search again with !search

After several searches, do !topsearches to list your top 10

Most frequent search terms and their frequency

```
!topsearches
Term: the
Frequency: 6

Term: moo moo
Frequency: 2

Term: the cow jumped over the moon
Frequency: 1

Type !commands to view commands
```

^ in the example above, I searched for “the” 6 times, “moo moo” 2 times, “the cow jumped over the moon” 1 time.

```
!topsearches
Term: the
Frequency: 6

Term: moo moo
Frequency: 2

Term: 354
Frequency: 1

Term: gioerlnkf
Frequency: 1

Term: ginlkrefds
Frequency: 1

Term: gubrkjefd
Frequency: 1

Term: the cow jumped over the moon
Frequency: 1

Term: 423543r
Frequency: 1

Term: rewilkds
Frequency: 1

Term: 00
Frequency: 1

Type !commands to view commands
```

Search for “pie” 4 times

Then do !topsearches again

```
!topsearches
Term: the
Frequency: 6

Term: pie
Frequency: 4

Term: moo moo
Frequency: 2

Term: 354
Frequency: 1

Term: gioerlnkf
Frequency: 1

Term: ginlkrefds
Frequency: 1

Term: gubrkjefd
Frequency: 1

Term: 423543r
Frequency: 1

Term: the cow jumped over the moon
Frequency: 1

Term: 00
Frequency: 1

Type !commands to view commands
```

Only displays 10

Testing my ability to record the order searches were added to the list of searches

Testing raw page rank for 5 cases

Testing if the order added to webcrawler = order recorded as raw pagerank

```
Enter your search term.
the
search # 0/100
successes: 0/5

**Visiting** Received web page at https://en.wikipedia.org/wiki/List_of_most_popular_websites
Found (413) links
Searching for the word the...
**Success** Word the found at https://en.wikipedia.org/wiki/List_of_most_popular_websites
search # 1/100
successes: 1/5

**Visiting** Received web page at https://en.wikipedia.org/wiki/Wikipedia:Protection_policy#semi
Found (679) links
Searching for the word the...
**Success** Word the found at https://en.wikipedia.org/wiki/Wikipedia:Protection_policy#semi
search # 2/100
successes: 2/5

**Visiting** Received web page at https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
Found (413) links
Searching for the word the...
**Success** Word the found at https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
search # 3/100
successes: 3/5

**Visiting** Received web page at https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
Found (413) links
Searching for the word the...
**Success** Word the found at https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
search # 4/100
successes: 4/5

**Visiting** Received web page at https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists
Found (1181) links
Searching for the word the...
**Success** Word the found at https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists
```

Order by order found

1) https://en.wikipedia.org/wiki/List_of_most_popular_websites
2) https://en.wikipedia.org/wiki/Wikipedia:Protection_policy#semi
3) https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
4) en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
5) en.wikipedia.org/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists

```
calling heapSortDisplay()
Google PageRank: 1
PageRank BEFORE being sorted: 4
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
Score: 397

Google PageRank: 2
PageRank BEFORE being sorted: 3
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
Score: 238

Google PageRank: 3
PageRank BEFORE being sorted: 1
https://en.wikipedia.org/wiki/List_of_most_popular_websites
Score: 234

Google PageRank: 4
PageRank BEFORE being sorted: 5
https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists
Score: 163

Google PageRank: 5
PageRank BEFORE being sorted: 2
https://en.wikipedia.org/wiki/Wikipedia:Protection_policy#semi
Score: 117


Type !commands to view commands
```

The “pagerank before being sorted” order

1) https://en.wikipedia.org/wiki/List_of_most_popular_websites
2) https://en.wikipedia.org/wiki/Wikipedia:Protection_policy#semi
3) https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
4) https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
5) https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists

=

6) https://en.wikipedia.org/wiki/List_of_most_popular_websites
7) https://en.wikipedia.org/wiki/Wikipedia:Protection_policy#semi
8) https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
9) en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
10) en.wikipedia.org/wiki/Wikipedia:WikiProject_Lists#Incomplete_lists

Random unimportant tests in eclipse for !commands. Scroll to console tests for better tests.

```
!qdisplay
Google PageRank: 1
https://en.wikipedia.org/w/index.php?title=List_of_most_popular_websites&action=edit
score: 384

Google PageRank: 2
https://en.wikipedia.org/wiki/Website
score: 378

Google PageRank: 3
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
score: 357

Google PageRank: 4
https://en.wikipedia.org/wiki/List_of_most_popular_websites#cite_note-Alexa-3
score: 344

Google PageRank: 5
https://en.wikipedia.org/wiki/Baidu
score: 343

Google PageRank: 6
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
score: 315

Google PageRank: 7
https://en.wikipedia.org/wiki/Yahoo!
score: 315

Google PageRank: 8
https://en.wikipedia.org/wiki/Online_encyclopedia
score: 309

Google PageRank: 9
https://en.wikipedia.org/wiki/Reddit
score: 304

Google PageRank: 10
https://en.wikipedia.org/wiki/Social_news
score: 297

Type !commands to view commands
```

!addsite
Adding “NETFLIX OI”

```
Type !commands to view commands
!addsite
Enter the URL of the new website. (STRING)
NETFLIX OI
Enter the amount of money they paid to the nearest (INTEGER)
345
Type !commands to view commands
Command not recognized. Type !commands for a list of commands.
Type !commands to view commands
!qdisplay
Google PageRank: 1
https://en.wikipedia.org/w/index.php?title=List_of_most_popular_websites&action=edit
score: 384

Google PageRank: 2
https://en.wikipedia.org/wiki/Website
score: 378

Google PageRank: 3
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
score: 357

Google PageRank: 4
NETFLIX OI
score: 345

Google PageRank: 5
https://en.wikipedia.org/wiki/List_of_most_popular_websites#cite_note-Alexa-3
score: 344

Google PageRank: 6
https://en.wikipedia.org/wiki/Baidu
score: 343

Google PageRank: 7
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
score: 315

Google PageRank: 8
```

!pay

```
score: 384

ID: 1
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
Score: 357

ID: 2
https://en.wikipedia.org/wiki/Website
Score: 378

ID: 3
https://en.wikipedia.org/wiki/Yahoo!
Score: 315

ID: 4
NETFLIX OI
Score: 345

ID: 5
https://en.wikipedia.org/wiki/Baidu
Score: 343

ID: 6
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
Score: 315
```

5 (selected baidu)

100

```
Type !commands to view commands
!qdisplay
Google PageRank: 1
https://en.wikipedia.org/wiki/Baidu
score: 443

Google PageRank: 2
https://en.wikipedia.org/w/index.php?title=List_of_most_popular_websites&action=edit
score: 384

Google PageRank: 3
https://en.wikipedia.org/wiki/Website
score: 378

Google PageRank: 4
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
score: 357

Google PageRank: 5
NETFLIX OI
score: 345
```

!customedit

```
ID: 0
https://en.wikipedia.org/wiki/Baidu
Score: 443

ID: 1
https://en.wikipedia.org/w/index.php?title=List_of_most_popular_websites&action=edit
Score: 384

ID: 2
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
Score: 357
```

0 (selected baidu

```
Enter the ID of the customer who you want to edit. (INTEGER)
0
Customer recieved: https://en.wikipedia.org/wiki/Baidu
Score of customer: 443

KEYWORDS value of this site: 188
Enter the new KEYWORD value for this site (INTEGER).
100
AGE value of this site: 57
Enter the new AGE value for this site (INTEGER).
0
LIKNKS value of this site: 49
Enter the new LINKS value for this site (INTEGER).
0
PAID value of this site: 49
Enter the new PAID value for this site (INTEGER).
250
 new KEYWORDS value of this site: 100
new AGE value of this site: 0
new LIKNKS value of this site: 0
new PAID value of this site: 250
new Score value of this site: 350
Type !commands to view commands
Command not recognized. Type !commands for a list of commands.
Type !commands to view commands
!qdisplay
```

!qdisplay

```
!qdisplay
Google PageRank: 1
https://en.wikipedia.org/w/index.php?title=List_of_most_popular_websites&action=edit
score: 384

Google PageRank: 2
https://en.wikipedia.org/wiki/Website
score: 378

Google PageRank: 3
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
score: 357

Google PageRank: 4
https://en.wikipedia.org/wiki/Baidu
score: 350

Google PageRank: 5
NETFLIX OI
score: 345

Google PageRank: 6
https://en.wikipedia.org/wiki/List_of_most_popular_websites#cite_note-Alexa-3
score: 344

Google PageRank: 7
https://en.wikipedia.org/wiki/Yahoo!
score: 315

Google PageRank: 8
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
score: 315

Google PageRank: 9
https://en.wikipedia.org/wiki/Online_encyclopedia
score: 309

Google PageRank: 10
https://en.wikipedia.org/wiki/Reddit
score: 304
```

!editscore

```
ID: 29
https://en.wikipedia.org/wiki/Wikipedia
Score: 284
```

29 (selected wikipedia)

```
Enter the ID of the customer who you want to edit. (INTEGER)
29
Customer recieved: https://en.wikipedia.org/wiki/Wikipedia
Score of customer: 284
Enter their new score
310
Their new score is 310
Type !commands to view commands
Command not recognized. Type !commands for a list of commands.
Type !commands to view commands
```

Now !qdisplay

```
score: 315

Google PageRank: 8
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
score: 315

Google PageRank: 9
https://en.wikipedia.org/wiki/Wikipedia
score: 310

Google PageRank: 10
https://en.wikipedia.org/wiki/Online_encyclopedia
score: 309
```

Now !bantop

Below is image of !qdisplay before !bantop

```
!qdisplay
Google PageRank: 1
https://en.wikipedia.org/w/index.php?title=List_of_most_popular_websites&action=edit
score: 384

Google PageRank: 2
https://en.wikipedia.org/wiki/Website
score: 378

Google PageRank: 3
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
score: 357

Google PageRank: 4
https://en.wikipedia.org/wiki/Baidu
score: 350

Google PageRank: 5
NETFLIX OI
score: 345

Google PageRank: 6
https://en.wikipedia.org/wiki/List_of_most_popular_websites#cite_note-Alexa-3
score: 344

Google PageRank: 7
https://en.wikipedia.org/wiki/Yahoo!
score: 315

Google PageRank: 8
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
score: 315

Google PageRank: 9
https://en.wikipedia.org/wiki/Wikipedia
score: 310

Google PageRank: 10
https://en.wikipedia.org/wiki/Online_encyclopedia
score: 309
```

!bantop

!qdisplay

```
Google PageRank: 1
https://en.wikipedia.org/wiki/Website
score: 378

Google PageRank: 2
https://en.wikipedia.org/wiki/List_of_most_popular_websites#p-search
score: 357

Google PageRank: 3
https://en.wikipedia.org/wiki/Baidu
score: 350

Google PageRank: 4
NETFLIX OI
score: 345

Google PageRank: 5
https://en.wikipedia.org/wiki/List_of_most_popular_websites#cite_note-Alexa-3
score: 344

Google PageRank: 6
https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
score: 315

Google PageRank: 7
https://en.wikipedia.org/wiki/Yahoo!
score: 315

Google PageRank: 8
https://en.wikipedia.org/wiki/Wikipedia
score: 310

Google PageRank: 9
https://en.wikipedia.org/wiki/Online_encyclopedia
score: 309

Google PageRank: 10
https://en.wikipedia.org/wiki/Reddit
score: 304
```

!highscore

```
Type !commands to view commands
!highscore
Top website is: https://en.wikipedia.org/wiki/Website
Score: 378
Type !commands to view commands
```