

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

How to start = 2 - 3

Implementation summary = 4

Classes and method descriptions = 4 - 14

Calculation of running times = 15 - 22

Problems encountered = 23 - 24

Lessons learned = 24

Testing = 25 - 57

Console tests (actual program) = 25 - 39

Individual heap function tests = 40 - 47

Search tracking = 48 - 49

Web crawler order tracking = 50 - 51

Eclipse program test = 52 - rest

HOW TO START:

I use winrar to extract and zip my files.

```
C:\Users\unded\Desktop>java -jar searcher.jar
Stops searching after 100 searches for convenience.
After 100 searches, rest of array is filled with DUMMY URLs.
So please use vague search terms!

Enter your search term.
```

- 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.

```
**Visiting** Received web page at https://en.wikipedia.org/wiki/N/A
Found (118) links
Searching for the word the cow...
search # 66/100
successes: 0/35

**Visiting** Received web page at https://en.wikipedia.org/wiki/List_of_most_popular_websites#cite_note-notsinc
Found (413) links
Searching for the word the cow...
search # 67/100
successes: 0/35

**Visiting** Received web page at https://en.wikipedia.org/wiki/Brazil
Found (4558) links
Searching for the word the cow...
search # 68/100
successes: 0/35

**Visiting** Received web page at https://en.wikipedia.org/wiki/Netflix
Found (3199) links
Searching for the word the cow...
search # 69/100
successes: 0/35

**Visiting** Received web page at https://en.wikipedia.org/wiki/Streaming_media
Found (730) links
Searching for the word the cow...
search # 70/100
successes: 0/35

**Visiting** Received web page at https://en.wikipedia.org/wiki/Television_programs
Found (564) links
Searching for the word the cow...
search # 71/100
successes: 0/35

**Visiting** Received web page at https://en.wikipedia.org/wiki/Movies
Found (997) links
Searching for the word the cow...
search # 72/100
successes: 0/35

**Visiting** Received web page at https://en.wikipedia.org/wiki/Germany
Found (4628) links
Searching for the word the cow...
search # 73/100
successes: 0/35
```

^ 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

```
Type !commands to view commands
!commands

!search - search again
!topsearches - shows yours most frequently searched terms
!qdisplay - uses priority queue implementation to show top searches
!addsite - adds a website with a custom score
!pay - increase a customer's score by X amount
!customedit - edit the 4 values of a single website
!editscore - directly edit the score of one website
!banstop - Removes the #1 ranked website from Google.
!highscore - shows the relevancy score of the most relevant website
!addcustomsite - Don't call this unless you want to. Implemented it by accident.
!exit - exits the program. Useless.
!displayraw - Displays ALL 35 websites. Used for debugging.
!display - prints out the websites in their proper order with webcrawler pull order with heapsort implementation. Only intended to be used by the initial search.
```

!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[sucesses] = 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.

```
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;
```

^ 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...

```

>  /*
> 1 public class SortTracker {
> 2     ArrayList<String> searches;
> 3
> 4     public SortTracker(ArrayList<String> passed) {
> 5         this.searches = new ArrayList<String>(passed);
> 6     }
> 7

```

```

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".

```

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("!banlist")) {
            banList();
        } 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.

```
② 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.

```
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
10    if (uniques.length <= 10) {
11        for (int i = uniques.length - 1; i >= 0; i--) {
12            System.out.println("Term: " + uniques[i].getSearch());
13            System.out.println("Frequency: " + uniques[i].getFrequency());
14            System.out.println();
15        }
16    } else if (uniques.length > 10) {
17        for (int i = uniques.length - 1; i > uniques.length - 11; i--) {
18            System.out.println("Term: " + uniques[i].getSearch());
19            System.out.println("Frequency: " + uniques[i].getFrequency());
20            System.out.println();
21        }
22    }
23 }
```

[^] 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.

```

/**
 * 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.

```

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 recievied = heap.getUrl(ID);
    System.out.println("Customer recievied: " + recievied.getName());
    System.out.println("Score of customer: " + recievied.getScore());
    System.out.println();

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

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

    System.out.println("LINKS value of this site: " + recievied.getLinks());
    System.out.println("Enter the new LINKS value for this site (INTEGER).");
    int links = scan.nextInt();
    recievied.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 received: " + 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 $O(n)[\text{copies values to new array}] + O(\lg(n))[\text{max heapify}] + c = O(n)$ 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 $n = \# \text{ of searches}$

```

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();
        // bmk
    }
}

```

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

```

71@   public void qDisplay() {
72
73     UrlHeap heap2 = new UrlHeap(heap.getArray().clone());
74
75     PriorityQueue<Url> queue = new PriorityQueue<Url>();
76     int size = heap2.getArray().length;
77
78     if (size <= 10) {
79         for (int i = 0; i < size; i++) {
80             // h2.bmh
81             Url extracted = heap2.Heap_Extract_Max_Url();
82             queue.add(extracted);
83         }
84     } else if (size > 10) {
85         for (int i = 0; i < 10; i++) {
86             // h2.bmh
87             Url extracted = heap2.Heap_Extract_Max_Url();
88             queue.add(extracted);
89         }
90     }
91     size = queue.size();
92     if (size <= 10) {
93         for (int i = 0; i < size; i++) {
94             int rank = i + 1;
95             System.out.println("Google PageRank: " + rank);
96             queue.remove().printValues();
97             System.out.println();
98         }
99     } else if (size > 10) {
100        for (int i = 0; i < 10; i++) {
101            int rank = i + 1;
102            System.out.println("Google PageRank: " + rank);
103            queue.remove().printValues();
104            System.out.println();
105        }
106    }
107 }

```

^ 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 $10 \times n$. It is $10 \times n$ and not $10 \times \lg(n)$ because heap extract max makes a new array without extracted node in my implementation. Therefore, the runtime of this is **O(n)**.

```

^/
@  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 $n \lg n + n + c$, which is $O(n \lg n)$

```

    /
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 $n \lg n$. Tracker.getsearchfrequencies() was already calculated to be $O(n)^2$. The for loop runs no more than 10 times. $n \lg n + n^2 + c = O(n^2)$.

```

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 $n \lg n$. It is not needed. Heap increase key is $\lg(n)$. The for loop is $O(n)$. $n \lg n + \lg n + n = O(n \lg n)$. 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). $2n + c = O(n)$

```

1     /*
2      public void editScore() {
3          // h.bmh
4          qDisplay();
5
5      for (int i = 0; i < heap.getArray().length; i++) {
6          // h.bmh
7          System.out.println("ID: " + i);
8          heap.printValue(i);
9          System.out.println();
10     }
11     System.out.println("Enter the ID of the customer who you want to edit. (INTEGER)");
12     int ID = scan.nextInt();
13     Url recieved = heap.getUrl(ID);
14     System.out.println("Customer recieved: " + recieved.getName());
15     System.out.println("Score of customer: " + recieved.getScore());
16     System.out.println("Enter their new score");
17     int newScore = scan.nextInt();
18     recieved.setScore(newScore);
19     heap.Build_Max_Heap(); // re heapifies. Can't use Increase key. Increase key only applies to bigger
20                           // values.
21     System.out.println("Their new score is " + recieved.getScore());
22 }
23 }

```

^ 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 Program 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 $n\lg n$, 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 $n\lg n$ 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”

```

calling heapSortDisplay()
Google PageRank: 1
PageRank BEFORE being sorted: 4
DUMMY_3_https://en.wikipedia.org/wiki/Windows_Live
Score: 413

Google PageRank: 2
PageRank BEFORE being sorted: 30
DUMMY_29_https://en.wikipedia.org/wiki/Video_hosting_service
Score: 376

Google PageRank: 3
PageRank BEFORE being sorted: 10
DUMMY_9_https://en.wikipedia.org/wiki/Website
Score: 365

Google PageRank: 4
PageRank BEFORE being sorted: 31
DUMMY_30_https://en.wikipedia.org/wiki/China
Score: 361

Google PageRank: 5
PageRank BEFORE being sorted: 21
DUMMY_20_https://en.wikipedia.org/wiki/Facebook
Score: 360

Google PageRank: 6
PageRank BEFORE being sorted: 33
DUMMY_32_https://en.wikipedia.org/wiki/United_States
Score: 338

Google PageRank: 7
PageRank BEFORE being sorted: 32
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
Score: 325

Google PageRank: 8
PageRank BEFORE being sorted: 18
DUMMY_17_https://en.wikipedia.org/wiki/Netflix
Score: 311

Google PageRank: 9
PageRank BEFORE being sorted: 8
DUMMY_7_https://en.wikipedia.org/wiki/Tmall
Score: 311

Google PageRank: 10
PageRank BEFORE being sorted: 17
DUMMY_16_https://en.wikipedia.org/wiki/Yahoo!_Japan
Score: 308

```

^ vague term = dummy urls

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_29_https://en.wikipedia.org/wiki/Video_hosting_service
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

W
Google PageRank: 8
DUMMY_17_https://en.wikipedia.org/wiki/Netflix
score: 312

Se
Google PageRank: 9
DUMMY_7_https://en.wikipedia.org/wiki/Tmall
score: 311

S
Google PageRank: 10
DUMMY_16_https://en.wikipedia.org/wiki/Yahoo!_Japan
score: 306

Type !commands to view commands
-
```

Enter “!display”

```
C:\ Command Prompt - java -jar searcher.jar
R DUMMY_3_https://en.wikipedia.org/wiki/Windows_Live
Score: 413

Google PageRank: 2
PageRank BEFORE being sorted: 30
DUMMY_29_https://en.wikipedia.org/wiki/Video_hosting_service
Score: 376

Google PageRank: 3
PageRank BEFORE being sorted: 10
DUMMY_9_https://en.wikipedia.org/wiki/Website
Score: 365

U Google PageRank: 4
PageRank BEFORE being sorted: 31
DUMMY_30_https://en.wikipedia.org/wiki/China
Score: 361

Google PageRank: 5
PageRank BEFORE being sorted: 21
DUMMY_20_https://en.wikipedia.org/wiki/Facebook
Score: 360

Google PageRank: 6
PageRank BEFORE being sorted: 33
DUMMY_32_https://en.wikipedia.org/wiki/United_States
Score: 338

Google PageRank: 7
DPAGERank BEFORE being sorted: 32
DUMMY_31_https://en.wikipedia.org/wiki/Social_news
Score: 325

Google PageRank: 8
W PAGERank BEFORE being sorted: 18
DUMMY_17_https://en.wikipedia.org/wiki/Netflix
Score: 312

Google PageRank: 9
e PAGERank BEFORE being sorted: 8
DUMMY_7_https://en.wikipedia.org/wiki/Tmall
Score: 311

Google PageRank: 10
S PAGERank BEFORE being sorted: 17
DUMMY_16_https://en.wikipedia.org/wiki/Yahoo!_Japan
Score: 306

Type !commands to view commands
```

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
!
```

```
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

```
DUMMY_24 https://en.wikipedia.org/wiki/List_of_most_popular_websites#cite_note-notsimi
Score: 179

ID: 25
DUMMY_25 https://en.wikipedia.org/wiki/List_of_most_popular_websites#mw-head
Score: 232

ID: 26
DUMMY_26 https://en.wikipedia.org/wiki/XVideos
Score: 201

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_1 https://en.wikipedia.org/wiki/Japan
Score: 197

ID: 32
DUMMY_15 https://en.wikipedia.org/wiki/E-commerce_payment_system
Score: 186

ID: 33
DUMMY_33 https://en.wikipedia.org/wiki/E-commerce
Score: 290

ID: 34
DUMMY_34 https://en.wikipedia.org/wiki/Software
Score: 114

ID: 35
DUMMY_8 https://en.wikipedia.org/wiki/Internet
Score: 250

Enter the ID of the customer who paid. (INTEGER)
```

Enter “31” to select japan

Japan’s score is 197. Say they paid 200

Enter “200”

Note: $200 + 197 = 397.$

```
: Enter the ID of the customer who paid. (INTEGER)
31
The customer is DUMMY_1_https://en.wikipedia.org/wiki/Japan
Enter the amount this customer paid, rounded to the nearest +INTEGER
200
Customer's score has been increased by 200
Type !commands to view commands
Command not recognized. Type !commands for a list of commands.
Type !commands to view commands
```

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
Dsore: 360

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

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

eGoogle PageRank: 9
NETFLIX
score: 313

SGoogle 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”

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

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

ID: 31
DUMMY_7_https://en.wikipedia.org/wiki/Tmall
Score: 311

ID: 32
DUMMY_15_https://en.wikipedia.org/wiki/E-commerce_payment_system
Score: 186

ID: 33
DUMMY_33_https://en.wikipedia.org/wiki/E-commerce
Score: 290

ID: 34
DUMMY_34_https://en.wikipedia.org/wiki/Software
Score: 114
```

Enter “31” to select Tmall

Enter “50” “100” “100” “100”

```
ENTER THE ID OF THE CUSTOMER WHO YOU WANT TO EDIT (+INTEGER)
31
Customer received: DUMMY_7_https://en.wikipedia.org/wiki/Tmall
Score of customer: 311

KEYWORDS value of this site: 159
Enter the new KEYWORD value for this site (+INTEGER).
50
AGE value of this site: 77
Enter the new AGE value for this site (+INTEGER).
100
LINKS value of this site: 4
Enter the new LINKS value for this site (+INTEGER).
100
PAID value of this site: 71
Enter the new PAID value for this site (+INTEGER).
100
new KEYWORDS value of this site: 50
new AGE value of this site: 100
new LINKS value of this site: 100
new PAID value of this site: 100
new Score value of this site: 350
S>Type !commands to view commands
Command not recognized. Type !commands for a list of commands.
Type !commands to view commands
```

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 “!leditscore”

```
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 received: 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”

```
Type !commands to view commands
!bantop
removed a website with progress 413
Type !commands to view commands
!qdisplay
Google PageRank: 1
DUMMY_1_https://en.wikipedia.org/wiki/Japan
score: 397

Google PageRank: 2
DUMMY_29_https://en.wikipedia.org/wiki/Video_hosting_ser
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_7_https://en.wikipedia.org/wiki/Tmall
score: 350

Google PageRank: 7
DUMMY_13_https://en.wikipedia.org/wiki/YouTube
score: 349

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
```

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

```

```

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

```

Console Output:

```

<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
11
14
16

```

Max heapify (I think I got array from textbook)

The screenshot shows an IDE interface with two main panes. The top pane is a code editor displaying Java code for a `Test_Max_Heapify` class. The bottom pane is a terminal window showing the execution of the program and its output.

```
1 package heapsort_tester;
2
3 public class Test_Max_Heapify {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         int[] arr = new int[] {16,4,10,14,7,9,3,2,8,1};
8         HeapArray heap = new HeapArray(arr);
9
10        heap.Max_Heapify(arr, 1);
11
12        for(int x:arr) {
13            System.out.println(x);
14        }
15        //works for one case
16        //{16,4,10,14,7,9,3,2,8,1}
17
18
19    }
20
21 }
22 }
```

The terminal window shows the following output:

```
<terminated> Test_Max_Heapify (1) [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe (Oct 20, 2016)
16
10
14
8
7
9
3
2
4
1
```

Build max heap

The screenshot shows a Java development environment with the following code in the editor:

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

The console tab is selected, showing the output of the program:

```
<terminated> Test_Build_Max_Heap (1) [Java Application] C:\Program Files\Java\jdk1.8.0_251\bin\java.exe
16
14
10
8
7
9
3
2
4
1
```

Heap extract max

The screenshot shows a Java development environment with the following details:

- Code Editor:** Displays the source code for `Test_Heap_Extract_Max.java`. The code initializes an array with values [16, 1, 15, 134, 23, 44], creates a `HeapQueue` heap, builds it, extracts the maximum value (44), and prints the remaining values (23, 15, 1).
- Console Output:** Shows the terminal window with the application's output:

```
<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

The screenshot shows an IDE interface with a code editor and a console window.

Code Editor:

```
1 package heapsort_tester;
2
3 public class Test_Increase_Key {
4     public static void main(String[] args) {
5         int[] arr = new int[] {16,14,21,12,155};
6         HeapQueue heap = new HeapQueue(arr);
7
8         heap.Build_Max_Heap();
9         heap.printValues();
10
11        System.out.println();
12
13        heap.Heap_Increase_Key(2, 156);
14        heap.printValues();
15    }
16}
17
18
```

Console Window:

```
<terminated> Test_Increase_Key (1) [Java Application] C:\Program Files\Java\jdk1
155
16
21
12
14

156
155
16
12
14
```

Max Heap insert

The screenshot shows the Eclipse IDE interface. The top bar has tabs for various projects: Test_Heap_Ex..., Test_Heaps..., Test_Increa..., Test_Max_Hea..., and a closed tab. The main editor window displays the following Java code:

```

1 package heapsort_tester;
2
3 public class Test_Max_Heap_Insert {
4
5     public static void main(String[] args) {
6         int[] arr = new int[] {16,1,15,134,23,44};
7         HeapQueue heap = new HeapQueue(arr);
8
9         heap.Build_Max_Heap();
10        heap.Max_Heap_Insert(50);
11        heap.printValues();
12
13        System.out.println();
14
15        heap.Build_Max_Heap();
16        heap.printValues();
17    }
18
19 }

```

Below the editor is the Eclipse toolbar with icons for file operations, search, and help.

The bottom part of the interface is the Console view, which shows the output of the application:

```

<terminated> Test_Max_Heap_Insert (1) [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe (0)
134
50
44
23
23
16
15
1

```

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

This screenshot shows the Eclipse IDE interface again. The top bar has tabs for Programming-1 (1).pdf, Introduction to Algorithms, and a closed tab. The main editor window displays the following Java code:

```

1 package UrlHeap_Working;
2
3 public class test_heap_maximum_for_report {
4
5     public static void main(String[] args) {
6         Url name16 = new Url("name16",16);
7         Url name1 = new Url("name1",1);
8         Url name15 = new Url("name15",15);
9         Url name134 = new Url("name134",134);
10        Url name23 = new Url("name23",23);
11        Url name44 = new Url("name44",44);
12
13        Url[] arr = new Url[6];
14
15        arr[0]= name16;
16        arr[1]= name1;
17        arr[2]= name15;
18        arr[3]= name134;
19        arr[4]= name23;
20        arr[5]= name44;
21        UrIHeap heap = new UrIHeap(arr);
22        heap.Build_Max_Heap();
23        System.out.println(heap.Heap_Maximum());
24

```

The left side of the interface shows the Project Explorer and Package Explorer views, displaying the project structure and source files.

```

3 public class Test_Heap_Extract_Max {
4
5@   public static void main(String[] args) {
6     Url name16 = new Url("name16",16);
7     Url name1 = new Url("name1",1);
8     Url name15 = new Url("name15",15);
9     Url name14 = new Url("name14",134);
10    Url name23 = new Url("name23",23);
11    Url name44 = new Url("name44",44);
12
13    Url[] arr = new Url[6];
14
15    arr[0]= name16;
16    arr[1]= name1;
17    arr[2]= name15;
18    arr[3]= name14;
19    arr[4]= name23;
20    arr[5]= name44;
21    UrlHeap heap = new UrlHeap(arr);
22    heap.Build_Max_Heap();
23    heap.Heap_Extract_Max();
24    heap.printValues();
25

```

Problems @ Javadoc Declaration Console Search

<terminated> Test_Heap_Extract_Max (3) [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe

```

name44
score: 44

name23
score: 23

name15
score: 15

name1
score: 1

name16
score: 16

```

Heapsort with urls

Console

<terminated> Test_Url_HeapSort [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe (Oct 20, 2018)

```

name1
score: 1
name2
score: 2
name3
score: 3
name4
score: 4
name5
score: 5
name6
score: 6
name7
score: 7
name8
score: 8
name9
score: 9
name10
score: 10
name11
score: 11
name12
score: 12
name13
score: 13
name14
score: 14
name15
score: 15
name16
score: 16

```

Test_Url_HeapSort.java - Eclipse

```

1 //WORKS
2
3 public class Test_Url_HeapSort {
4
5@   public static void main(String[] args) {
6     Url name16 = new Url("name16",16);
7     Url name4 = new Url("name4",4);
8     Url name10 = new Url("name10",10);
9     Url name14 = new Url("name14",14);
10    Url name7 = new Url("name7",7);
11    Url name9 = new Url("name9",9);
12    Url name3 = new Url("name3",3);
13    Url name2 = new Url("name2",2);
14    Url name8 = new Url("name8",8);
15    Url name1 = new Url("name1",1);
16    Url[] arr = new Url[10];
17    arr[0]= name16;
18    arr[1]= name4;
19    arr[2]= name10;
20    arr[3]= name14;
21    arr[4]= name7;
22    arr[5]= name9;
23    arr[6]= name3;
24    arr[7]= name2;
25    arr[8]= name8;
26    arr[9]= name1;
27
28    UrlHeap heap = new UrlHeap(arr);
29    heap.HeapSort();
30    heap.printValues();
31    //heapsort works
32
33
34

```

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

The screenshot shows an IDE interface with multiple windows. The left window displays a console log with several entries, each consisting of a name and a recommendation score. The right window shows the source code for a Java application named 'z_testing_build_max_heap'. The code includes a main method that creates URLs for names 16 through 44 and stores them in an array. It then creates a 'UrlHeap' object, prints its values, and performs a 'Build_Max_Heap' operation before printing the values again. The bottom right window shows a search results summary for 'max_heapify' with 357 matches across 10 files.

```
<terminated> z_testing_build_max_heap [Java Application] C:\Program Files\Java\jdk1.8.0_102\bin\javaw.exe (Oct 10, 2018 4:28:43 PM)  
name16  
Recommendation score: 16  
  
name1  
Recommendation score: 1  
  
name15  
Recommendation score: 15  
  
ss  
name134  
Recommendation score: 134  
  
ds  
name23  
Recommendation score: 23  
  
its  
name44  
Recommendation score: 44  
  
gh  
name134  
Recommendation score: 134  
  
out  
name23  
Recommendation score: 23  
  
ou  
name44  
Recommendation score: 44  
  
name1  
Recommendation score: 1  
  
ts  
name16  
Recommendation score: 16  
  
its  
name15  
Recommendation score: 15
```

```
ipse2018 - CS 146 HW2 V4 added set this version and adding real priorityqueue implementation/src/v5.1_debugging_build_max_heap.java  
File Source Refactor Navigate Project Run Window Help  
Console  
Recommendation score: 16  
name16  
Recommendation score: 1  
name15  
Recommendation score: 15  
ss  
name134  
Recommendation score: 134  
ds  
name23  
Recommendation score: 23  
name44  
Recommendation score: 44  
gh  
name134  
Recommendation score: 134  
out  
name23  
Recommendation score: 23  
ou  
name44  
Recommendation score: 44  
name1  
Recommendation score: 1  
ts  
name16  
Recommendation score: 16  
its  
name15  
Recommendation score: 15
```

```
age Explorer  
rawler with search tracking  
S 146 HW 1  
S 146 HW2 V1  
S 146 HW2 V10 working  
S 146 HW2 V2 - working on problem set  
S 146 HW2 V3 - BACKUP - WORKING V  
S 146 HW2 V4 added set this version ar  
JRE System Library [JavaSE-1.8]  
src  
v4_fixed__adding_ability_to_edit_v  
MainJava  
New_Program_Class.java  
SearchFrequencyJava  
SearchHeap.java  
SortTracker.java  
SpiderJava  
SpiderLegJava  
UrlJava  
UrlHeap.java  
v5.1_debugging_build_max_heap  
MainJava  
New_Program_Class.java  
SearchFrequencyJava  
SearchHeap.java  
SortTracker.java  
SpiderJava  
SpiderLegJava  
UrlJava  
UrlHeap.java
```

```
3 public class z_testing_build_max_heap {  
4  
5     public static void main(String[] args) {  
6         Url name16 = new Url("name16", 16);  
7         Url name1 = new Url("name1", 1);  
8         Url name15 = new Url("name15", 15);  
9         Url name134 = new Url("name134", 134);  
10        Url name23 = new Url("name23", 23);  
11        Url name44 = new Url("name44", 44);  
12  
13        Url[] arr = new Url[6];  
14  
15        arr[0] = name16;  
16        arr[1] = name1;  
17        arr[2] = name15;  
18        arr[3] = name134;  
19        arr[4] = name23;  
20        arr[5] = name44;  
21  
22        UrlHeap heap = new UrlHeap(arr);  
23        heap.printValues();  
24        System.out.println();  
25        heap.Build_Max_Heap();  
26        heap.printValues();  
27  
28    }  
29  
30    public static void main(String[] args) {  
31        Url name16 = new Url("name16", 16);  
32        Url name1 = new Url("name1", 1);  
33        Url name15 = new Url("name15", 15);  
34        Url name134 = new Url("name134", 134);  
35        Url name23 = new Url("name23", 23);  
36        Url name44 = new Url("name44", 44);  
37  
38        Url[] arr = new Url[6];  
39  
40        arr[0] = name16;  
41        arr[1] = name1;  
42        arr[2] = name15;  
43        arr[3] = name134;  
44        arr[4] = name23;  
45        arr[5] = name44;  
46  
47        UrlHeap heap = new UrlHeap(arr);  
48        heap.printValues();  
49        System.out.println();  
50        heap.Build_Max_Heap();  
51        heap.printValues();  
52    }  
53  
54    public static void main(String[] args) {  
55        Url name16 = new Url("name16", 16);  
56        Url name1 = new Url("name1", 1);  
57        Url name15 = new Url("name15", 15);  
58        Url name134 = new Url("name134", 134);  
59        Url name23 = new Url("name23", 23);  
60        Url name44 = new Url("name44", 44);  
61  
62        Url[] arr = new Url[6];  
63  
64        arr[0] = name16;  
65        arr[1] = name1;  
66        arr[2] = name15;  
67        arr[3] = name134;  
68        arr[4] = name23;  
69        arr[5] = name44;  
70  
71        UrlHeap heap = new UrlHeap(arr);  
72        heap.printValues();  
73        System.out.println();  
74        heap.Build_Max_Heap();  
75        heap.printValues();  
76    }  
77  
78    public static void main(String[] args) {  
79        Url name16 = new Url("name16", 16);  
80        Url name1 = new Url("name1", 1);  
81        Url name15 = new Url("name15", 15);  
82        Url name134 = new Url("name134", 134);  
83        Url name23 = new Url("name23", 23);  
84        Url name44 = new Url("name44", 44);  
85  
86        Url[] arr = new Url[6];  
87  
88        arr[0] = name16;  
89        arr[1] = name1;  
90        arr[2] = name15;  
91        arr[3] = name134;  
92        arr[4] = name23;  
93        arr[5] = name44;  
94  
95        UrlHeap heap = new UrlHeap(arr);  
96        heap.printValues();  
97        System.out.println();  
98        heap.Build_Max_Heap();  
99        heap.printValues();  
100    }  
101  
102    public static void main(String[] args) {  
103        Url name16 = new Url("name16", 16);  
104        Url name1 = new Url("name1", 1);  
105        Url name15 = new Url("name15", 15);  
106        Url name134 = new Url("name134", 134);  
107        Url name23 = new Url("name23", 23);  
108        Url name44 = new Url("name44", 44);  
109  
110        Url[] arr = new Url[6];  
111  
112        arr[0] = name16;  
113        arr[1] = name1;  
114        arr[2] = name15;  
115        arr[3] = name134;  
116        arr[4] = name23;  
117        arr[5] = name44;  
118  
119        UrlHeap heap = new UrlHeap(arr);  
120        heap.printValues();  
121        System.out.println();  
122        heap.Build_Max_Heap();  
123        heap.printValues();  
124    }  
125  
126    public static void main(String[] args) {  
127        Url name16 = new Url("name16", 16);  
128        Url name1 = new Url("name1", 1);  
129        Url name15 = new Url("name15", 15);  
130        Url name134 = new Url("name134", 134);  
131        Url name23 = new Url("name23", 23);  
132        Url name44 = new Url("name44", 44);  
133  
134        Url[] arr = new Url[6];  
135  
136        arr[0] = name16;  
137        arr[1] = name1;  
138        arr[2] = name15;  
139        arr[3] = name134;  
140        arr[4] = name23;  
141        arr[5] = name44;  
142  
143        UrlHeap heap = new UrlHeap(arr);  
144        heap.printValues();  
145        System.out.println();  
146        heap.Build_Max_Heap();  
147        heap.printValues();  
148    }  
149  
150    public static void main(String[] args) {  
151        Url name16 = new Url("name16", 16);  
152        Url name1 = new Url("name1", 1);  
153        Url name15 = new Url("name15", 15);  
154        Url name134 = new Url("name134", 134);  
155        Url name23 = new Url("name23", 23);  
156        Url name44 = new Url("name44", 44);  
157  
158        Url[] arr = new Url[6];  
159  
160        arr[0] = name16;  
161        arr[1] = name1;  
162        arr[2] = name15;  
163        arr[3] = name134;  
164        arr[4] = name23;  
165        arr[5] = name44;  
166  
167        UrlHeap heap = new UrlHeap(arr);  
168        heap.printValues();  
169        System.out.println();  
170        heap.Build_Max_Heap();  
171        heap.printValues();  
172    }  
173  
174    public static void main(String[] args) {  
175        Url name16 = new Url("name16", 16);  
176        Url name1 = new Url("name1", 1);  
177        Url name15 = new Url("name15", 15);  
178        Url name134 = new Url("name134", 134);  
179        Url name23 = new Url("name23", 23);  
180        Url name44 = new Url("name44", 44);  
181  
182        Url[] arr = new Url[6];  
183  
184        arr[0] = name16;  
185        arr[1] = name1;  
186        arr[2] = name15;  
187        arr[3] = name134;  
188        arr[4] = name23;  
189        arr[5] = name44;  
190  
191        UrlHeap heap = new UrlHeap(arr);  
192        heap.printValues();  
193        System.out.println();  
194        heap.Build_Max_Heap();  
195        heap.printValues();  
196    }  
197  
198    public static void main(String[] args) {  
199        Url name16 = new Url("name16", 16);  
200        Url name1 = new Url("name1", 1);  
201        Url name15 = new Url("name15", 15);  
202        Url name134 = new Url("name134", 134);  
203        Url name23 = new Url("name23", 23);  
204        Url name44 = new Url("name44", 44);  
205  
206        Url[] arr = new Url[6];  
207  
208        arr[0] = name16;  
209        arr[1] = name1;  
210        arr[2] = name15;  
211        arr[3] = name134;  
212        arr[4] = name23;  
213        arr[5] = name44;  
214  
215        UrlHeap heap = new UrlHeap(arr);  
216        heap.printValues();  
217        System.out.println();  
218        heap.Build_Max_Heap();  
219        heap.printValues();  
220    }  
221  
222    public static void main(String[] args) {  
223        Url name16 = new Url("name16", 16);  
224        Url name1 = new Url("name1", 1);  
225        Url name15 = new Url("name15", 15);  
226        Url name134 = new Url("name134", 134);  
227        Url name23 = new Url("name23", 23);  
228        Url name44 = new Url("name44", 44);  
229  
230        Url[] arr = new Url[6];  
231  
232        arr[0] = name16;  
233        arr[1] = name1;  
234        arr[2] = name15;  
235        arr[3] = name134;  
236        arr[4] = name23;  
237        arr[5] = name44;  
238  
239        UrlHeap heap = new UrlHeap(arr);  
240        heap.printValues();  
241        System.out.println();  
242        heap.Build_Max_Heap();  
243        heap.printValues();  
244    }  
245  
246    public static void main(String[] args) {  
247        Url name16 = new Url("name16", 16);  
248        Url name1 = new Url("name1", 1);  
249        Url name15 = new Url("name15", 15);  
250        Url name134 = new Url("name134", 134);  
251        Url name23 = new Url("name23", 23);  
252        Url name44 = new Url("name44", 44);  
253  
254        Url[] arr = new Url[6];  
255  
256        arr[0] = name16;  
257        arr[1] = name1;  
258        arr[2] = name15;  
259        arr[3] = name134;  
260        arr[4] = name23;  
261        arr[5] = name44;  
262  
263        UrlHeap heap = new UrlHeap(arr);  
264        heap.printValues();  
265        System.out.println();  
266        heap.Build_Max_Heap();  
267        heap.printValues();  
268    }  
269  
270    public static void main(String[] args) {  
271        Url name16 = new Url("name16", 16);  
272        Url name1 = new Url("name1", 1);  
273        Url name15 = new Url("name15", 15);  
274        Url name134 = new Url("name134", 134);  
275        Url name23 = new Url("name23", 23);  
276        Url name44 = new Url("name44", 44);  
277  
278        Url[] arr = new Url[6];  
279  
280        arr[0] = name16;  
281        arr[1] = name1;  
282        arr[2] = name15;  
283        arr[3] = name134;  
284        arr[4] = name23;  
285        arr[5] = name44;  
286  
287        UrlHeap heap = new UrlHeap(arr);  
288        heap.printValues();  
289        System.out.println();  
290        heap.Build_Max_Heap();  
291        heap.printValues();  
292    }  
293  
294    public static void main(String[] args) {  
295        Url name16 = new Url("name16", 16);  
296        Url name1 = new Url("name1", 1);  
297        Url name15 = new Url("name15", 15);  
298        Url name134 = new Url("name134", 134);  
299        Url name23 = new Url("name23", 23);  
300        Url name44 = new Url("name44", 44);  
301  
302        Url[] arr = new Url[6];  
303  
304        arr[0] = name16;  
305        arr[1] = name1;  
306        arr[2] = name15;  
307        arr[3] = name134;  
308        arr[4] = name23;  
309        arr[5] = name44;  
310  
311        UrlHeap heap = new UrlHeap(arr);  
312        heap.printValues();  
313        System.out.println();  
314        heap.Build_Max_Heap();  
315        heap.printValues();  
316    }  
317  
318    public static void main(String[] args) {  
319        Url name16 = new Url("name16", 16);  
320        Url name1 = new Url("name1", 1);  
321        Url name15 = new Url("name15", 15);  
322        Url name134 = new Url("name134", 134);  
323        Url name23 = new Url("name23", 23);  
324        Url name44 = new Url("name44", 44);  
325  
326        Url[] arr = new Url[6];  
327  
328        arr[0] = name16;  
329        arr[1] = name1;  
330        arr[2] = name15;  
331        arr[3] = name134;  
332        arr[4] = name23;  
333        arr[5] = name44;  
334  
335        UrlHeap heap = new UrlHeap(arr);  
336        heap.printValues();  
337        System.out.println();  
338        heap.Build_Max_Heap();  
339        heap.printValues();  
340    }  
341  
342    public static void main(String[] args) {  
343        Url name16 = new Url("name16", 16);  
344        Url name1 = new Url("name1", 1);  
345        Url name15 = new Url("name15", 15);  
346        Url name134 = new Url("name134", 134);  
347        Url name23 = new Url("name23", 23);  
348        Url name44 = new Url("name44", 44);  
349  
350        Url[] arr = new Url[6];  
351  
352        arr[0] = name16;  
353        arr[1] = name1;  
354        arr[2] = name15;  
355        arr[3] = name134;  
356        arr[4] = name23;  
357        arr[5] = name44;  
358  
359        UrlHeap heap = new UrlHeap(arr);  
360        heap.printValues();  
361        System.out.println();  
362        heap.Build_Max_Heap();  
363        heap.printValues();  
364    }  
365  
366    public static void main(String[] args) {  
367        Url name16 = new Url("name16", 16);  
368        Url name1 = new Url("name1", 1);  
369        Url name15 = new Url("name15", 15);  
370        Url name134 = new Url("name134", 134);  
371        Url name23 = new Url("name23", 23);  
372        Url name44 = new Url("name44", 44);  
373  
374        Url[] arr = new Url[6];  
375  
376        arr[0] = name16;  
377        arr[1] = name1;  
378        arr[2] = name15;  
379        arr[3] = name134;  
380        arr[4] = name23;  
381        arr[5] = name44;  
382  
383        UrlHeap heap = new UrlHeap(arr);  
384        heap.printValues();  
385        System.out.println();  
386        heap.Build_Max_Heap();  
387        heap.printValues();  
388    }  
389  
390    public static void main(String[] args) {  
391        Url name16 = new Url("name16", 16);  
392        Url name1 = new Url("name1", 1);  
393        Url name15 = new Url("name15", 15);  
394        Url name134 = new Url("name134", 134);  
395        Url name23 = new Url("name23", 23);  
396        Url name44 = new Url("name44", 44);  
397  
398        Url[] arr = new Url[6];  
399  
400        arr[0] = name16;  
401        arr[1] = name1;  
402        arr[2] = name15;  
403        arr[3] = name134;  
404        arr[4] = name23;  
405        arr[5] = name44;  
406  
407        UrlHeap heap = new UrlHeap(arr);  
408        heap.printValues();  
409        System.out.println();  
410        heap.Build_Max_Heap();  
411        heap.printValues();  
412    }  
413  
414    public static void main(String[] args) {  
415        Url name16 = new Url("name16", 16);  
416        Url name1 = new Url("name1", 1);  
417        Url name15 = new Url("name15", 15);  
418        Url name134 = new Url("name134", 134);  
419        Url name23 = new Url("name23", 23);  
420        Url name44 = new Url("name44", 44);  
421  
422        Url[] arr = new Url[6];  
423  
424        arr[0] = name16;  
425        arr[1] = name1;  
426        arr[2] = name15;  
427        arr[3] = name134;  
428        arr[4] = name23;  
429        arr[5] = name44;  
430  
431        UrlHeap heap = new UrlHeap(arr);  
432        heap.printValues();  
433        System.out.println();  
434        heap.Build_Max_Heap();  
435        heap.printValues();  
436    }  
437  
438    public static void main(String[] args) {  
439        Url name16 = new Url("name16", 16);  
440        Url name1 = new Url("name1", 1);  
441        Url name15 = new Url("name15", 15);  
442        Url name134 = new Url("name134", 134);  
443        Url name23 = new Url("name23", 23);  
444        Url name44 = new Url("name44", 44);  
445  
446        Url[] arr = new Url[6];  
447  
448        arr[0] = name16;  
449        arr[1] = name1;  
450        arr[2] = name15;  
451        arr[3] = name134;  
452        arr[4] = name23;  
453        arr[5] = name44;  
454  
455        UrlHeap heap = new UrlHeap(arr);  
456        heap.printValues();  
457        System.out.println();  
458        heap.Build_Max_Heap();  
459        heap.printValues();  
460    }  
461  
462    public static void main(String[] args) {  
463        Url name16 = new Url("name16", 16);  
464        Url name1 = new Url("name1", 1);  
465        Url name15 = new Url("name15", 15);  
466        Url name134 = new Url("name134", 134);  
467        Url name23 = new Url("name23", 23);  
468        Url name44 = new Url("name44", 44);  
469  
470        Url[] arr = new Url[6];  
471  
472        arr[0] = name16;  
473        arr[1] = name1;  
474        arr[2] = name15;  
475        arr[3] = name134;  
476        arr[4] = name23;  
477        arr[5] = name44;  
478  
479        UrlHeap heap = new UrlHeap(arr);  
480        heap.printValues();  
481        System.out.println();  
482        heap.Build_Max_Heap();  
483        heap.printValues();  
484    }  
485  
486    public static void main(String[] args) {  
487        Url name16 = new Url("name16", 16);  
488        Url name1 = new Url("name1", 1);  
489        Url name15 = new Url("name15", 15);  
490        Url name134 = new Url("name134", 134);  
491        Url name23 = new Url("name23", 23);  
492        Url name44 = new Url("name44", 44);  
493  
494        Url[] arr = new Url[6];  
495  
496        arr[0] = name16;  
497        arr[1] = name1;  
498        arr[2] = name15;  
499        arr[3] = name134;  
500        arr[4] = name23;  
501        arr[5] = name44;  
502  
503        UrlHeap heap = new UrlHeap(arr);  
504        heap.printValues();  
505        System.out.println();  
506        heap.Build_Max_Heap();  
507        heap.printValues();  
508    }  
509  
510    public static void main(String[] args) {  
511        Url name16 = new Url("name16", 16);  
512        Url name1 = new Url("name1", 1);  
513        Url name15 = new Url("name15", 15);  
514        Url name134 = new Url("name134", 134);  
515        Url name23 = new Url("name23", 23);  
516        Url name44 = new Url("name44", 44);  
517  
518        Url[] arr = new Url[6];  
519  
520        arr[0] = name16;  
521        arr[1] = name1;  
522        arr[2] = name15;  
523        arr[3] = name134;  
524        arr[4] = name23;  
525        arr[5] = name44;  
526  
527        UrlHeap heap = new UrlHeap(arr);  
528        heap.printValues();  
529        System.out.println();  
530        heap.Build_Max_Heap();  
531        heap.printValues();  
532    }  
533  
534    public static void main(String[] args) {  
535        Url name16 = new Url("name16", 16);  
536        Url name1 = new Url("name1", 1);  
537        Url name15 = new Url("name15", 15);  
538        Url name134 = new Url("name134", 134);  
539        Url name23 = new Url("name23", 23);  
540        Url name44 = new Url("name44", 44);  
541  
542        Url[] arr = new Url[6];  
543  
544        arr[0] = name16;  
545        arr[1] = name1;  
546        arr[2] = name15;  
547        arr[3] = name134;  
548        arr[4] = name23;  
549        arr[5] = name44;  
550  
551        UrlHeap heap = new UrlHeap(arr);  
552        heap.printValues();  
553        System.out.println();  
554        heap.Build_Max_Heap();  
555        heap.printValues();  
556    }  
557  
558    public static void main(String[] args) {  
559        Url name16 = new Url("name16", 16);  
560        Url name1 = new Url("name1", 1);  
561        Url name15 = new Url("name15", 15);  
562        Url name134 = new Url("name134", 134);  
563        Url name23 = new Url("name23", 23);  
564        Url name44 = new Url("name44", 44);  
565  
566        Url[] arr = new Url[6];  
567  
568        arr[0] = name16;  
569        arr[1] = name1;  
570        arr[2] = name15;  
571        arr[3] = name134;  
572        arr[4] = name23;  
573        arr[5] = name44;  
574  
575        UrlHeap heap = new UrlHeap(arr);  
576        heap.printValues();  
577        System.out.println();  
578        heap.Build_Max_Heap();  
579        heap.printValues();  
580    }  
581  
582    public static void main(String[] args) {  
583        Url name16 = new Url("name16", 16);  
584        Url name1 = new Url("name1", 1);  
585        Url name15 = new Url("name15", 15);  
586        Url name134 = new Url("name134", 134);  
587        Url name23 = new Url("name23", 23);  
588        Url name44 = new Url("name44", 44);  
589  
590        Url[] arr = new Url[6];  
591  
592        arr[0] = name16;  
593        arr[1] = name1;  
594        arr[2] = name15;  
595        arr[3] = name134;  
596        arr[4] = name23;  
597        arr[5] = name44;  
598  
599        UrlHeap heap = new UrlHeap(arr);  
600        heap.printValues();  
601        System.out.println();  
602        heap.Build_Max_Heap();  
603        heap.printValues();  
604    }  
605  
606    public static void main(String[] args) {  
607        Url name16 = new Url("name16", 16);  
608        Url name1 = new Url("name1", 1);  
609        Url name15 = new Url("name15", 15);  
610        Url name134 = new Url("name134", 134);  
611        Url name23 = new Url("name23", 23);  
612        Url name44 = new Url("name44", 44);  
613  
614        Url[] arr = new Url[6];  
615  
616        arr[0] = name16;  
617        arr[1] = name1;  
618        arr[2] = name15;  
619        arr[3] = name134;  
620        arr[4] = name23;  
621        arr[5] = name44;  
622  
623        UrlHeap heap = new UrlHeap(arr);  
624        heap.printValues();  
625        System.out.println();  
626        heap.Build_Max_Heap();  
627        heap.printValues();  
628    }  
629  
630    public static void main(String[] args) {  
631        Url name16 = new Url("name16", 16);  
632        Url name1 = new Url("name1", 1);  
633        Url name15 = new Url("name15", 15);  
634        Url name134 = new Url("name134", 134);  
635        Url name23 = new Url("name23", 23);  
636        Url name44 = new Url("name44", 44);  
637  
638        Url[] arr = new Url[6];  
639  
640        arr[0] = name16;  
641        arr[1] = name1;  
642        arr[2] = name15;  
643        arr[3] = name134;  
644        arr[4] = name23;  
645        arr[5] = name44;  
646  
647        UrlHeap heap = new UrlHeap(arr);  
648        heap.printValues();  
649        System.out.println();  
650        heap.Build_Max_Heap();  
651        heap.printValues();  
652    }  
653  
654    public static void main(String[] args) {  
655        Url name16 = new Url("name16", 16);  
656        Url name1 = new Url("name1", 1);  
657        Url name15 = new Url("name15", 15);  
658        Url name134 = new Url("name134", 134);  
659        Url name23 = new Url("name23", 23);  
660        Url name44 = new Url("name44", 44);  
661  
662        Url[] arr = new Url[6];  
663  
664        arr[0] = name16;  
665        arr[1] = name1;  
666        arr[2] = name15;  
667        arr[3] = name134;  
668        arr[4] = name23;  
669        arr[5] = name44;  
670  
671        UrlHeap heap = new UrlHeap(arr);  
672        heap.printValues();  
673        System.out.println();  
674        heap.Build_Max_Heap();  
675        heap.printValues();  
676    }  
677  
678    public static void main(String[] args) {  
679        Url name16 = new Url("name16", 16);  
680        Url name1 = new Url("name1", 1);  
681        Url name15 = new Url("name15", 15);  
682        Url name134 = new Url("name134", 134);  
683        Url name23 = new Url("name23", 23);  
684        Url name44 = new Url("name44", 44);  
685  
686        Url[] arr = new Url[6];  
687  
688        arr[0] = name16;  
689        arr[1] = name1;  
690        arr[2] = name15;  
691        arr[3] = name134;  
692        arr[4] = name23;  
693        arr[5] = name44;  
694  
695        UrlHeap heap = new UrlHeap(arr);  
696        heap.printValues();  
697        System.out.println();  
698        heap.Build_Max_Heap();  
699        heap.printValues();  
700    }  
701  
702    public static void main(String[] args) {  
703        Url name16 = new Url("name16", 16);  
704        Url name1 = new Url("name1", 1);  
705        Url name15 = new Url("name15", 15);  
706        Url name134 = new Url("name134", 134);  
707        Url name23 = new Url("name23", 23);  
708        Url name44 = new Url("name44", 44);  
709  
710        Url[] arr = new Url[6];  
711  
712        arr[0] = name16;  
713        arr[1] = name1;  
714        arr[2] = name15;  
715        arr[3] = name134;  
716        arr[4] = name23;  
717        arr[5] = name44;  
718  
719        UrlHeap heap = new UrlHeap(arr);  
720        heap.printValues();  
721        System.out.println();  
722        heap.Build_Max_Heap();  
723        heap.printValues();  
724    }  
725  
726    public static void main(String[] args) {  
727        Url name16 = new Url("name16", 16);  
728        Url name1 = new Url("name1", 1);  
729        Url name15 = new Url("name15", 15);  
730        Url name134 = new Url("name134", 134);  
731        Url name23 = new Url("name23", 23);  
732        Url name44 = new Url("name44", 44);  
733  
734        Url[] arr = new Url[6];  
735  
736        arr[0] = name16;  
737        arr[1] = name1;  
738        arr[2] = name15;  
739        arr[3] = name134;  
740        arr[4] = name23;  
741        arr[5] = name44;  
742  
743        UrlHeap heap = new UrlHeap(arr);  
744        heap.printValues();  
745        System.out.println();  
746        heap.Build_Max_Heap();  
747        heap.printValues();  
748    }  
749  
750    public static void main(String[] args) {  
751        Url name16 = new Url("name16", 16);  
752        Url name1 = new Url("name1", 1);  
753        Url name15 = new Url("name15", 15);  
754        Url name134 = new Url("name134", 134);  
755        Url name23 = new Url("name23", 23);  
756        Url name44 = new Url("name44", 44);  
757  
758        Url[] arr = new Url[6];  
759  
760        arr[0] = name16;  
761        arr[1] = name1;  
762        arr[2] = name15;  
763        arr[3] = name134;  
764        arr[4] = name23;  
765        arr[5] = name44;  
766  
767        UrlHeap heap = new UrlHeap(arr);  
768        heap.printValues();  
769        System.out.println();  
770        heap.Build_Max_Heap();  
771        heap.printValues();  
772    }  
773  
774    public static void main(String[] args) {  
775        Url name16 = new Url("name16", 16);  
776        Url name1 = new Url("name1", 1);  
777        Url name15 = new Url("name15", 15);  
778        Url name134 = new Url("name134", 134);  
779        Url name23 = new Url("name23", 23);  
780        Url name44 = new Url("name44", 44);  
781  
782        Url[] arr = new Url[6];  
783  
784        arr[0] = name16;  
785        arr[1] = name1;  
786        arr[2] = name15;  
787        arr[3] = name134;  
788        arr[4] = name23;  
789        arr[5] = name44;  
790  
791        UrlHeap heap = new UrlHeap(arr);  
792        heap.printValues();  
793        System.out.println();  
794        heap.Build_Max_Heap();  
795        heap.printValues();  
7
```

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 pagerank_display()
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)

```
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 received: 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
LINKS 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 LINKS 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

```
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
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 received: 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

```

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
```