

Programming Assignment #2

CS146-08

Professor: Mike Wu

By: Eric Wu

Table of Contents

- I. Explanation of my Google Search Engine Simulator design and implementation details. (Page 3)
- II. A list of classes/subroutines/function calls and explanations of each purpose. (Pages 4 – 13)
- III. Screen shots of each simulation process/procedure (Pages 14 – 22)
- IV. How to get the program running (Pages 23 – 27)
 - a. How to run the program (Page 28)
- V. Problems encountered (Page 29)
- VI. Lessons learned (Page 30)

1) Explain/Illustrate your Google Search Engine Simulator design and implementation details.

I built upon the program that I have written for programming assignment 1. I have deleted the heap class but kept the pair class which keeps track of the url and score. I used the same webcrawler to get the URLs and I assigned a random score value to each of them. I added the quicksort, bucket, node, and binarysearchtree classes and modified the pagerank class so that it is now testing the above classes instead of heap. For the first part of the program, I test out the quicksort on the URLs. For this, I made a quicksort class that takes an arraylist as an argument. In the class, I have the quicksort method which calls upon the partition method so that it can sort it. I print out the results of the arraylist before sorting and then after so the user can see the difference. I then ask user if they want to search for another word. If they don't the program prints the top 10 keywords and then proceeds to ask user to test the Binary Search Tree. For the BST I created a node class so that we can have a way to create the tree data structure. Like before I ask user to input a keyword. Then I inserted the URL and corresponding scores into the binary search tree structure. I then print out the websites and scores in sorted order using the inorder treewalk method. I then test the BST's adding by inserting two more websites to the tree. I then print the websites and scores in sorted order so that the user can see that there are 2 more websites in the BST. I test the delete method on one of the website that I have added and then print out tree again to show user that the website indeed has been deleted. I then ask user to input a page rank. And I would return the url and score corresponding to that pagerank. I then ask user if they want to search for another word. If they don't the program prints the top 10 keywords (cumulative) and then proceeds to ask user to test the bucket sort. For the bucket sort, I ask user what keyword they want to put into the bucket from the list of top 10 keywords. I then use the webcrawler to get the URLs for that keyword and then I put the results into an array that hold arraylists. Then I sort it and concatenate the arrays to get a single sorted arraylist. I print out the arraylist and the user can see that the list is indeed sorted in alphabetical order.

2)A list of classes/subroutines/function calls and explanations of each purpose.

Pair Class

```
public class Pair {
    private String url;
    private int value;
    /**
     * This creates a Pair object which has a string and a value
     * @param url The URL of the pair
     * @param value The value associated with the URL
     */
    public Pair(String url, int value) {
        this.url = url; //store user inputted url into the object's url
        this.value = value; //store user inputted value into the object's value
    }
    /**
     * Gets the URL
     * @return the URL
     */
    public String getUrl() {
        return url; //return the object's url
    }
    /**
     * Gets the value
     * @return the value
     */
    public int getValue() {
        return value; //return the object's value
    }
    /**
     * Combine the URL and Value into a string
     * @return the combined URL and value string
     */
    public String toString() {
        return "Score: " + value + " URL: " + url; //return a string contraining the url and value
    }
}
```

The purpose of this pair class is so that we can store each URL with a corresponding score. I could have used a map to store the data but by making a new object to store the URL and score, I know for certain that the data would be stable.

The constructor takes in a String (which is the URL) and a integer (which is the score of the URL). I have 3 methods in this class.

One of them is **getUrl()** which returns the URL that is associated with the object.

Another one is the **getValue()** which returns the score that is associated with the object.

And finally there is the **toString()** that just returns the score and URL in one line, which just makes it easier to print out the results.

QuickSort Class

```

import java.util.*;
public class QuickSort {
    ArrayList<Pair> A;

    /**
     * Creates a quicksort object that holds an arraylist of type pair
     * @param A the array you need to provide the object
     */
    public QuickSort(ArrayList<Pair> A) {
        this.A = A;
    }

    /**
     * Perform the quicksort
     * @param A the arraylist we are sorting
     * @param p the first index that we want to sort
     * @param r the last index that we want to sort
     */
    public void quickSort(ArrayList<Pair> A, int p, int r) {
        if(p < r) {
            int q = partition(A, p, r);
            quickSort(A, p, q - 1);
            quickSort(A, q + 1, r);
        }
    }

    /**
     * the partition process of quicksort
     * @param A the arraylist we want to sort
     * @param p the first index that we want to sort
     * @param r the last index that we want to sort or the pivot
     * @return the position on the pivot
     */
    public int partition(ArrayList<Pair> A, int p, int r) {
        int x = A.get(r).getValue(); //pivot is at last index
        int i = p - 1;
        for(int j = p; j < r; j++) { //split array into 4 parts, less/equal/greater than pivot and unsorted
            if(A.get(j).getValue() <= x) {
                i++;
                Pair temp = A.get(i); //swap
                A.set(i, A.get(j));
                A.set(j, temp);
            }
        }
        Pair temp = A.get(i + 1); //swap
        A.set(i + 1, A.get(r));
        A.set(r, temp);
        return i + 1;
    }
}

```

The purpose of this class is so we can perform quicksort on a given arraylist. This class is fairly simple, it only has 2 methods.

The constructor takes in an arraylist with type Pair which holds a URL and its score.

The quicksort method takes the arraylist , first index , and last index that you want to check and recursively call itself until the arraylist is sorted.

The partition method is the part that breaks the problem into smaller size and puts the pivot and in this case the last index value into its correct place.

Node Class

```
public class Node {
    private Pair key;
    private Node left;
    private Node right;
    private Node parent;
    /**
     * creates a node object
     * @param key the url and corresponding score
     */
    public Node (Pair key){
        this.key = key;
        this.parent = null;
        this.left = null;
        this.right = null;
    }
    /**
     * sets the parent of a node
     * @param x the parent node
     */
    public void setParent(Node x) {
        this.parent = x;
    }
    /**
     * sets the left child of a node
     * @param x the left child
     */
    public void setLeft(Node x) {
        this.left = x;
    }
    /**
     * sets the right child of a node
     * @param x the right child
     */
    public void setRight(Node x) {
        this.right = x;
    }
}
```

```

/**
 * gets the parent of a node
 * @return the parent of a node
 */
public Node getParent() {
    return parent;
}

/**
 * gets the left child of a node
 * @return the left child
 */
public Node getLeft() {
    return left;
}

/**
 * gets the right child of a node
 * @return the right child
 */
public Node getRight() {
    return right;
}

/**
 * gets the key or the url and corresponding score of a node
 * @return the key of a node
 */
public Pair getKey() {
    return key;
}
}

```

The point of this class is because we need this in order to implement a BST data structure. Each node keeps track of its parents, left child, right child, and the key or the score and URL.

The constructor sets key to the argument which is of type Pair. Then it sets all its other connections to null.

The methods are self explanatory and I wrote comments on it. Basically they are getter and setter methods which we can use to modify the Node's connections and retrieve data.

BinarySearchTree Class

```

public class BinarySearchTree {
    Node root;
    private int counter = -1;
    private int counter2 = 0;
    /**
     * Creates a bst object
     */
    public BinarySearchTree() {
        root = null;
    }
    /**
     * print out the tree nodes in sorted order
     * @param x the node where we want to start at usually the root of the tree
     */
    public void inOrderTreeWalk(Node x) {
        if (x != null) {
            inOrderTreeWalk(x.getLeft());
            counter++;
            System.out.println("Index [" + counter + "] " + "Rank: " + (counter2 - counter) + " " + x.getKey().toString());
            inOrderTreeWalk(x.getRight());
        }
    }
    /**
     * resets the counter for the inordertreewalk that keeps track of index and pagerank
     */
    public void resetCounter() {
        counter = -1;
    }
    /**
     * gets the url and score of a node given a pagerank
     * @param rank the page rank of the url we want to search
     * @return the url and score
     */
    public String getUrl(int rank) {
        int rankCounter = 1;
        Node x = getMax(getRoot());
        if (rank == 1) { //rank 1 is max in tree
            return x.getKey().getUrl() + " with the score " + x.getKey().getValue();
        } else { //start from max and go to predecessor until we hit desired pagerank
            while (rankCounter != rank) {
                x = treePredecessor(x);
                rankCounter++;
            }
        }
        return x.getKey().getUrl() + " with the score " + x.getKey().getValue();
    }
}

```

This is the BST class and it creates and maintains a binary tree structure.

The constructor sets the root node to null.

The inordertreewalk prints out the values of the tree in sorted ascending order.

The resetcounter allows us to continue keep track of the counter and pagerank.

The Geturl method takes in a pagerank value and return the corresponding URL and score


```

/**
 * gets the root of the tree
 * @return the root of the tree
 */
public Node getRoot() {
    return root;
}

/**
 * get the min value of a tree could be a subtree
 * @param min the node that we want to get the min of
 * @return the min node
 */
public Node getMin(Node min) {
    Node x = min;
    while(x.getLeft() != null) {
        x = x.getLeft();
    }
    return x;
}

/**
 * get the max value of a tree could be a subtree
 * @param max the node that we want the max of
 * @return the max node
 */
public Node getMax(Node max) {
    Node x = max;
    while(x.getRight() != null) {
        x = x.getRight();
    }
    return x;
}

/**
 * gets the predecessor of a node or the next smallest one
 * @param predecessor the node we want the predecessor of
 * @return the predecessor of the given node
 */
public Node treePredecessor(Node predecessor) {
    Node x = predecessor;
    if(x.getLeft() != null) {
        return getMax(x.getLeft());
    }
    Node y = x.getParent();
    while(y != null && x == y.getLeft()) {
        x = y;
        y = y.getParent();
    }
    return y;
}

```

The getroot method simply just return the root of the tree.

The getmin method gets the minimum value of a tree or subtree

The getmax method gets the maximum value of a tree or subtree

The treePredecessor method gets the next smallest value of the given argument

```

/**
 * gets the successor of a node or the next biggest one
 * @param successor the node we want the successor of
 * @return the successor of the given node
 */
public Node treeSuccessor(Node successor) {
    Node x = successor;
    if(x.getRight() != null) {
        return getMin(x.getRight());
    }
    Node y = x.getParent();
    while(y != null && x == y.getRight()) {
        x = y;
        y = y.getParent();
    }
    return y;
}

/**
 * inserting a node to the tree and keeping its bst structure
 * @param z the node that we want to insert to the tree
 */
public void treeInsert(Node z) {
    Node y = null;
    Node x = root;
    while(x != null) {
        y = x;
        if(z.getKey().getValue() < x.getKey().getValue()) {
            x = x.getLeft();
        } else {
            x = x.getRight();
        }
    }
    z.setParent(y);
    if(y == null) { //case 1
        root = z;
    }
    else if(z.getKey().getValue() < y.getKey().getValue()) { //case 2
        y.setLeft(z);
    } else { //case 3
        y.setRight(z);
    }
    counter2++; //keep track of the counter
}

```

The treesuccessor method is much like the treepredecessor method but it now gets the next biggest value instead of the smallest

The treeinsert method inserts a node into the BST and the tree keeps its BST properties and structure. Notice that I have also increased the counter to keep track of the index and total amount.

```

/**
 * the transplant part of the delete method
 * @param u the node we want to replace
 * @param v the node we want to transplant
 */
public void transplant(Node u, Node v) {
    if (u.getParent() == null) {
        root = v;
    } else if (u == u.getParent().getLeft()) {
        u.getParent().setLeft(v);
    } else {
        u.getParent().setRight(v);
    }
    if (v != null) {
        v.setParent(u.getParent());
    }
}

/**
 * deleting a node from the bst
 * @param z the node we want to delete
 */
public void treeDelete(Node z) {
    if (z.getLeft() == null) { //case 1
        transplant(z, z.getRight());
    } else if (z.getRight() == null) { //case 2
        transplant(z, z.getLeft());
    } else { //case 3
        Node y = getMin(z.getRight());
        if (y.getParent() != z) {
            transplant(y, y.getRight());
            y.setRight(z.getRight());
            y.getRight().setParent(y);
        }
        transplant(z, y);
        y.setLeft(z.getLeft());
        y.getLeft().setParent(y);
    }
    counter2--; //keep track of the counter
}
}

```

The transplant method is part of the treeDelete method. It basically completely replaces a node with another node.

The treedelete method deletes a given node. It has three cases it can go through, no child, one child and 2 children. Notice that I have also decreased the counter to keep track of the index and total amount.

Bucket Class

```

import java.util.*;
public class bucket {
    private ArrayList<String> list;
    private ArrayList<String>[] bucket;
    private ArrayList<String> result;

    /**
     * creates an bucket object that store an arraylist
     * @param A the arraylist we need to provide the object
     */
    public bucket(ArrayList<String> A) {
        this.list = A;
        bucket = new ArrayList<String>[list.size()];
        result = new ArrayList<>();
    }

    /**
     * sorts the arraylist
     */
    public void bucketSort(){
        int n = list.size();
        for(int i = 0; i < n; i++) { //create an array that stores arraylists
            bucket[i] = new ArrayList<String>();
        }
        for(int i = 0; i < n; i++) {
            if(list.get(i).substring(5, 6).equals(":")) { //three cases
                char loc = list.get(i).charAt(8); //if website is https: we ignore that part
                int location = (int)loc - 97;
                bucket[location].add(list.get(i).substring(8, list.get(i).length()));
            } else if(list.get(i).substring(5, 6).equals("/")) {
                char loc = list.get(i).charAt(7); //if website is http: we ignore that part
                int location = (int)loc - 97;
                bucket[location].add(list.get(i).substring(7, list.get(i).length()));
            } else { //if website doesn't have http(s): we include the whole url
                char loc = list.get(i).charAt(0);
                int location = (int)loc - 97; //a has ascii value of 97, so index 0 is a etc...
                bucket[location].add(list.get(i)); //add it to array
            }
        }
        for(int i = 0; i < n; i++) {
            if(bucket[i] != null) { //if bucket is not null we sort it
                Collections.sort(bucket[i]);
            }
        }
        for(int i = 0; i < n; i++) { //concatenate the buckets into an arraylist
            if(bucket[i] != null) {
                for(int j = 0; j < bucket[i].size(); j++) {
                    result.add(bucket[i].get(j));
                }
            }
        }
    }

    /**
     * gets the arraylist that has the sorted concatenated results
     * @return the sorted arraylist
     */
    public ArrayList<String> getResult() {
        return result;
    }
}

```

This bucket class is used to perform bucketsort on a list of URLs so that it alphabetically sorted via domain name.

The constructor takes the array that you want to sort as an argument.

The bucketsort method is separated into 4 sections. The first part creates a new array and initializes each element with an arraylist. The next part adds the website url into the arraylists and in increasing order. Ex: a website that starts with a, goes into the first index, etc. And I also don't add the whole URL to make sorting more obvious. So I just removed the http:// or https:// from the beginning if the url has it and added them to the corresponding array's arraylist. The next part, I check if the array is null, if it is not, we sort the arraylist in that array. Then finally I concatenate the results into one arraylist.

The getresult method simply returns the sorted arraylist of URLs

3) Provide a lot of screen shots of each simulation process/procedure including inputs and outputs for each of function listed in item 6 associated with the two major features listed in the end of the Problem Statements. This is to verify your codes and to check to see if your codes match the functional requirements and run correctly by yourself.

PageRank Class

```

/**
 * This class implements and uses the pair, Node, BST, bucket, and spider classes.
 * @author wueric
 */
import java.util.*;

public class Pagerank {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        boolean done = false;
        LinkedHashSet<String> list = new LinkedHashSet<>();
        while(!done) {
            System.out.print("What keyword do you want to search?: ");
            String keyWord = scan.next();
            list.add(keyWord); //storing unique searches
            Spider crawler = new Spider(); //new spider object
            crawler.search("https://arstechnica.com/", keyWord); //website we are getting URLs from, and keyword is science
            ArrayList<String> array = crawler.getPageList(); //set the string arraylist equal to the crawler's list of URLs
            ArrayList<Pair> A = new ArrayList<>();

            for(int i = 0; i < array.size(); i++) {
                Pair pair = new Pair(array.get(i), makeRandomScore()); //assign each url a random score
                A.add(pair); //store each url and random score into the arraylist
            }
            System.out.println("URLs we have collected and their scores (UNSORTED)");
            Quicksort quicksort = new Quicksort(A); //Making a new quicksort object with the A arraylist as the argument
            for(int i = 0; i < A.size(); i++) {
                System.out.println("Index [" + i + "] " + A.get(i).toString()); //prints the URLs unmodified
            }
            System.out.println();
            System.out.println("URLs after QuickSort:");
            quicksort.quickSort(A, 0, A.size() - 1); //perform quicksort
            int rankcounter = A.size();
            for(int i = 0; i < A.size(); i++) {
                System.out.println("Index [" + i + "] " + "Rank: " + rankcounter + " " + A.get(i).toString()); //prints the URLs
                rankcounter--;
            }
            System.out.println();

            System.out.print("Do you want to search for any other word? (Y/N): ");
            String response = scan.next().toUpperCase();
            if(response.equals("Y")) {
                done = false;
            } else {
                done = true;
            }
        }
    }
}

```

```

System.out.println("The Top 10 Unique Searches:");
Iterator it = list.iterator();
int counter = 0;
while(it.hasNext() && counter < 10) {
    System.out.println(it.next());
}
System.out.println();
System.out.println("Testing Binary Search Tree:");
done = false;
while(!done) {
    System.out.print("What keyword do you want to search?: ");
    String keyWord = scan.next();
    list.add(keyWord);
    Spider crawler = new Spider(); //new spider object
    crawler.search("https://arstechnica.com/", keyWord); //website we are getting URLs from, and keyword is science
    ArrayList<String> array = crawler.getPageList(); //set the string arraylist equal to the crawler's list of URLs
    BinarySearchTree bst = new BinarySearchTree(); //create a bst object

    for(int i = 0; i < array.size(); i++) {
        Pair pair = new Pair(array.get(i), makeRandomScore()); //assign each url a random score
        Node node = new Node(pair); //node has value of url and score
        bst.treeInsert(node); //store each node into the tree
    }
    System.out.println("Binary Search Tree in order tree walk");
    bst.inOrderTreeWalk(bst.getRoot());
    bst.resetCounter();
    System.out.println();
    System.out.println("Adding two websites to the BST");
    Pair website = new Pair("MikeWu.com", 88888888); //website and score 1
    Pair website2 = new Pair("cs.com", 100); //website and score 2
    Node mikeWu = new Node(website);
    Node cs = new Node(website2);
    bst.treeInsert(mikeWu); //inserting the two websites
    bst.treeInsert(cs);

    System.out.println("Binary Search Tree in order tree walk with the added URLs");
    bst.inOrderTreeWalk(bst.getRoot());
    bst.resetCounter();
    System.out.println();

    System.out.println("Deleting cs.com from tree");
    bst.treeDelete(cs);

    System.out.println("Binary Search Tree in order tree walk after deleted URL");
    bst.inOrderTreeWalk(bst.getRoot());
    bst.resetCounter();
    System.out.println();

```

```

System.out.print("Enter a pagerank to search for the URL: ");
int rankToCheck = scan.nextInt();
scan.nextLine();
System.out.println("The score and Url with pagerank " + rankToCheck + " is " + bst.getUrl(rankToCheck));
System.out.println();

System.out.print("Do you want to search for any other word? (Y/N): ");
String response = scan.nextLine().toUpperCase();
if(response.equals("Y")) {
    done = false;
} else {
    done = true;
}
}
System.out.println();
System.out.println("The Top 10 Unique Searches:");
counter = 0;
it = list.iterator();
while(it.hasNext() && counter < 10) {
    System.out.println(it.next());
} System.out.println();
System.out.print("Which of the popular keyword would you like to store in a bucket? ");
String answer = scan.nextLine();
Spider crawler = new Spider(); //new spider object
crawler.search("https://arstechnica.com/", answer); //website we are getting URLs from, and keyword is science
ArrayList<String> array = crawler.getPageList(); //set the string arraylist equal to the crawler's list of URLs
Bucket bucket = new Bucket(array); //create a bucket object
bucket.bucketSort(); //sort the bucket with bucket sort
ArrayList<String> result = new ArrayList<>();
result = bucket.getResult();
Iterator it2 = result.iterator();
System.out.println();
System.out.println("The URL's sorted alphabetically via bucketsort");
while(it2.hasNext()) {
    System.out.println(it2.next());
}
}

public static int makeRandomScore() { //creates a random score based on the 4 criterias and return it
    Random random = new Random();
    int randomFrequency = random.nextInt(100) + 1; //random frequency score of the keyword
    int randomHistory = random.nextInt(100) + 1; //random time score of how long the website has been created
    int randomRelevancy = random.nextInt(100) + 1; //random relevancy score of the website
    int randomAdvertisement = random.nextInt(100) + 1; //random score on how much website paid google
    int totalScore = randomFrequency + randomHistory + randomRelevancy + randomAdvertisement; //add up the 4 scores
    return totalScore; //return the total score
}
}

```

First I will explain how the WebCrawler works. The WebCrawler application I am using also uses jsoup and it has two classes, a Spider class and a SpiderLeg class. The spider class basically uses the spiderleg class to “crawl” on a website, trying to find if it matches with the keyword we provide. The spider class takes two arguments to begin, a website url and a keyword string from which it is going to search the website for. The spider class has three instance variables, one called MAX_PAGES_TO_SEARCH which I set to 30 because we are looking for 30 websites. The next variable is pagesVisted which is a stored in a set. And finally the last variable is called pagesToVisit which keeps track of the remaining pages needed to check. It is stored with a list. This program gets 30 URLs by navigating from the main page. It then looks around for any links on the website, and it adds them to pagesToVisit. Then it traverse down the list of pagesToVisit and if check if the URL page contains the keyword and if it does, it tells us by printing out it found it. Then it adds the URL to the pagesVisited set. In this case, I used the made a spider

object with the URL as <https://arstechnica.com/> and user input as the keyword. I made this program repeat asking user for keywords until they type N and they will exit out of the program. After that, the program will print out the top 10 unique keywords that the user have inputted. The program starts by testing quicksort. After that it will go to test binary search tree, then bucketsort. To make the program seem less clogged up by text, I have commented out the print statements from the webcrawler.

The following is the result from running the program with further explanation.

```
run:
What keyword do you want to search?: apple
URLs we have collected and their scores (UNSORTED)
Index [0] Score: 299 URL: http://arstechnica.com/?theme=dark
Index [1] Score: 239 URL: https://arstechnica.com/search/
Index [2] Score: 310 URL: http://arstechnica.com/?view=archive
Index [3] Score: 181 URL: https://arstechnica.com/tag/ars-approved/
Index [4] Score: 177 URL: https://arstechnica.com/reviews/
Index [5] Score: 213 URL: http://arstechnica.com/?theme=light
Index [6] Score: 182 URL: https://arstechnica.com/about-us/
Index [7] Score: 187 URL: https://arstechnica.com/staff-directory/
Index [8] Score: 141 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index [9] Score: 249 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index [10] Score: 270 URL: https://arstechnica.com
Index [11] Score: 23 URL: https://arstechnica.com/rss-feeds/
Index [12] Score: 132 URL: https://arstechnica.com/science/
Index [13] Score: 306 URL: https://arstechnica.com/advertise-with-us/
Index [14] Score: 218 URL: https://arstechnica.com/?view=mobile
Index [15] Score: 139 URL: https://arstechnica.com/contact-us/
Index [16] Score: 107 URL: https://arstechnica.com/
Index [17] Score: 98 URL: http://arstechnica.com/?view=grid
Index [18] Score: 246 URL: https://arstechnica.com/tech-policy/
Index [19] Score: 215 URL: https://arstechnica.com/gadgets/
Index [20] Score: 232 URL: https://arstechnica.com/civis/
Index [21] Score: 157 URL: https://arstechnica.com/#site-menu
Index [22] Score: 259 URL: https://arstechnica.com/gaming/
Index [23] Score: 235 URL: https://arstechnica.com/store/product/subscriptions/
Index [24] Score: 160 URL: https://arstechnica.com/features/
Index [25] Score: 224 URL: https://arstechnica.com/information-technology/
Index [26] Score: 208 URL: https://arstechnica.com/reprints/
Index [27] Score: 215 URL: https://arstechnica.com/cars/
Index [28] Score: 99 URL: https://arstechnica.com/#main
Index [29] Score: 63 URL: http://video.arstechnica.com/
```

This is the result from using the WebCrawler, we have gone through the 30 URLs.

Notice that the program ask the user what keyword they want to search for and in this case the user entered apple as the keyword

```

URLs after QuickSort:
Index [0] Rank: 30 Score: 23 URL: https://arstechnica.com/rss-feeds/
Index [1] Rank: 29 Score: 63 URL: http://video.arstechnica.com/
Index [2] Rank: 28 Score: 98 URL: http://arstechnica.com/?view=grid
Index [3] Rank: 27 Score: 99 URL: https://arstechnica.com/#main
Index [4] Rank: 26 Score: 107 URL: https://arstechnica.com/
Index [5] Rank: 25 Score: 132 URL: https://arstechnica.com/science/
Index [6] Rank: 24 Score: 139 URL: https://arstechnica.com/contact-us/
Index [7] Rank: 23 Score: 141 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index [8] Rank: 22 Score: 157 URL: https://arstechnica.com/#site-menu
Index [9] Rank: 21 Score: 160 URL: https://arstechnica.com/features/
Index [10] Rank: 20 Score: 177 URL: https://arstechnica.com/reviews/
Index [11] Rank: 19 Score: 181 URL: https://arstechnica.com/tag/ars-approved/
Index [12] Rank: 18 Score: 182 URL: https://arstechnica.com/about-us/
Index [13] Rank: 17 Score: 187 URL: https://arstechnica.com/staff-directory/
Index [14] Rank: 16 Score: 208 URL: https://arstechnica.com/reprints/
Index [15] Rank: 15 Score: 213 URL: http://arstechnica.com/?theme=light
Index [16] Rank: 14 Score: 215 URL: https://arstechnica.com/cars/
Index [17] Rank: 13 Score: 215 URL: https://arstechnica.com/gadgets/
Index [18] Rank: 12 Score: 218 URL: https://arstechnica.com/?view=mobile
Index [19] Rank: 11 Score: 224 URL: https://arstechnica.com/information-technology/
Index [20] Rank: 10 Score: 232 URL: https://arstechnica.com/civis/
Index [21] Rank: 9 Score: 235 URL: https://arstechnica.com/store/product/subscriptions/
Index [22] Rank: 8 Score: 239 URL: https://arstechnica.com/search/
Index [23] Rank: 7 Score: 246 URL: https://arstechnica.com/tech-policy/
Index [24] Rank: 6 Score: 249 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index [25] Rank: 5 Score: 259 URL: https://arstechnica.com/gaming/
Index [26] Rank: 4 Score: 270 URL: https://arstechnica.com
Index [27] Rank: 3 Score: 299 URL: http://arstechnica.com/?theme=dark
Index [28] Rank: 2 Score: 306 URL: https://arstechnica.com/advertise-with-us/
Index [29] Rank: 1 Score: 310 URL: http://arstechnica.com/?view=archive

```

Displays the results after quicksort has been done. The list of URLs are now in ascending order based on the score that the URL have. Also note that each URL has a corresponding score, rank and index.

```

Do you want to search for any other word? (Y/N): N
The Top 10 Unique Searches:
apple

```

We ask user if they want to search for any other word. If they type Y, we continue. If they type N, we move on to the next part

As we can see, this search keyword (apple) displays the top 10 as this. We print out the top 10 unique searches. Since the user only input 1 unique keywords, we only have 1 unique keywords to show for it which is apple.

```

Testing Binary Search Tree:
What keyword do you want to search?: tree
Binary Search Tree in order tree walk
Index [0] Rank: 30 Score: 72 URL: https://arstechnica.com/reviews/
Index [1] Rank: 29 Score: 75 URL: https://arstechnica.com/#site-menu
Index [2] Rank: 28 Score: 107 URL: https://arstechnica.com/advertise-with-us/
Index [3] Rank: 27 Score: 121 URL: https://arstechnica.com/tag/ars-approved/
Index [4] Rank: 26 Score: 135 URL: https://arstechnica.com/gadgets/
Index [5] Rank: 25 Score: 137 URL: https://arstechnica.com/information-technology/
Index [6] Rank: 24 Score: 143 URL: https://arstechnica.com/rss-feeds/
Index [7] Rank: 23 Score: 145 URL: https://arstechnica.com/search/
Index [8] Rank: 22 Score: 157 URL: https://arstechnica.com/?view=mobile
Index [9] Rank: 21 Score: 157 URL: http://arstechnica.com/?view=grid
Index [10] Rank: 20 Score: 164 URL: http://arstechnica.com/?view=archive
Index [11] Rank: 19 Score: 165 URL: https://arstechnica.com/features/
Index [12] Rank: 18 Score: 184 URL: https://arstechnica.com/store/product/subscriptions/
Index [13] Rank: 17 Score: 185 URL: https://arstechnica.com
Index [14] Rank: 16 Score: 196 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index [15] Rank: 15 Score: 198 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index [16] Rank: 14 Score: 203 URL: https://arstechnica.com/reprints/
Index [17] Rank: 13 Score: 228 URL: http://arstechnica.com/?theme=light
Index [18] Rank: 12 Score: 239 URL: https://arstechnica.com/gaming/
Index [19] Rank: 11 Score: 239 URL: https://arstechnica.com/#main
Index [20] Rank: 10 Score: 250 URL: https://arstechnica.com/
Index [21] Rank: 9 Score: 264 URL: https://arstechnica.com/contact-us/
Index [22] Rank: 8 Score: 265 URL: https://arstechnica.com/civis/
Index [23] Rank: 7 Score: 267 URL: https://arstechnica.com/tech-policy/
Index [24] Rank: 6 Score: 270 URL: https://arstechnica.com/cars/
Index [25] Rank: 5 Score: 282 URL: http://video.arstechnica.com/
Index [26] Rank: 4 Score: 291 URL: https://arstechnica.com/science/
Index [27] Rank: 3 Score: 294 URL: https://arstechnica.com/staff-directory/
Index [28] Rank: 2 Score: 307 URL: http://arstechnica.com/?theme=dark
Index [29] Rank: 1 Score: 324 URL: https://arstechnica.com/about-us/

```

We are now testing the binary search tree and the user has input tree as the keyword.

Then we print out the results in ascending order where the lowest score comes first. Also note that we can see the index, score, and rank for each URL.

```

Adding two websites to the BST
Binary Search Tree in order tree walk with the added URLs
Index [0] Rank: 32 Score: 72 URL: https://arstechnica.com/reviews/
Index [1] Rank: 31 Score: 75 URL: https://arstechnica.com/#site-menu
Index [2] Rank: 30 Score: 100 URL: cs.com
Index [3] Rank: 29 Score: 107 URL: https://arstechnica.com/advertise-with-us/
Index [4] Rank: 28 Score: 121 URL: https://arstechnica.com/tag/ars-approved/
Index [5] Rank: 27 Score: 135 URL: https://arstechnica.com/gadgets/
Index [6] Rank: 26 Score: 137 URL: https://arstechnica.com/information-technology/
Index [7] Rank: 25 Score: 143 URL: https://arstechnica.com/rss-feeds/
Index [8] Rank: 24 Score: 145 URL: https://arstechnica.com/search/
Index [9] Rank: 23 Score: 157 URL: https://arstechnica.com/?view=mobile
Index [10] Rank: 22 Score: 157 URL: http://arstechnica.com/?view=grid
Index [11] Rank: 21 Score: 164 URL: http://arstechnica.com/?view=archive
Index [12] Rank: 20 Score: 165 URL: https://arstechnica.com/features/
Index [13] Rank: 19 Score: 184 URL: https://arstechnica.com/store/product/subscriptions/
Index [14] Rank: 18 Score: 185 URL: https://arstechnica.com
Index [15] Rank: 17 Score: 196 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index [16] Rank: 16 Score: 198 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index [17] Rank: 15 Score: 203 URL: https://arstechnica.com/reprints/
Index [18] Rank: 14 Score: 228 URL: http://arstechnica.com/?theme=light
Index [19] Rank: 13 Score: 239 URL: https://arstechnica.com/gaming/
Index [20] Rank: 12 Score: 239 URL: https://arstechnica.com/#main
Index [21] Rank: 11 Score: 250 URL: https://arstechnica.com/
Index [22] Rank: 10 Score: 264 URL: https://arstechnica.com/contact-us/
Index [23] Rank: 9 Score: 265 URL: https://arstechnica.com/civis/
Index [24] Rank: 8 Score: 267 URL: https://arstechnica.com/tech-policy/
Index [25] Rank: 7 Score: 270 URL: https://arstechnica.com/cars/
Index [26] Rank: 6 Score: 282 URL: http://video.arstechnica.com/
Index [27] Rank: 5 Score: 291 URL: https://arstechnica.com/science/
Index [28] Rank: 4 Score: 294 URL: https://arstechnica.com/staff-directory/
Index [29] Rank: 3 Score: 307 URL: http://arstechnica.com/?theme=dark
Index [30] Rank: 2 Score: 324 URL: https://arstechnica.com/about-us/
Index [31] Rank: 1 Score: 88888888 URL: MikeWu.com

```

We are testing the insert method of the binary search tree. In this case I added two websites, MikeWu.com with the score 88888888 and cs.com with the score 100. They are also in the proper place with their respective score, counter, and rank.

```

Deleting cs.com from tree
Binary Search Tree in order tree walk after deleted URL
Index [0] Rank: 31 Score: 72 URL: https://arstechnica.com/reviews/
Index [1] Rank: 30 Score: 75 URL: https://arstechnica.com/#site-menu
Index [2] Rank: 29 Score: 107 URL: https://arstechnica.com/advertise-with-us/
Index [3] Rank: 28 Score: 121 URL: https://arstechnica.com/tag/ars-approved/
Index [4] Rank: 27 Score: 135 URL: https://arstechnica.com/gadgets/
Index [5] Rank: 26 Score: 137 URL: https://arstechnica.com/information-technology/
Index [6] Rank: 25 Score: 143 URL: https://arstechnica.com/rss-feeds/
Index [7] Rank: 24 Score: 145 URL: https://arstechnica.com/search/
Index [8] Rank: 23 Score: 157 URL: https://arstechnica.com/?view=mobile
Index [9] Rank: 22 Score: 157 URL: http://arstechnica.com/?view=grid
Index [10] Rank: 21 Score: 164 URL: http://arstechnica.com/?view=archive
Index [11] Rank: 20 Score: 165 URL: https://arstechnica.com/features/
Index [12] Rank: 19 Score: 184 URL: https://arstechnica.com/store/product/subscriptions/
Index [13] Rank: 18 Score: 185 URL: https://arstechnica.com
Index [14] Rank: 17 Score: 196 URL: https://arstechnica.com/civis/ucp.php?mode=sendpassword
Index [15] Rank: 16 Score: 198 URL: https://arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
Index [16] Rank: 15 Score: 203 URL: https://arstechnica.com/reprints/
Index [17] Rank: 14 Score: 228 URL: http://arstechnica.com/?theme=light
Index [18] Rank: 13 Score: 239 URL: https://arstechnica.com/gaming/
Index [19] Rank: 12 Score: 239 URL: https://arstechnica.com/#main
Index [20] Rank: 11 Score: 250 URL: https://arstechnica.com/
Index [21] Rank: 10 Score: 264 URL: https://arstechnica.com/contact-us/
Index [22] Rank: 9 Score: 265 URL: https://arstechnica.com/civis/
Index [23] Rank: 8 Score: 267 URL: https://arstechnica.com/tech-policy/
Index [24] Rank: 7 Score: 270 URL: https://arstechnica.com/cars/
Index [25] Rank: 6 Score: 282 URL: http://video.arstechnica.com/
Index [26] Rank: 5 Score: 291 URL: https://arstechnica.com/science/
Index [27] Rank: 4 Score: 294 URL: https://arstechnica.com/staff-directory/
Index [28] Rank: 3 Score: 307 URL: http://arstechnica.com/?theme=dark
Index [29] Rank: 2 Score: 324 URL: https://arstechnica.com/about-us/
Index [30] Rank: 1 Score: 88888888 URL: MikeWu.com

```

We are testing the delete function of the binary search tree and we are deleting cs.com in this case. If you take a look above, we can see that cs.com has been removed from the tree.

```

Enter a pagerank to search for the URL: 11
The score and Url with pagerank 11 is https://arstechnica.com/ with the score 250

Do you want to search for any other word? (Y/N): N

The Top 10 Unique Searches:
apple
tree

```

Here we are asking user to input a pagerank and the program will return its corresponding score and URL. We can double check the result with the picture above and indeed we get the right results. I then ask user if they want to search for another word. Then I print out the top 10 unique searches. We can see that the results are cumulative, it has the search word from the quicksort also.

Which of the popular keyword would you like to store in a bucket? **apple**

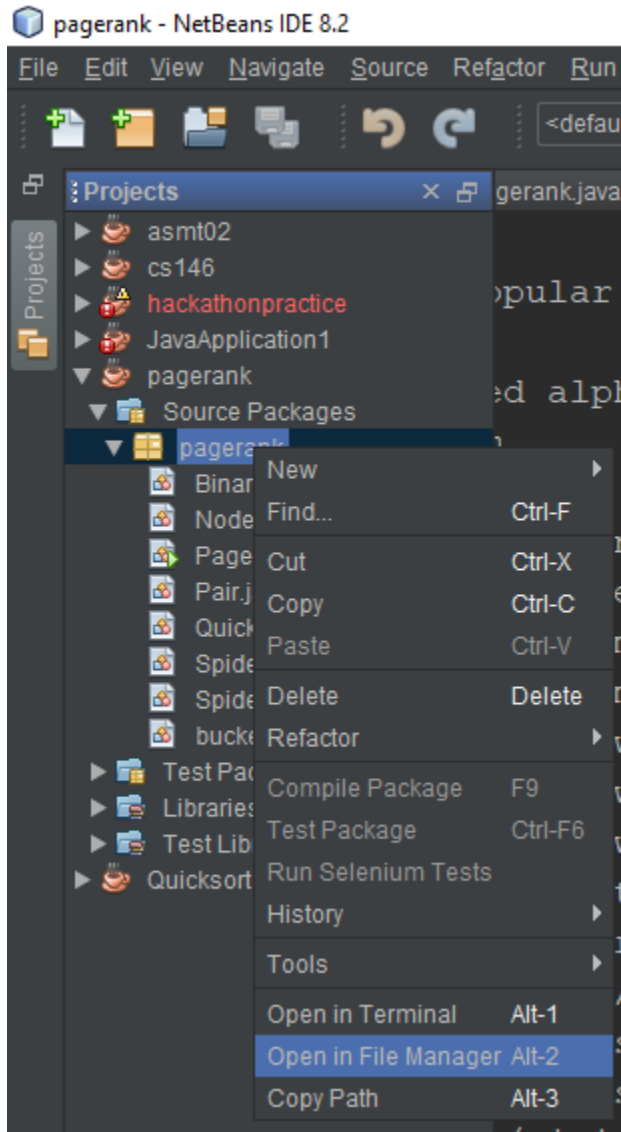
The URL's sorted alphabetically via bucketsort

```
arstechnica.com
arstechnica.com/
arstechnica.com/#main
arstechnica.com/#site-menu
arstechnica.com/?theme=dark
arstechnica.com/?theme=light
arstechnica.com/?view=archive
arstechnica.com/?view=grid
arstechnica.com/?view=mobile
arstechnica.com/about-us/|
arstechnica.com/advertise-with-us/
arstechnica.com/cars/
arstechnica.com/civis/
arstechnica.com/civis/ucp.php?mode=login&return_to=%2F
arstechnica.com/civis/ucp.php?mode=sendpassword
arstechnica.com/contact-us/
arstechnica.com/features/
arstechnica.com/gadgets/
arstechnica.com/gaming/
arstechnica.com/information-technology/
arstechnica.com/reprints/
arstechnica.com/reviews/
arstechnica.com/rss-feeds/
arstechnica.com/science/
arstechnica.com/search/
arstechnica.com/staff-directory/
arstechnica.com/store/product/subscriptions/
arstechnica.com/tag/ars-approved/
arstechnica.com/tech-policy/
video.arstechnica.com/
BUILD SUCCESSFUL (total time: 2 minutes 44 seconds)
```

Finally, the program asks user what word from the top 10 searches that they want to sort with a bucketsort. The user input apple, and the results are what the searchword returned. Notice that the sorted URLs are sorted alphabetically.









4)The procedure (step by step) of how to unzip your files, install your application, and run/test your codes.

I am using netbeans to write my code. First begin by unzipping the files. Drag out everything from the folder and drag it into your desktop. Then open up your IDE's file path.

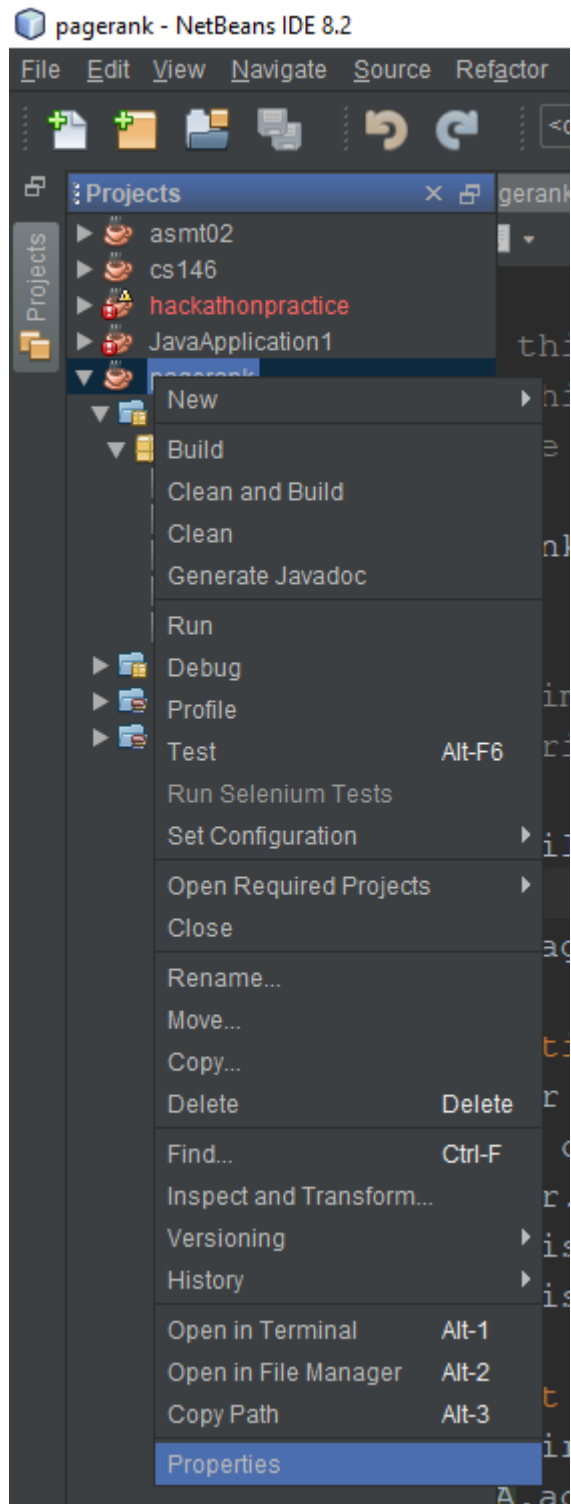


Then drag everything from the code folder into the file path.

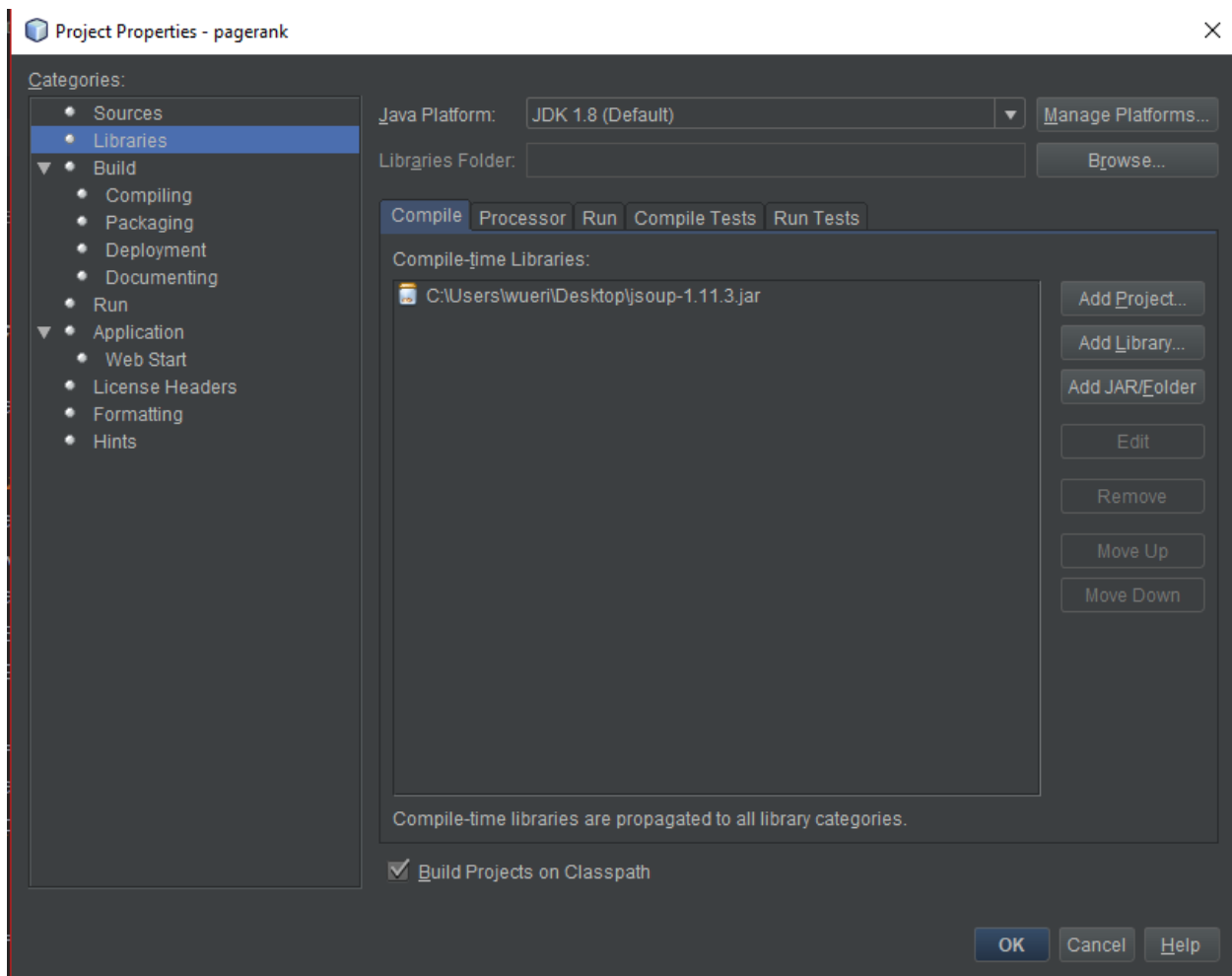
This PC > OS (C:) > Users > wueri > Documents > NetBeansProjects > pagerank > src > pagerank

Name ^		Date modified	Type	Size
	BinarySearchTree.java	11/16/2018 10:21 PM	JAVA File	6 KB
	bucket.java	11/16/2018 7:58 PM	JAVA File	3 KB
	Node.java	11/16/2018 7:58 PM	JAVA File	2 KB
	Pagerank.java	11/16/2018 10:21 PM	JAVA File	8 KB
	Pair.java	11/16/2018 8:04 PM	JAVA File	2 KB
	Quicksort.java	11/16/2018 7:58 PM	JAVA File	2 KB
	Spider.java	11/16/2018 10:21 PM	JAVA File	3 KB
	SpiderLeg.java	11/16/2018 6:12 PM	JAVA File	4 KB

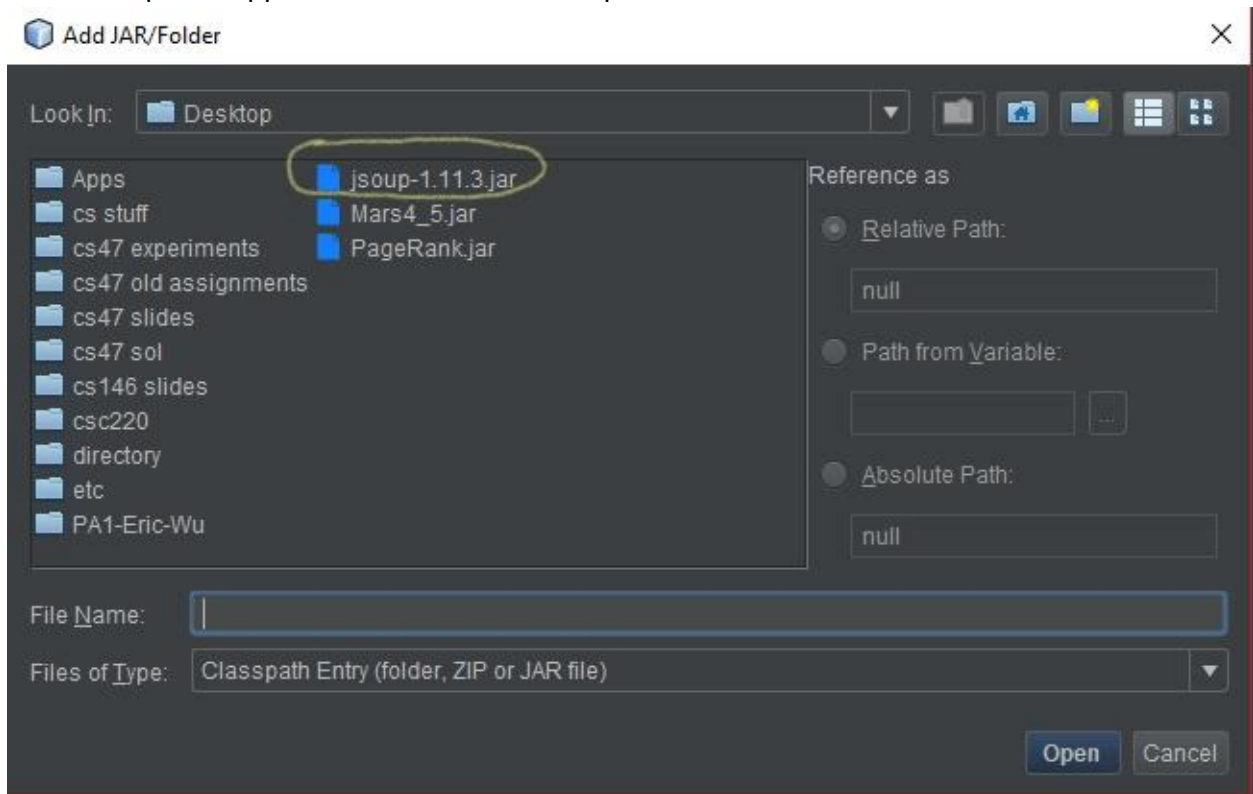
Right click on the project and click on properties.



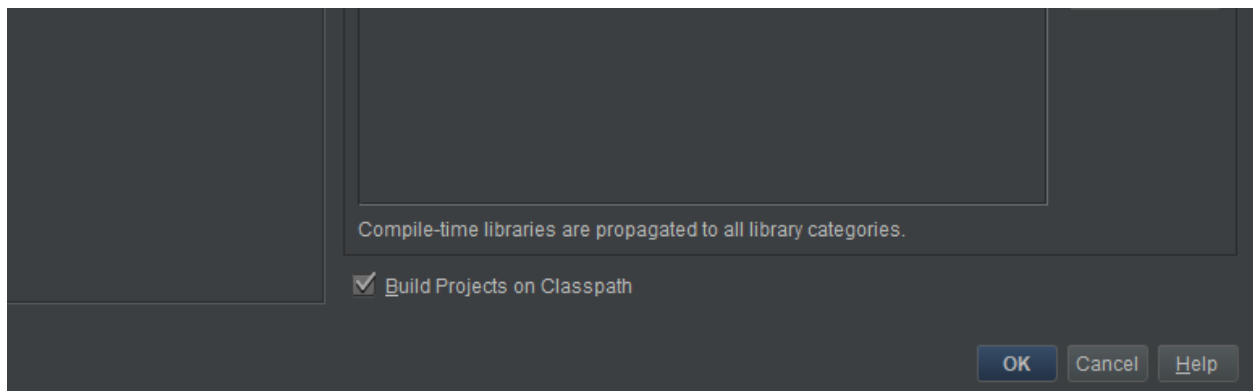
Click on libraries on the left hand side and then click on the Add JAR/Folder on the right side.



Select the jsoup-1.11.3.jar file that you have unzipped along with the code. It can be found in the desktop if unzipped the files to the desktop.



Then click on the Ok button.



You are done with setting up the program! Now proceed to run the main class, Pagerank.java.

To Run the Jar file make sure it is in you desktop.

Then open command prompt and type

```
cd desktop
java -jar PageRank.jar
```

Overview on how to use the program

The program begins by asking user to input a keyword to test the quicksort.

```
run:
What keyword do you want to search?: apple
```

The program then ask you if you want to search for any other word. If you type Y, it will repeat the steps over again, but if you type N, the program will move on to the next part.

```
Do you want to search for any other word? (Y/N): N
```

We are now testing the binary search tree, enter a keyword again.

```
Testing Binary Search Tree:
What keyword do you want to search?: tree
```

We are now asked what pagerank we want to check, enter a rank that is within range.

```
Enter a pagerank to search for the URL: 11
The score and Url with pagerank 11 is https://arstechnica.com/ with the score 250
```

The program then ask you if you want to search for any other word. If you type Y, it will repeat the steps over again, but if you type N, the program will move on to the next part.

```
Do you want to search for any other word? (Y/N): N
```

We are now asked what keyword we want to bucket sort. Enter a keyword from the list provided above

```
The Top 10 Unique Searches:
apple
tree

Which of the popular keyword would you like to store in a bucket? apple
```

5) Problems encountered during the implementation

I have still encountered many problems while trying to write this program even though I was building on old code. The first problem I encountered is the out of bounds error from the quicksort. Because I did not implement the quicksort pseudocode correctly, I had to meticulously go through the code in order to find out what I did wrong. I am also proud to say that the debugger was my best friend while doing the programming assignment. Another problem I had was null pointer errors. I had no idea what was wrong since I had not work with nodes in quite a while so I had to bring out my old textbook and read that until I got how to do it. I also had to refer back to my professor's slides and book many times before I got the correct java code down. Another problem I had was the way I implemented the user interface. I used the code I wrote before for programming assignment 1 and it doesn't translate over perfectly. I had to remove many lines and add many so if I started blank, I might have finished the program even faster. It was also hard for me to understand what some part of my old code did because of pointless variable names and lack of detailed comments. I have encountered many problems while trying to finish the programming assignment but I hope that I can learn from the mistakes I have made and use that experience and apply it to other projects I will be working on in the future.

6) Lessons Learned

I have learned many things from doing this programming assignment. Again it would be time management. I have procrastinated on completing this assignment and that turned out to be a bad idea. For the past few days, I have been getting little sleep because I am trying to finish the project and also juggle with work from other classes. The next time I am given an assignment, I will start it as soon as possible, so if I have any major bugs with the program I will have time to fix it and also ask my professor for help. I would also be less stressed out because I should be doing a decent amount of work on the assignment each day over the course of several week and not all in several days. Having dislocated my shoulder and not being able to type with two hands also made this really difficult for me. I also learned the importance of writing good comments and using relevant variable names so that I can better understand the code the next time I look at it. Another thing I learned was that I need to practice my programming skills and concepts. I don't usually review concepts once I learned them so I usually forget how to do them. For example when making the tree I was lost on the node part, so reviewing old concepts is really important in being a successful student and programmer. I should also work on projects alongside with classes so that I could always keep my programming knowledge sharp.