Design Document of Google Search Engine Simulator

Name: Yuehongxiao Ma

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

There is a total of 5 classes in the simulator program.

The main implementation is in the class Main and Simulator.

Class Main:

Method Main():

The main method here is the start of the whole program. It prints out a menu to tell the user what to enter to go to the next step of the program.

It uses the Scanner to take the input from the user. And it also creates a simulator object to implements the real simulator functions.

There is a boolean variable start with true used with a while loop for printing the main menu of the simulator. The while loop is to control the action “quit” to quit the program. A switch statement is inside of the while loop to call a different function that the user input.

If the user enters “S”, the search() method inside of the simulator class will be called to start the search function.

If the user enters “C”, the changeScore() method inside of the simulator class will be called to allow user change website score.

If the user enters “P”, the printTop10Keyword() method will be called, the top 10 keywords will be print.

If the user enters “Q”, the boolean variable will change to be false and the while loop will stop, the program will stop.

If the user enters other input, wrong input message will be print out and the menu will be printed again to ask for a new input.

Class Simulator:

The class firstly have 4 objects, an ArrayList for URL objects to store the searching result URL, an ArrayList for URL objects that store the URL whose score was changed by the user, and HashMap for String key and Integer value to store the searching times of every keyword the user entered, and a Scanner to take user input.

Method search():

The method collects all the partial functions to make a search completed.

It initializes an ArrayList to store the URL objects for the searching results every time when running the method to keep the ArrayList only have the searching result of one keyword.

Then it asks the user to enter the keyword that the user wants to search.

The keyword will be saved to the HashMap by calling the method storeKeyword().

Then a WebCrawler object will be created with the keyword that the user just entered and call its search method to get the search result. The search result will be then put into the ArrayList for sorting. When putting the URL into the ArrayList, for the requirement of the output, the index of the URL object before sorting will also be set at the same time.

Variation: when adding the URL object to the ArrayList, the URL object will be checked with the URL objects in the ArrayList saveScoreChangedList by comparing their String URL. If found the same URL, then it means a user changed the score of the website in the past and the URL object has the same URL in the saveScoreChangedList will be added to the ArrayList urls to keep the website’s score changes.

Finally, creating a copy of the current result list to make sure the ArrayList of URL can be used again by the changeScore() method, because when getting the top 10 searching result, the extract method will replace the top 10 URL. Then call the getTop10Resul() method to get the Top 10 searching result link.

Method changescore():

The method will first check if the urlList (Store the search result’s URLs) is empty, if it is empty, the method will simply print out an error message to ask the user to do at least 1 search before changing the score.

If the urlList is not empty:

The method first creates a Heap Sort object to use the Heap Sort method.

The 30 links in the searching result will be sorted and printed out with its score and URL for the user to know which one he wants to change.

Then ask the user to enter the link of the website that scores will be changed.

There will be a for loop to go through the urlList to check if the website that the user entered is already in the result list.

* If the website is already in the result list, the current 4 factors score will be printed out as a reference and ask the user to enter 4 new factors score. Then it will call the setHeapSize() and BuildMaxHeap() in the HeapSortto get a new Max Heap tree.
* If the website is new, a new URL object will be created, and the 4 random generated scores will still be printed out as a reference. The new URL object will be saved to the saveScoreChangedList to keep the changed score after the user is asked to change the 4 scores. Then setHeapSize(), BuildMaxHeap() will be called to build a Max Heap Tree, and then call MaxHeapInsert() to add the new website to the Heap Tree also the list and call setIndex() to give the website a before sorted index.

Method print4Scores():

The method will print the 4 factors score.

Method setNewScores():

The method will print out guide message and ask the user to enter 4 score one by one.  
Method storeKeyword():

The method will first check if the key already exists in the HashMap keywordMap.

If exists, the map will update the value of the key by one.

If it does not exist, a new key will be created and pushed with initial value 1.

Method getTop10Result():

The method creates a new HeapSort object with the ArrayList urlList which store the URL result of searching. Call setHeapSize() and BuildMaxHeap() to build a Max Heap Tree. Then use a for loop with a counter i, call HeapExtractMax() to extract the largest value in the tree for 10 times and print the information of the URL out.

Method printTop10Keyword():

The method first put the HashMap that stores the keywords and its search time into a List, then create an anonymous comparator to sort the keywords in the list by the searching times. Then by using a for loop with a counter I, print the top 10 keywords in the list.

Method checkUrlScoreChanged():

The method will go through the ArrayList saveScoreChangedList and compare the URL objects by comparing the String URL to find if the score of the website is changed manually before. If find it, return true, if not, return false.

Method getSavedUrlIndex():

The method will go through the ArrayList saveScoreChangedList and compare the URL objects by comparing the String URL to find if the score of the website is changed manually before and return that object.

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

Class Main: The main class where the program starts. It contains the main() method.

Method Main(): This is the method where the program starts. It will print the menu of the search engine.

Class Simulator: This class is the simulator of a search engine. It collects the most fundamental functions of this search engine.

Method search(): This method collects all the partial functions to make a search and print out the Top 10 search result.

Method change score(): Allow the user to input a URL to change the score of a website.

Method print4Scores(): The method will print the 4 factors score.

Method setNewScores(): The method will print out guide message and ask the user to enter 4 score one by one.  
Method storeKeyword(): The method will first check if the key already exists in the HashMap keywordMap.

Method getTop10Result(): The method will build the heap tree of the URLs by scores, extract the top 10 URLs and print them out

Method checkUrlScoreChanged(): The method will check if the user already changed the score of a website in the past search.

Method getSavedUrlIndex(): The method will get the saved URL with its changed score. It will be called after checkUrlScoreChanged.

Class URL: This class is for storing the URL and its scores

Method setOwnerPaidScore: To set a new score to the OwnerPaidScore

Method setKeywordsScore: To set a new score to the KeywordsScore

Method setExistedStore: To set a new score to the ExistedStore

Method setPageLinkToScore: To set a new score to the PageLinkToScore

Method setTotalScore: To set the TotalScore to a specific number, used in the priority queue to set it to be infinitygetTotalScore: Accessor to get the TotalScore

Method setIndex: To set(save) the page index(the index before it is sorted)

Method getOwnerPaidScore: Accessor to get the OwnerPaidScore

Method getKeywordsScore: Accessor to get the KeywordsScore

Method getExistedStore: Accessor to get the ExistedStore

Method getPageLinkToScore: Accessor to get the PageLinkToScore

Method getUrl: Accessor to get the URL

Method getIndex: Accessor to get the index(the index before it is sorted)

Class HeapSort: The class to do heap sort for the URL ArrayList

Method MaxHeapify: To compare a node's two children and to maintain the property of a max heap

Method BuildMaxHeap: To build a max heap tree

Method HeapSort: Sort the elements in the array in ascending order

Method HeapExtractMax: Extract the largest(root) from a heap tree

Method HeapIncreaseKey: After inserting a new element to the heap, sort it again to maintain the property of a max heap

Method MaxHeapInsert: Insert a new element to the max heap

Method setHeapSize: set the heap size back to the size of the ArrayList

Method HeapMaximum: Accessor to get the largest element(root) of the mas heap

Method getParent: Accessor to get the index of the parent node

Method getLeft: Accessor to get the index of the left child node

Method getRight: Accessor to get the index of the right child node

Method printList: to print all the elements in the list, put it in the heapsort file for the convenience of giving every method's output

Class WebCrawler: By using this class to create a web crawler which can crawl URLs from the Internet.

Method search: This method starts the search. It will check if the page contains the keyword, then call crawl method to crawl the URLs on the page

Method crawl: The method will crawl the links and put them into a set to keep uniqueness

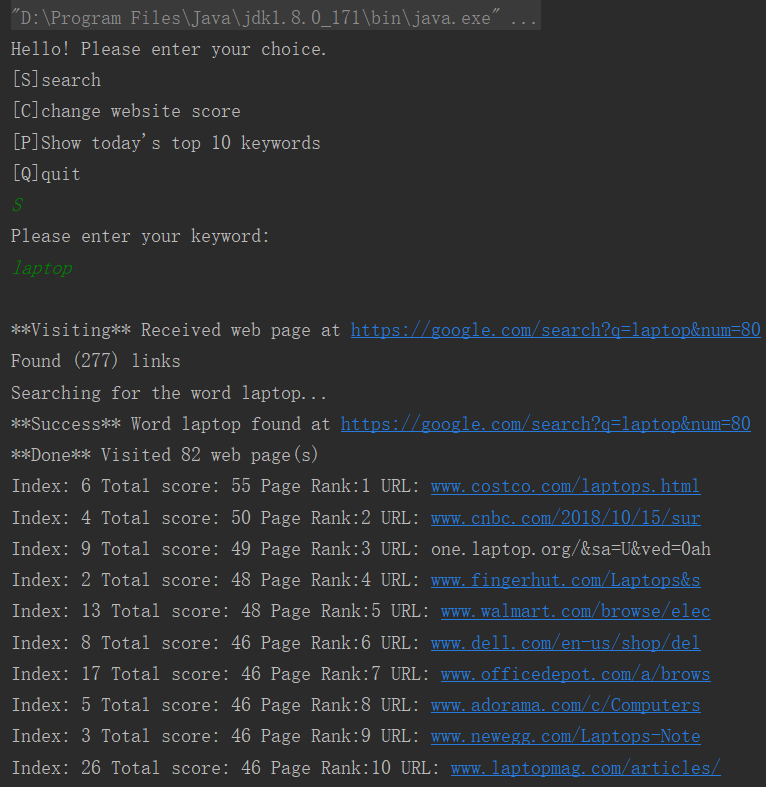
Method getDomainName: The method will use pattern and matcher to extract the domain URL from google result links

Method searchForWord: This method will check if the website contains the keyword

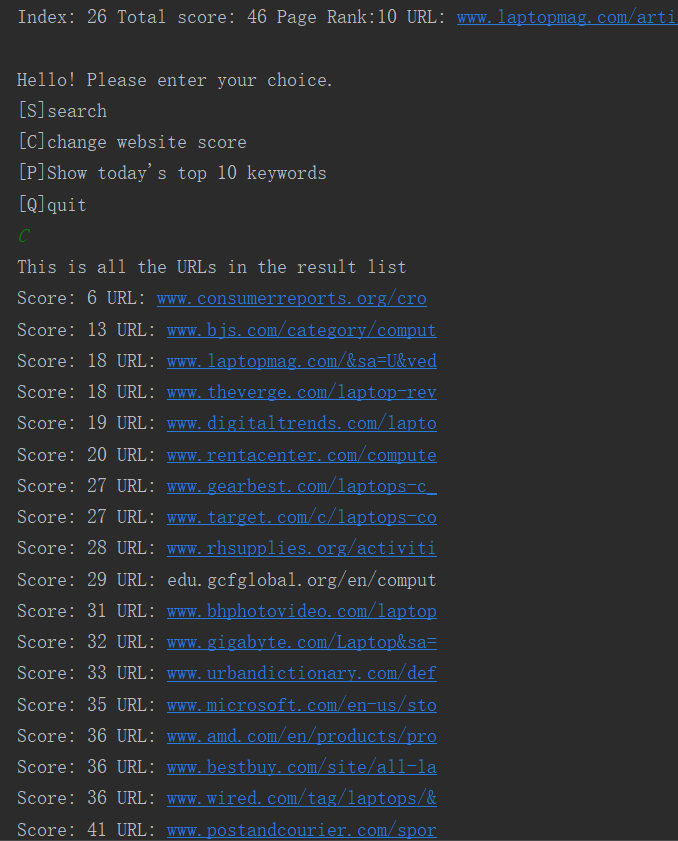
Method getUrls: Accessor to get the set of URLs result

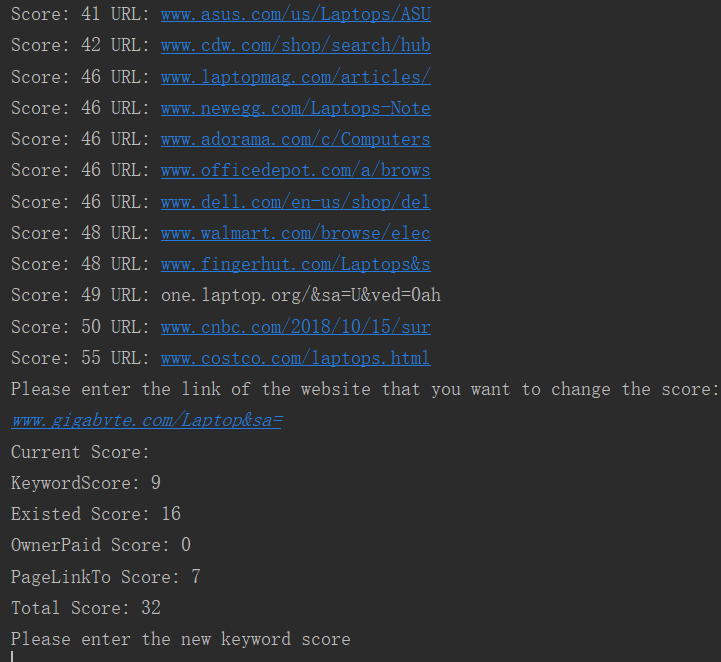
**c) Provide a lot of screenshots of each simulation process/procedure including inputs and outputs for each of function listed in item 6 associated with the two major features listed at the end of the Problem Statements.**

Search():

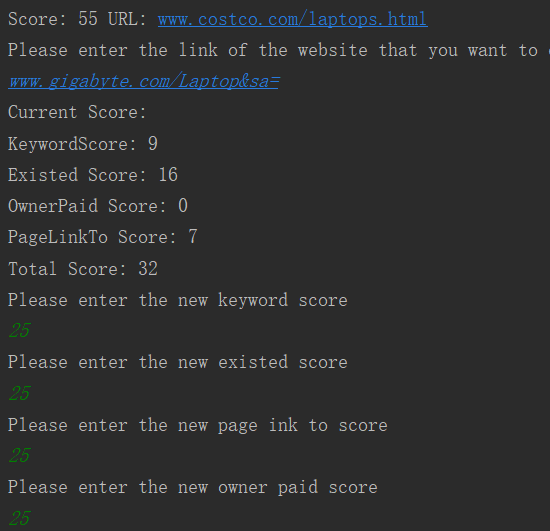


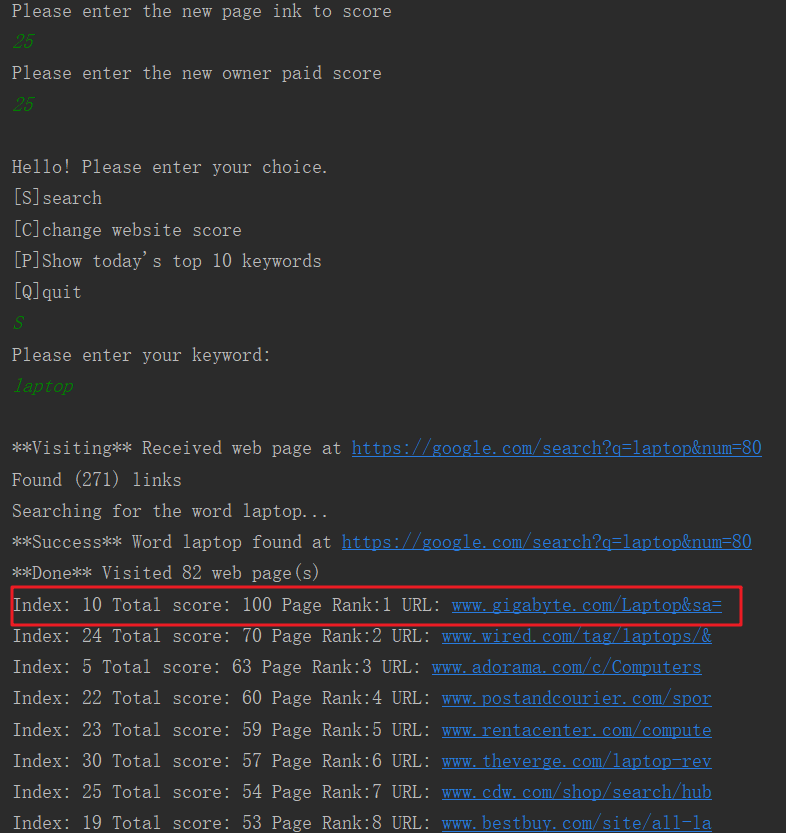
changeScore(): (continue with the search())

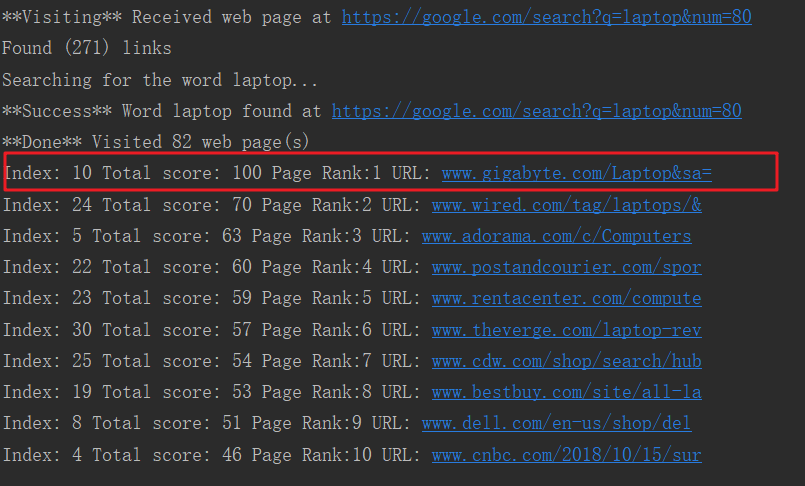


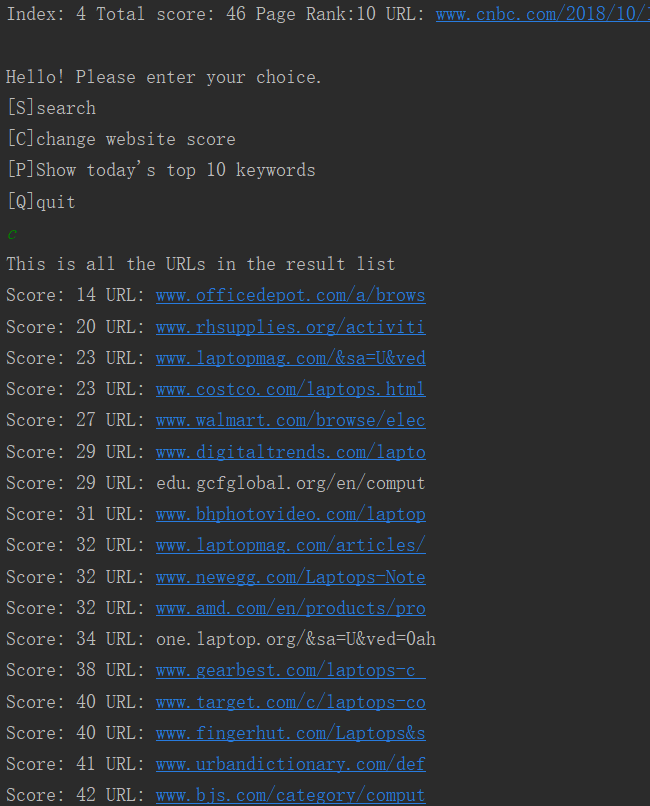


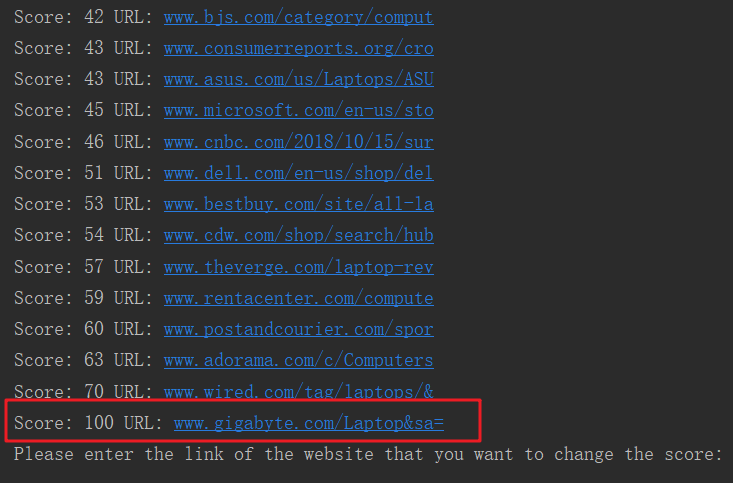
This is change the score of the link that is already in the urlList



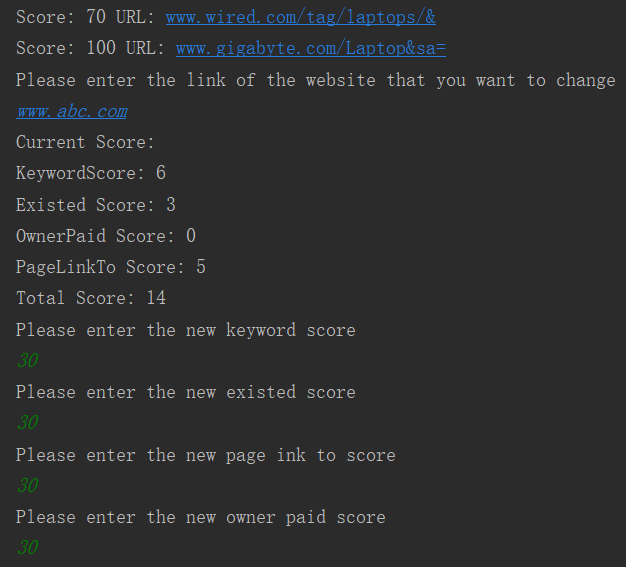


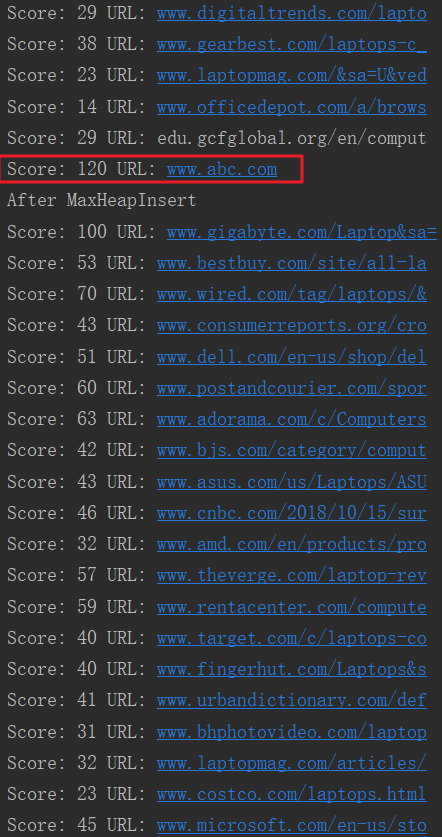




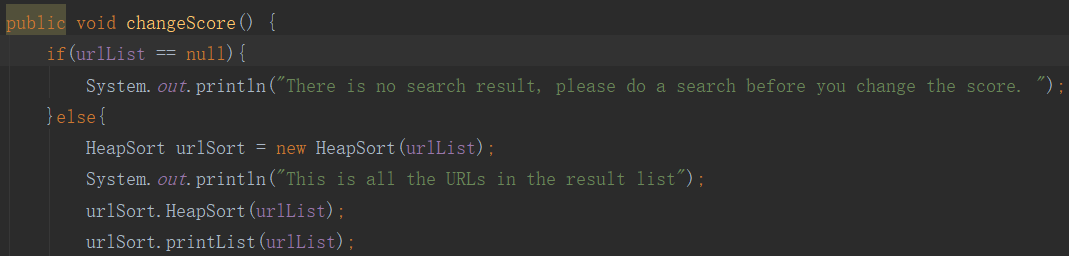
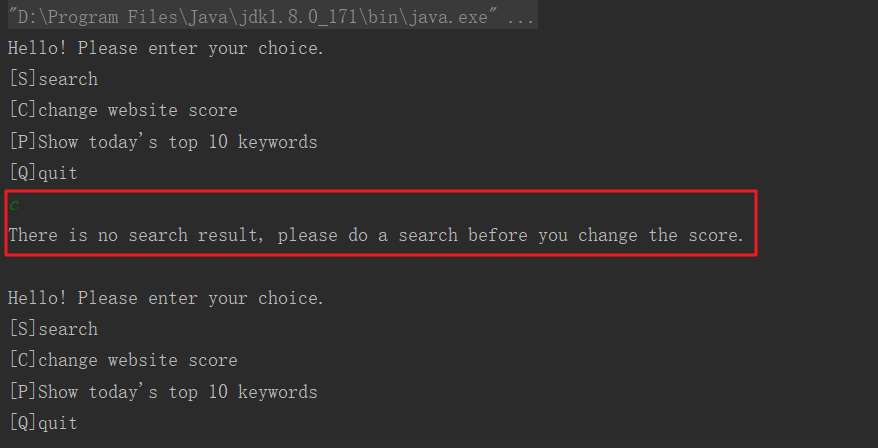


Continue to add a new website that is not already in the urlList

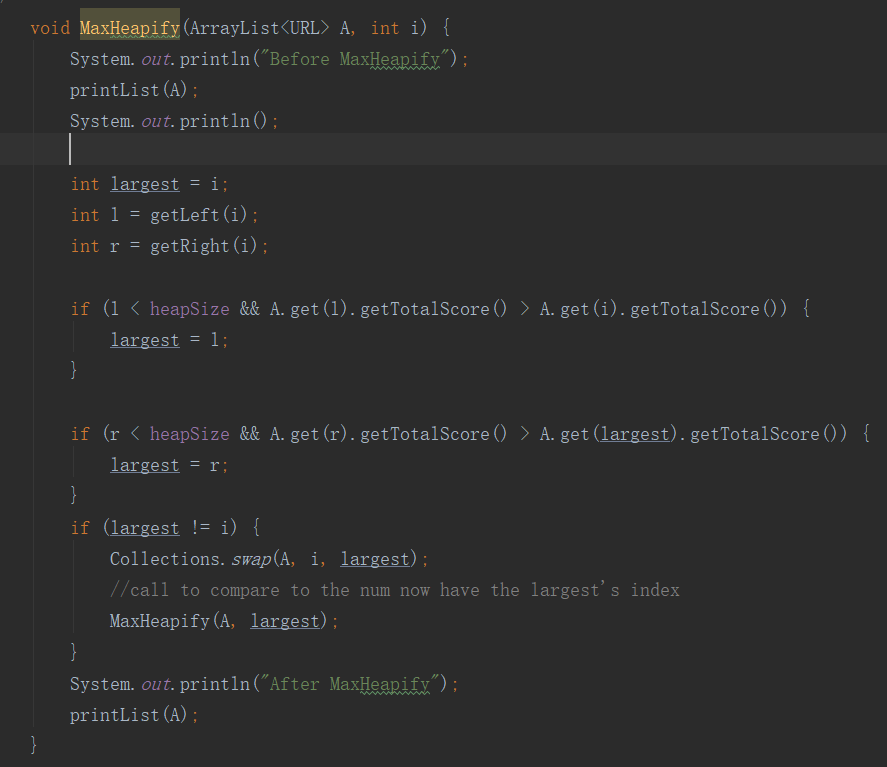




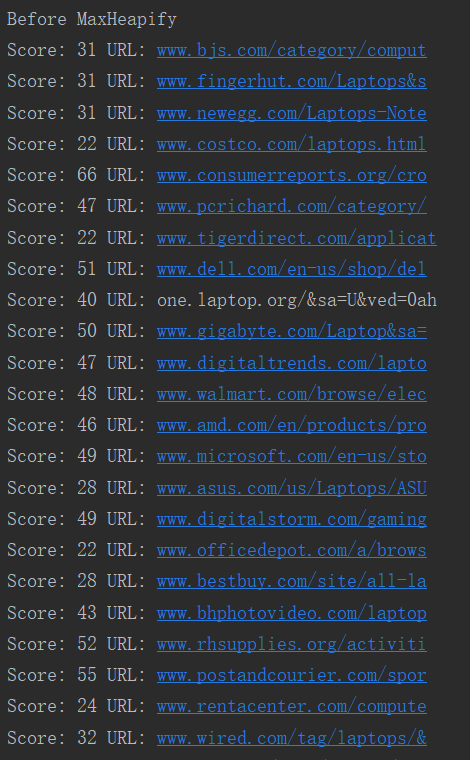
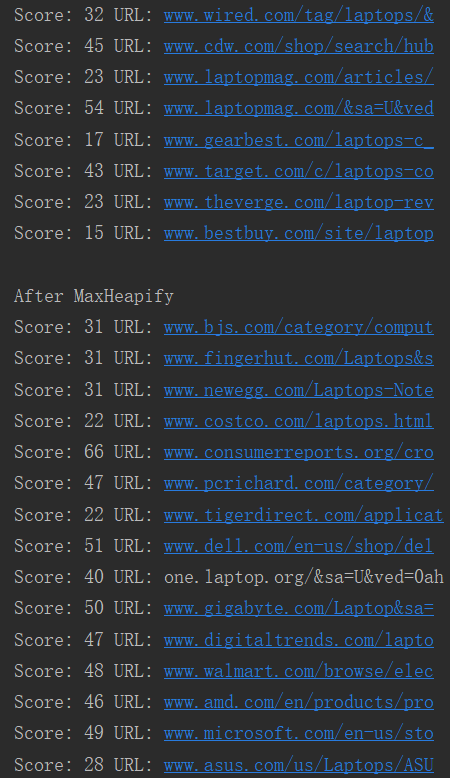
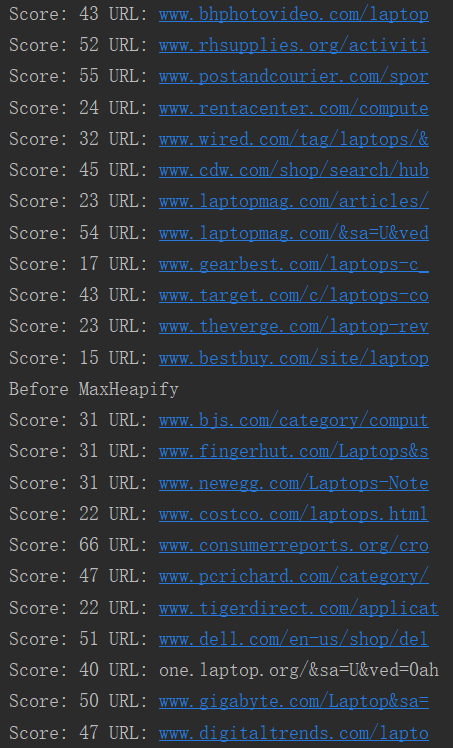
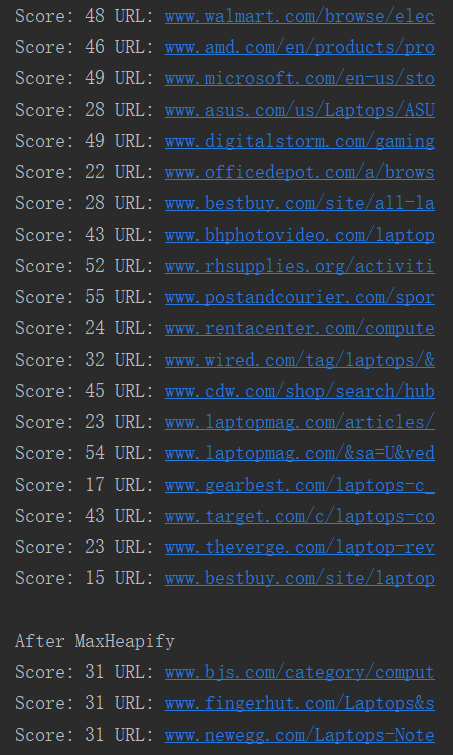
When trying to change a website score without searching first:

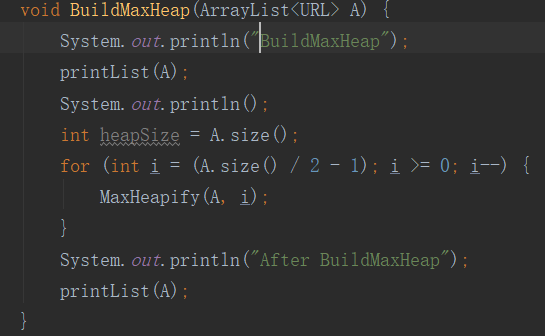
Max-Heapify():



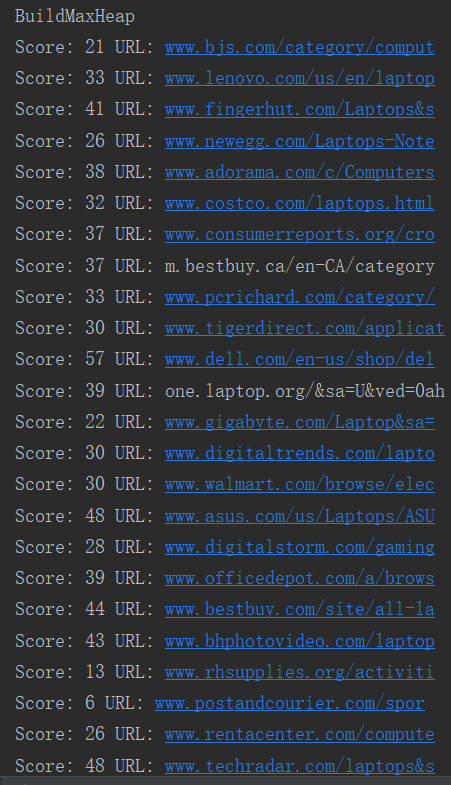
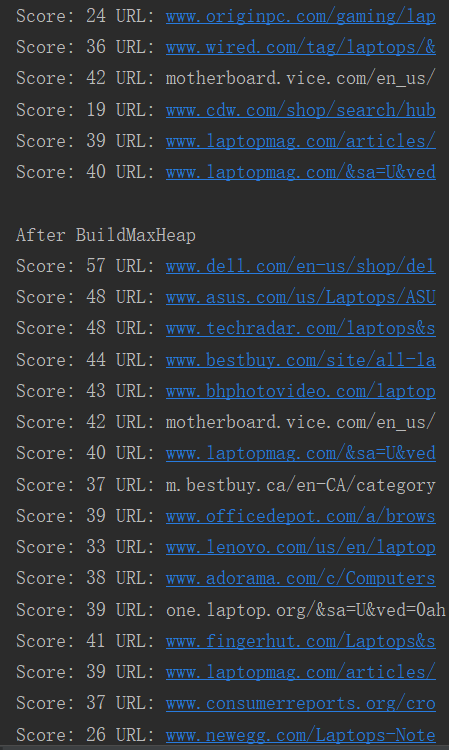
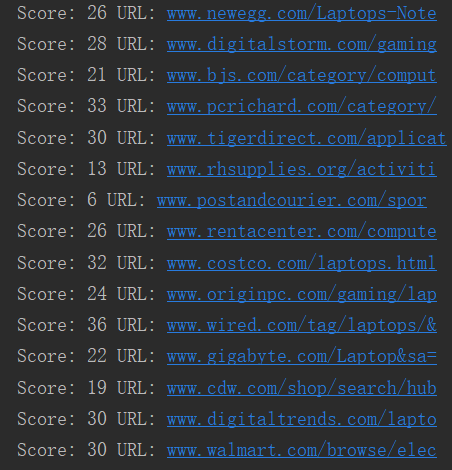
Output:

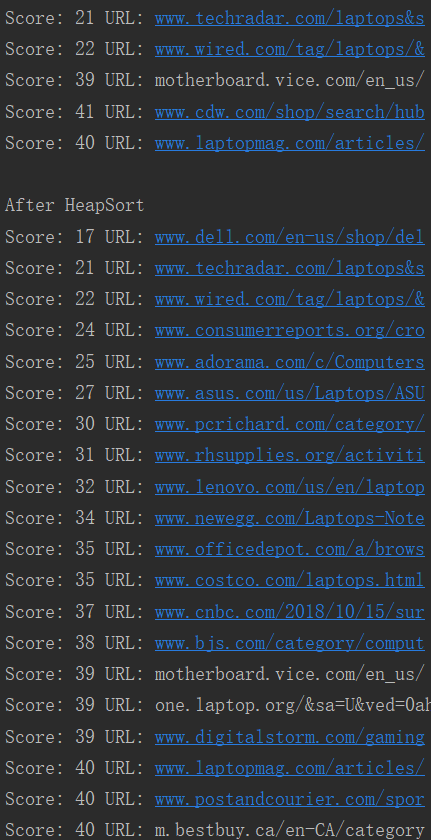
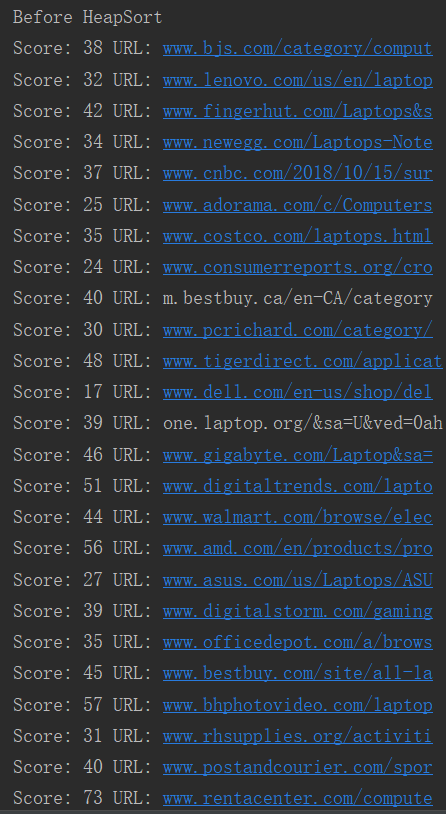
2) Build-Max-Heap()

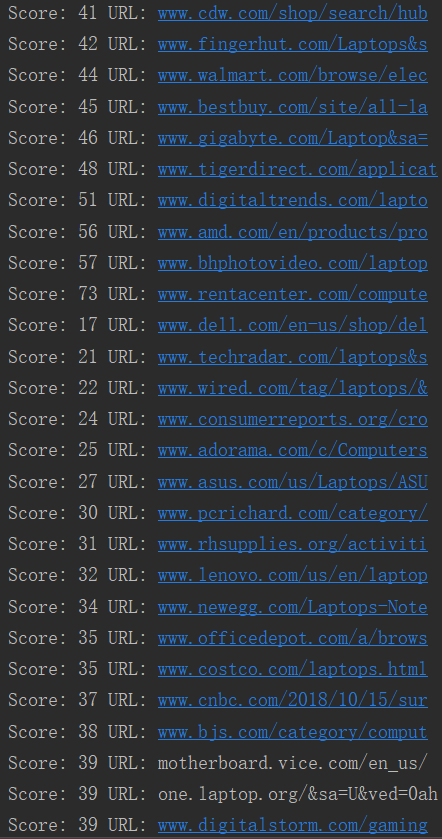
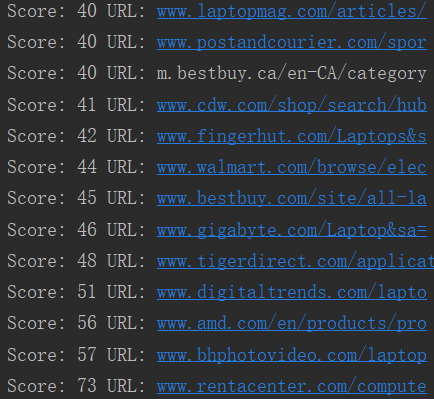


Output:

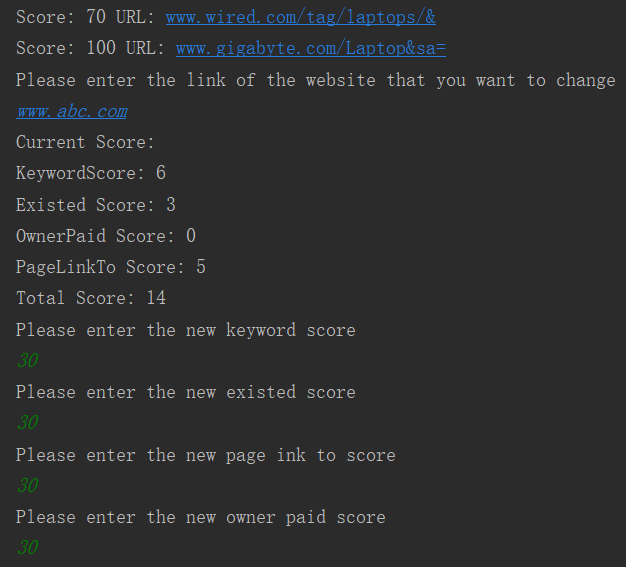
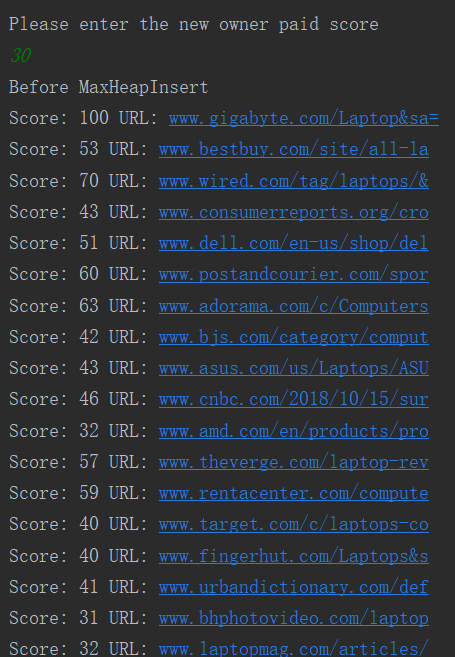
3) Heapsort()

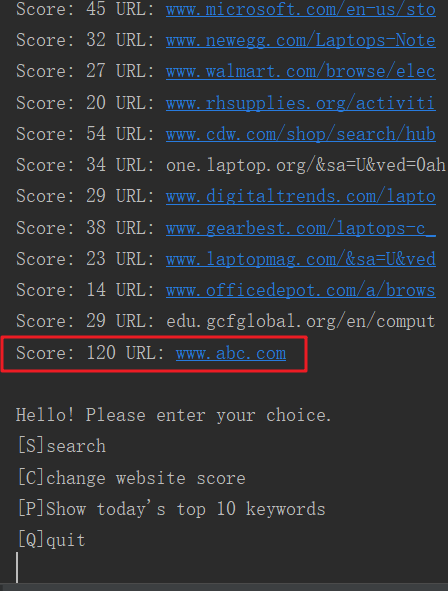
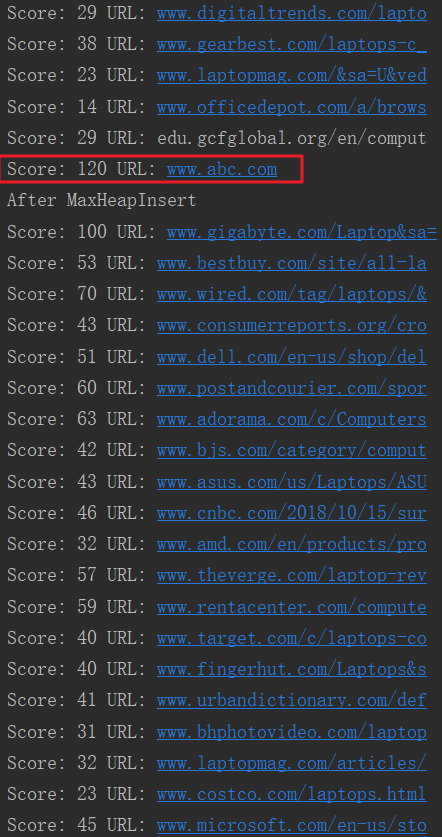
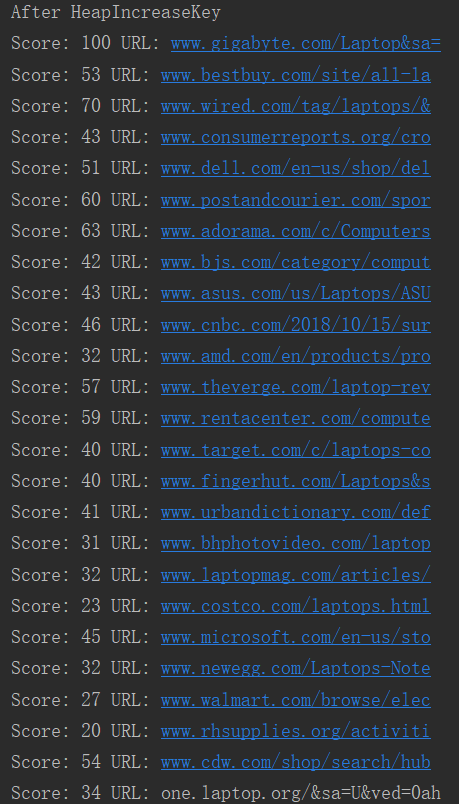
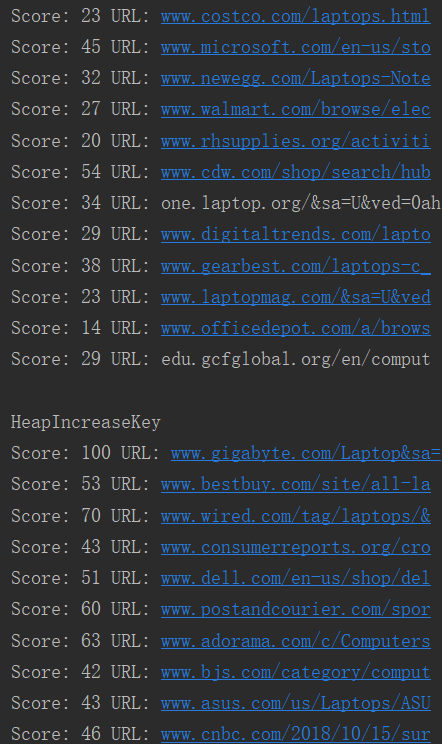


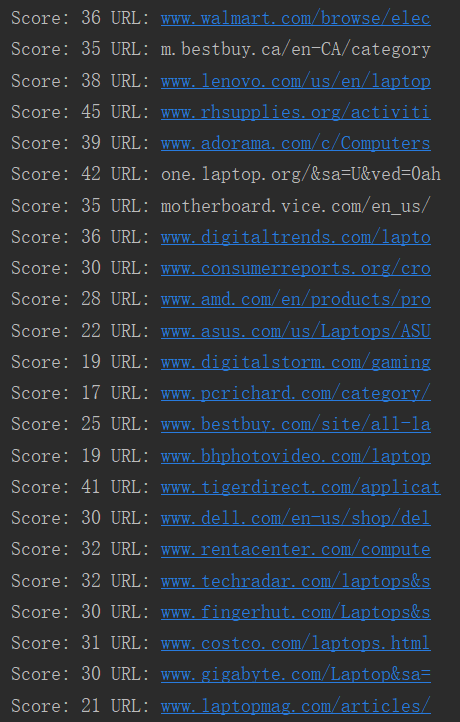
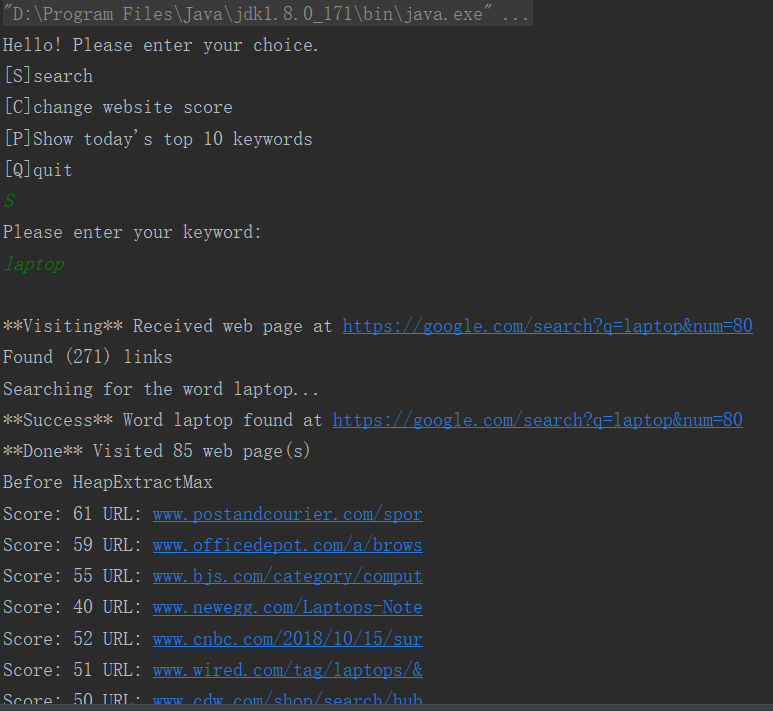
4) Max-Heap-Insert() & 6) Heap-Increase-Key

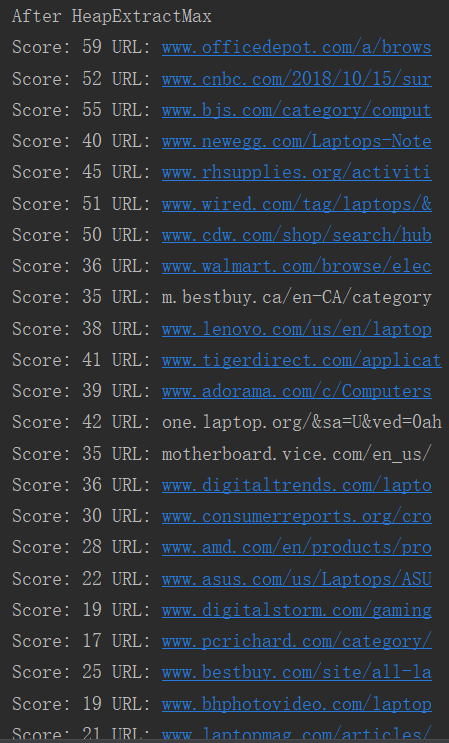
This part countiue part to the search() and changeScore() when add a new url to the rank



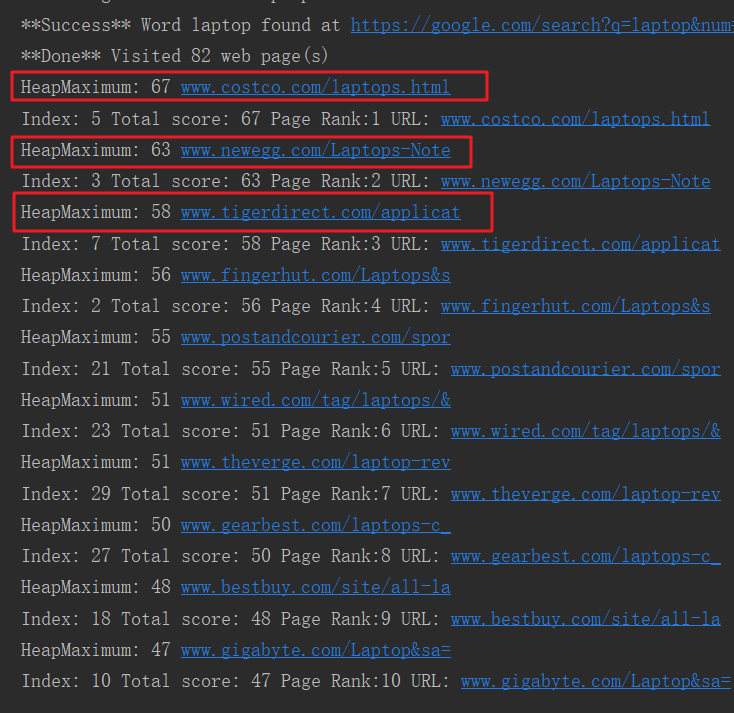


5) Heap-Extract-Max

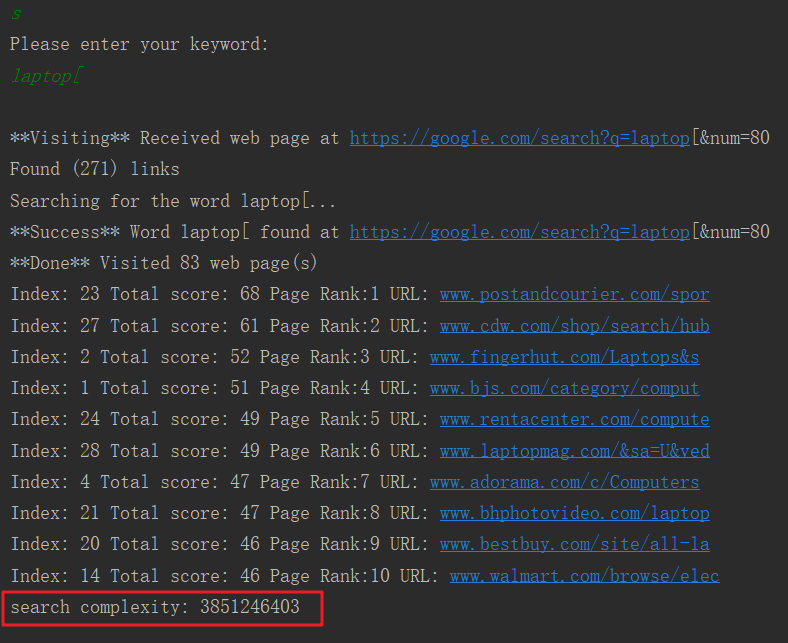


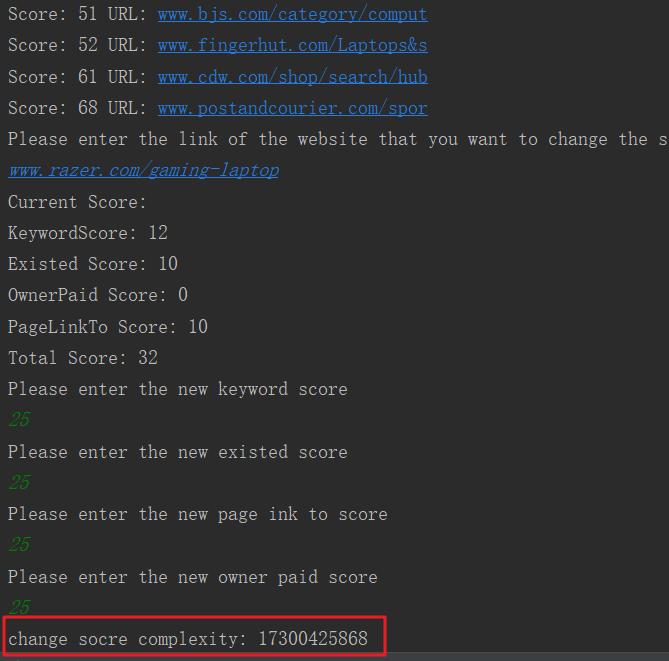


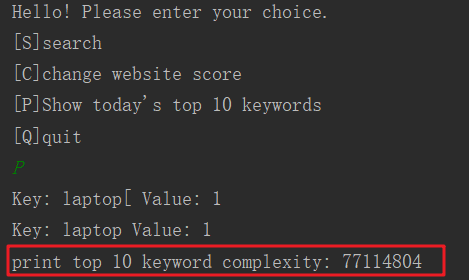
7) Heap-Maximum



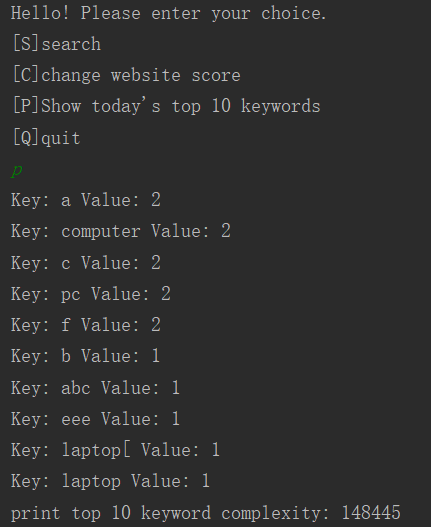
Complexity:







Print Top 10 Keywords:

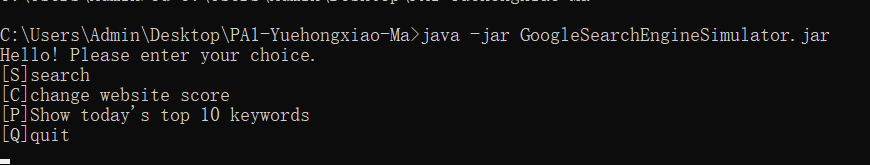


**d) The procedure (step by step) of how to unzip your files, install your application and run/test**

**your codes.**

(For Windows, because I don’t know how to use Macbook)

1. The zip file name is “PA1-Yuehongxiao-Ma.zip”, right click and find your common use extract tool (like 7-zip) to “extract files”
2. Open command line, use “cd” + folder directory to change to the folder that contains the extracted files above.
3. In the command line, enter java -jar GoogleSearchEngineSimulator.jar
4. And then it will show up the menu



1. Follow the instructions on the menu, enter the letter “S” “C” “P” or “C” to start test.

Attention: When changing score, if you want to enter an URL that is already in the result URL list, you might need to add a space after the URL and then “enter” to finish the input. This is because when using IDE to run the problem, without space after the URL, the IDE will automatically pop out the website as I press “Enter”.

**e) Problems encountered during the implementation.**

The first problem is about the heap size. I translate the pseudo code at first, and when I tried to use an array of integer to test it. The output always shows the exception of out of bounds. I tried to print every array changes during the sort and follow the pseudo code to do the sorting by hand, but it does not work for a while until I use the debug tool in the IDE in where I found my heap size stays 0. I realized that the heap size comes from the Array size but it is changing as an independent variable. This problem was fixed by using a heap size reset method and use it before methods that use the array size as the heap size to start.

The second problem is about the extract max method. When I test the function of change website scores, my multiple times of changes to the score of different website lead to my heap tree output with a lot of repeated element, and all of them are not in the top 1,0 results. My top 10 results keep changing and the old top 10 results are gone. I finally noticed that the extract max method replaced the largest one with a smaller one. And I fixed it with making a copy every time when I call that method related to extracting max and return the original array back after the method finished for the reuse.

The third problem is about the way to sort the top 10 keywords. When design saving a keyword and its single integer variables, I decided to use HashMap because of its obvious function of the key and value. However, when I want to sort the HashMap by the value of the key, HashMap does not have a domestic method to support sorting. It takes me a while to figure out the way to put the HashMap’s elements into a list to sort it. By the meantime, I also take advantage of the comparator that I learned from other class to make the sorting part simpler.

There is still a problem that I haven’t solved yet in the implementation. When designing the change of a website score, I thought that user might want to change the score of a website that is not in the search result, so I added the feature to check if the URL is in the result list or not and have a subroutine to create a new URL object and added to the list. The new URL could be any string or even a website that has no relation to the keyword. By print the ArrayList before and after the insert, the new URL is able to be added to the result list and also saved to another ArrayList as a score manually changed website. However, when I do the search with the same keyword, even the website is saved, should be able to reuse, it does not show up in the search result. The reason is that my web crawler will get result URLs from the Internet (Google search result page), and even I changed the URL’s score in the local, it does not affect the result page online, and the new URL won’t show up in the search result page, so it won’t be crawled and saved to the result URL ArrayList. To solve this, it might take more time to resign my URL class to add a feature to record the keyword of this URL, and I might also need a method to check the new URL’s keywords with the user input keyword to get the new URL back to the result list. I ran out of time so did not fix this problem.

**f) Lessons Learned**

The assignment gives me a deeper understanding of the Heap Sort when I ran into different implementation problems and fix them by reading through the code many times and use real test case to debug. Besides that, I also learned to build a simple web crawler and had a brief idea about how it works. Accidentally, I learned a new usage of the data structure HashMap and List on combining them to sort the HashMap’s key.

What I learned important from this assignment is the importance of spending the time to do a good design before starting to code. I started to code from translating the pseudo code of Heap Sort, build the web crawler and the URL class. I planned to make the heap sort work first, then get some links from the crawler, make them into URL objects and put into an ArrayList. But I did not go deep to think about how the sorting work inside of the heap sort and what will happen to the ArrayList when I sort it.

After I connect them together and start my testing, a number of small errors came out. And for the rest of my coding time is to debug every small mistake that I made. Besides, I also find out when I did the debugging and tried to fix the errors, instead of coding consistently, what I am doing is more like to use small code pieces to fix every small error. Because of that, my code somewhat lost a continuous usage to the current existed fields because I might do some unnecessary initialization within my assisted method. When I write the report, I reviewed my code and tried to figure out every method’s purpose and check their implementations. I noticed that there are so many lines of code that I should take more time to think about to simplify them.