# Web Scraping Program

For IBE 151. Due: 5 November 2020
By: Dena E. Utne (Note: I am making an individual submission.)

The functionality of the web scraping program can be divided into the following three main steps:

1) The program scrapes the web site https://xkcd.com. It collects the file names and sizes of the main comic images that are in the standard format for the web site. The images are .png, .gif or .jpg. (There are a handful of images that are out-of-format. It skips these.)
2) The program sorts the images in descending order by file size using a merge sort algorithm.
3) The program prints a table for the user containing the 10 largest files in descending order. It then asks the user to pick one image file to open. The program validates the user input and continues to requestsnew input as long as the user's input is invalid. The program then opens the image file selected by the user.

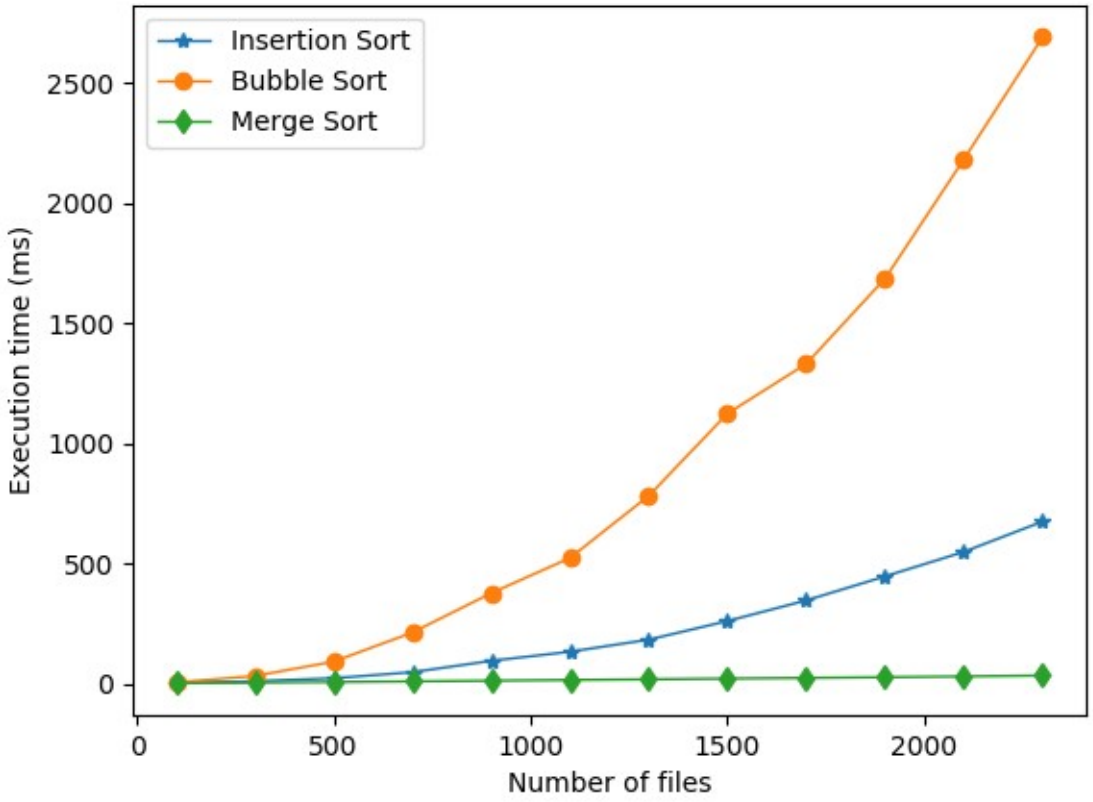
The "main" function that controls the overall flow of the program is in web_scraping.py. The sub-functions needed for step #1 are in the scrape.py file. The sub-functions needed for step #2 are in the sort_lib.py. The sub-functions needed for step #3 are in the pick_and_show.py file.

The web scraping program is documented in detail within the .py files themselves.

It should be noted that the web scraping program generates a text log file called "web_scraping_log_file.txt" that can be used for debugging.

## Comparing sorting algorithms

The code for testing  and comparing sorting algorithms is in plot.py and sort_lib.py. Each sorting algorithm was run on varying number of files from 100 files to 2300 files in increments of 200. For each number of files and each algorithm, the algorithm was run with 1000 iterations, repeated three times. The minimum of the three repetitions was used for the results. Since the timeit.repeat function gave the results for each 1000 iterations in seconds, the average time for 1 iteration is the same value but in milliseconds.




The graph clearly indicates that the merge sort algorithm performed best. The bubble sort algorithm was very slow. The insertion sort algorithm performed in the middle of the other two but closer to the merge sort's performance than to the bubble sort's. As expected, execution times increase with increasing numbers of files to sort.

For debugging purposes, some numerical results were written to a text log file called "timing_data_log_file.txt."

# Libraries used

1) sys: I call sys.exit() when I wish to terminate the program after certain errors are encountered.
2) requests: This is used for opening web urls and creating response objects. It handles internet connection issues. It is used to pass the text component of web pages to bs4. BeautifulSoup().
3) lxml: html parser used when creating bs4.BeautifulSoup objects.
4) bs4.BeautifulSoup: Used along with an html parser for parsing html into BeautifulSoup objects. Allows for extraction of information from html.
5) os: This module is used when creating a subdirectory to store the downloaded image in.
6) PIL.Image: used to display the image selected by the user.
7) json: For writing data from the web scraping that was used later for timing the various sorting algorithms.

For comparing sorting algorithms, I also used the following two additional libraries:
1) matplotlib.pyplot: Used for plotting the execution times of the various sorting algorithms.
2) timeit.repeat: Used for running timed iterations of sorting algorithms.

# Bibliography

Freeman, E. 2018. Head First Learn to Code, Chapter 4, part 2 and pp. 435-448 of Chapter 10.

Sweigart, A. 2020. *Automate the Boring Stuff with Python*, 2e, Chapter 12.

For the function get_image_file_size, I used code from:
https://stackoverflow.com/questions/14270698/get-file-size-using-python-requests-while-only-getting-the-header.

For the function def show_image(image_name), I used:
https://stackoverflow.com/questions/35286540/display-an-image-with-python/35286593

For the insertion sort and merge sort algorithms in the functions insertion_sort_alg, merge_sort_alg, and merge, I relied heavily on the code from IBE151 Canvas modules, week 43:
https://himolde.instructure.com/courses/1607/pages/sequence-algorithms-searching-and-sorting-lists?module_item_id=34576

For the bubble sort algorithm in the function bubble_sort_alg, I used:
Freeman, E. 2018. Head First Learn to Code, Chapter 4, part 2

For understanding some of the sorting and plotting code we were given in class, I conversed a bit with another member of my household. He is by no means a python expert, but some shared brainpower was helpful.