CMPS 1500 Introduction to Computer Science I – Spring 2019

# Lab 6

Due **Wednesday 3/27/19** at 11:59 p.m. on Canvas and Zybook

Please follow the usual lab guidelines (file naming rules, honor code, comment requirements). New additional lab rules:

- The code will be graded both on functionality and on style (meaningful variable names, presence of comments and docstrings, absence of commented-out code, etc). Perfectly running programs will lose points for lack of comments, presence of commented-out code, etc
- Your code has to run. Any code that immediately crashes will receive 0 points.
- Whenever you submit code on Canvas, please submit the entire code (function and main part), so that we can run the code.

**Read the entire assignment before starting your work in order to plan your time.**

0. **Sort verification,** `lab6pr0.py`**, Zybook**

   (a) (Warm-up) Write a program that checks whether a list of numbers is sorted in increasing order and prints `True` if the list is sorted in increasing order, and `False` otherwise. Test your program for several lists. If a list contains duplicates of a number, they are considered in increasing order if they appear next to each other. So, `1, 2, 2, 2, 3, 8, 9, 9, 9, 12` are considered to be in increasing order.

   (b) Write a function `is_sorted(lst)` that takes as input a list of numbers, and returns `True` if the list is sorted in increasing order, and `False` otherwise. Provide doctests that test your function for several inputs. (It's fantastic if you can make your function recursive. But it is ok to use loops as well).

   (c) Write a function `is_file_sorted(filename)`. This function takes as input a name of the input file containing a list of integer numbers, one number per line, and returns `True` if the list is sorted in increasing order, and `False` otherwise. Test your function with several inputs.

   **Important:** *Remember to convert string values to integers before testing. Otherwise your numbers will be sorted lexicographically, for example, numbers* `[1,2,11,12]` *will be sorted* `[1,11,12,2]`*, which is not the right order for integers.*

1. **Sort verification, code analysis, code style,** `lab6pr1.py`**, Canvas**

   (a) Now add the main part to the program from problem 0. It should prompt the user for an input file name, and it should output a user-friendly message with the result, as follows:

```
>>>
Please enter file name: input.txt
Looks like input.txt needs to be sorted
```

or

```
>>>
Please enter file name: inputsorted.txt
Congratulations! The file inputsorted.txt is nicely sorted!
```

(b) For each code line in the main part of your file, add a comment describing the asymptotic running time of that line of code in terms of $n$, the number of entries in the input file. On the last line in your file, add a comment with the total big-Oh running time of this program, and a brief justification.

2. **Sort usage**, `lab6pr2.py`, Zybook

   Carefully study the file `sortingalgos.py` that includes implementations of sorting algorithms provided with this lab.

   (a) Write a function `use_mergesort(inputfile, outputfile)`. This function takes as input a name of the input file containing a list of integer numbers, one number per line, and the output file name. This function should read-in numbers from the file, sort them using the provided mergesort function, and output the sorted numbers to the output file, one number per line.

   **Important:** *Remember to convert string values to integers before testing. Otherwise your numbers will be sorted lexicographically, for example, numbers* `[1,2,11,12]` *will be sorted* `[1,11,12,2]`*, which is not the right order for integers.*

   (b) Write a function `use_selectionsort(inputfile, outputfile)`. This function takes as input a name of the input file containing a list of integer numbers, one number per line, and the output file name. This function should read-in numbers from the file, sort them using the provided selection sort function, and output the sorted numbers to the output file, one number per line.

   (c) Insert

   ```
   import random
   random.seed(0)
   ```

   into your code file. A call to `random.randrange(a)` generates a random number between `0` and `a-1`, inclusive.

   Now write a function `generate_nums(filename, n)` that creates a file named `filename` and writes to it `n` random numbers from 0 to 99, inclusive.

   Use the `generate_nums()` function to generate files of different size that you would then use for testing your `use_mergesort()` and `use_selectionsort()`.

3. **Practical performance of sorting, `lab6pr3.py` and `lab6pr3.pdf`, Canvas**

   In class we spoke about the asymptotic runtime analysis of merge-sort and selection-sort. In this exercise you evaluate the performance of these two algorithms in

practice. Start by carefully studying the file `timeexample.py` that shows how to use the time functions to measure runtime.

(a) Modify your `use_mergesort()` into a function called `analyze_mergesort(inputfile, outputfile)` that calculates and displays to the user the time (in seconds) it took to read in the values from the input file, the time it took to sort the values, the time it took to output the values to a file file, the total time it took to run the program, and the total number of values that were processed. Round the seconds value to 6 decimal digits. For example,

```
>>> analyze_mergesort("input10.txt", "sorted10.txt")
It took 0.008058 seconds to input 10 values from file input10.txt
It took 1.112337 seconds to sort 10 values using merge sort
It took 0.121144 seconds to output 10 sorted values to file sorted10.txt
Total time the program took is 1.241539 seconds
```

Your numbers will be different from mine.

(b) Similarly modify your `use_selectionsort()` function into a function called `analyze_selectionsort(inputfile, outputfile)`.

```
>>> analyze_selectionsort("input10.txt", "sorted10.txt")
It took 0.008058 seconds to input 10 values from file input10.txt
It took 1.112337 seconds to sort 10 values using selection sort
It took 0.121144 seconds to output 10 sorted values to file sorted10.txt
Total time the program took is 1.241539 seconds
```

(c) Now you are ready to compare merge sort and selection sort side-by-side. First, use your `generate_nums()` function to generate files with 10, 100, 1,000, 10,000, 100,000 and 1,000,000 numbers. Then, run your analyze functions from exercise 3(b) on these files. Your goal is to see how much time selection sort and merge sort take on the same list of numbers.

(Note: in 2014 selection sort on a million numbers took 29 hours on a standard laptop. You can use this value in your analysis if you don't have patience to wait until your simulation completes. It would be interesting to see the numbers for this year, let me know if you figure it out.)

If the program takes too long to run and you stop it early, please include in your submission an explanation with the approximate time that you waited before you stopped the program, and also your prediction on how long the program would actually take (based on its performance for a smaller set of numbers).

i. First, for each of the input sizes, copy the information about the time taken by both functions into a Word file. Clearly separate the outputs produced by running different files. Highlight the time taken by the sorting portion of the program.
*You only need to copy the time measurements. Do not copy the numbers that were actually sorted.*

ii. In each case, state what percentage of total running time is taken by input and output. Look at these percentages and answer - what are these numbers telling us?

iii. Make four graphs. In the first one, plot the time taken to merge-sort the file vs. the number of values in the input. On the second one, plot the time taken to selection-sort the file vs. the number of values in the input. Make sure you only use the actual sorting time, not the total time of the program. Now, repeat these steps but plot the actual time taken by the entire program vs. the number of values processed, one graph for merge sort and one graph for selection sort. Add clear captions to your graphs.

    You may want to Excel to produce graphs and copy them into your Word file. You are free to use any other word processing/plotting software that you're familiar with.

iv. In a few sentences, interpret your graphs. What are the plots telling us?

Save your file as .pdf and submit it to Canvas.