

COSC122 (2020) Lab 6.1

Simple Sorting

Goals

This lab will give you some practice with the Insertion, Selection and Shell Sort algorithms; in this lab you will:

- implement a reverse and minimum selection sort algorithm;
- explore the best, worst and average cases of an insertion sort algorithm and
- explore the effect of different sequences of sub list count on a shell sort algorithm.

You should be familiar with the material in Section 5.3 ¹ of the textbook before attempting this lab.

Selection Sort

The `sorting.py` module provides maximum selection sort algorithm (listing 5.11 page 211) ².

- Modify this implementation to produce a reverse-sorted list.
- Try to predict the number of data comparisons needed to sort a list of 1000 items.
- Include statements to count the data comparisons (this was covered in lab1). Test your implementation with the files (`file0.txt`, `file1.txt`, `file2.txt` and `file3.txt` containing 10, 100, 1000, 10000 elements respectively. Code needed to read the elements is provided in the `sorting.py` module.
- Implement a minimum selection sort that sorts the numbers in ascending order by selecting the minimum in each iteration. Test your implementation with the files provided.

Insertion Sort

The `sorting` module provides insertion sort algorithm (listing 5.12 page 215) ³. You will be measuring how the program behaves in the worst case, average case and the best cases.

- Before running the insertion sort method, try to predict the number of data comparisons when insertion sort is given sorted and reverse sorted lists.
- Include statements to count the data comparisons. Test your implementation with the files 1 to 7. Files 4 and 5 contain a sorted list of 1000 and 10000 numbers respectively. Files 6 and 7 contain a reverse sorted list of 1000 and 10000 numbers respectively.
- Why doesn't insertion sort on file6 or file7 use the worst case number of comparisons?

> **Complete the Selection and Insertion sort questions in Lab Quiz 6.1**

¹Online textbook: [Sorting](#)

²Online textbook: ActiveCode 1 in the [Selection Sort section](#)

³Online Textbook: ActiveCode 1 in the [Insertion Sort section](#)

Shell Sort

The `sorting.py` module provides the shell sort algorithm (listing 5.13 page 218, 1st edition listing 4.21 page 170)⁴ with the gap starting at `gap = n // 2` and changing to `gap = gap // 2` in each subsequent iteration.

- Include statements to measure how shell sort behaves with sorted, reverse-sorted and random lists.
- Now write a shell sort function (called something like `shell_sort2`) that accepts a gap list as a parameter and try the sequence `[31, 15, 7, 3, 1]` to see if the performance improves.
- Compare the data comparisons for the new gap list version with the previous version.
- Can you find a better sequence?

> *Complete the Shell sort questions in Lab Quiz 6.1*

(Extras)

- Write a small program to generate a file with 10000 items (called `file8.txt`) that will give a worst case number of comparisons for insertion sort (and Shell sort). Calculate the worst case number of comparisons and confirm that your file takes that many comparisons to sort.

⁴Online Textbook: ActiveCode 1 in the [Shell Sort section](#).