Carleton University
Department of Systems and Computer Engineering
SYSC 2100 - Algorithms and Data Structures - Winter 2021

Lab 12 - Sorting Algorithms

Submitting Lab Work for Grading

Remember, you don't have to finish the lab by the end of your lab period. The deadlines for submitting solutions to cuLearn for grading are listed in the *Wrap Up* section at the end of this handout. Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the deadline.

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

References

Problem Solving with Algorithms and Data Structures Using Python, Chapter 6, *Sorting and Searching*

Getting Started

In this lab, you'll explore a few of the "classic" algorithms for sorting collections of values.

When analyzing sorting algorithms, we usually focus on two operations:

- the total number of times that pairs of values are compared to determine if they are out of order;
- the total number of times that pairs of out-of-order values are exchanged (swapped).

In this lab, you'll modify some of the sorting functions presented in the textbook to obtain this information.

Download bubble_sort.py, selection_sort.py and merge_sort.py from the *Lab Materials* section of the main cuLearn course page.

Exercise 1 - Bubble Sort

Read Section 6.7, *The Bubble Sort*.

Bubble sort is one of the simplest sorting algorithms, but is considered to be one of the most inefficient.. (See the analysis immediately above Table 1.)

In bubble_sort.py, modify function `bubble_sort` to keep track of the number of comparisons and swaps performed by the algorithm. The function should return a tuple: the first value will be the number of comparisons and the second value will be the number of swaps.

Note: each time a statement of the form `a, b = b, a` is executed, it should be counted as one swap.

Run the docstring tests. Notice that one test sorts a list of integers that are arranged in ascending order, and the other test sorts a list of integers that are arranged in descending order. Verify that your modified bubble sort function sorts the list and returns the correct number of comparisons and swaps.

Exercise 2 - Short Bubble Sort

Section 6.7 (below Table 1) describes an improved bubble sort algorithm that finishes as soon as it determines that the list is sorted. The code is in function `bubble_sort_short` in bubble_sort.py.

In bubble_sort.py, modify function `bubble_sort_short` to keep track of the number of comparisons and swaps performed by the algorithm. The function should return a tuple: the first value will be the number of comparisons and the second value will be the number of swaps.

Run the same tests on `bubble_sort_short` that you used to test `bubble_sort`. Compare the results with the results you obtained in Exercise 1 Make sure you can explain any differences in the results. (You don't have to submit an explanation, but you should know this for the final exam.)

Exercise 3 - Selection Sort

Read Section 6.8, *The Selection Sort*.

Selection sort is $O(n^2)$, like bubble sort; however, it tends to execute faster than bubble sort.

In selection_sort.py, modify function `selection_sort` to keep track of the number of comparisons and swaps performed by the algorithm. The function should return a tuple: the first value will be the number of comparisons and the second value will be the number of swaps.

Run the same tests you used in Exercises 1 and 2. Compare the results from the three exercises. Do they support the claim that `selection_sort` should execute faster than `bubble_sort`? (You don't have to submit an explanation, but you should know this for the final exam.)

Exercise 4 - Merge Sort

Read Section 6.11, *The Merge Sort*.

Merge sort is a recursive sorting algorithm that is $O(n \log n)$. As such, this algorithm should have significantly better performance than bubble sort and selection sort for large lists.

In merge_sort.py, add brief comments before each of the three `while` loops to explain the purpose of each loop.

Modify function `merge_sort` to keep track of the number of comparisons and swaps performed by the algorithm. The function should return a tuple: the first value will be the number

of comparisons and the second value will be the number of swaps.

Run the same tests you used in Exercises 1, 2 and 3. Compare the results from the four exercises. Do they support the claim that merge sort has better performance that bubble sort and selection sort? If not, what additional tests could you perform to prove or refute this claim?

Note: the textbook's `merge_sort` function is not the most efficient implementation of this algorithm, because the first thing each recursive call does is create two slices from the function's argument list. Each slice operation copies half the list elements to a new list, which increases the amount of memory used by the algorithm and increases the running time. You don't have to fix this, because, for this lab, we're only interested in determining the number of comparisons and swaps during the sorting.

A more space-efficient implementation of this function would require one "auxiliary" list with the same size as the original list. This is the biggest drawback to merge sort: despite its running time is $O(n \log n)$ (after we fix the slicing "problem"), merge sort may not be suitable when working with large lists.

Wrap Up

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

The submission deadline for this lab is Sunday, April 11, 23:55 (Ottawa time), for all lab sections.

To submit your lab work, go to the cuLearn page **for your lab section** (not the main course page). Submit your modified bubble_sort.py, selection_sort.py and merge_sort.py files. Ensure you submit the version of the files that contain your solutions, and not the unmodified file you downloaded from cuLearn! You are permitted to make changes to your solutions and resubmit the file as many times as you want, up to the deadline. Only the most recent submission is saved by cuLearn.

After completing this lab, start Assignment 3, which uses the modified sorting functions.

Last edited: April 6, 2021 (initial release)