University of St. Gallen

Institute of Computer Science

Fundamentals of Computer Science (3,125), HS2019
I. Mizutani, R. Seiger, D. Vachtsevanou, S. Mayer, B. Weber
iori.mizutani@unisg.ch

Assignment 3: Sorting Algorithms

Deadline: Oct 8, 2019; 12:00 CET

Introduction Sorting algorithms rearrange elements in a list in order, so it is much easier to search for something in a sorted list than an unsorted one. In this assignment, we will perform an empirical analysis of sorting algorithms based on Doubling Hypothesis: "What is the effect on the running time of doubling the size of the input?" [1].

Rationale In the lecture, we walked through Binary search, Insertion sort, and Merge sort. You will have a closer look at the algorithms by implementing it by yourselves, and discover what difference they yield by measuring the time of execution.

Prerequisites Understanding of classic sorting algorithms and the template files. You will work on the provided files:

- `sorter.py` contains `Sorter` class with 4 incomplete instance methods. You will complete those methods to perform the analysis on different sorting algorithms.
- `doubling_test.py` is a script to perform a doubling test with the `Sorter` class. **You don't need to make any change to this file.**

NOTE All the sorting methods of `Sorter` take an optional argument `reverse` – when `True` is passed, the method sorts a list in descending order, otherwise in ascending order by default. Also the returned sorted list from the sorting methods **MUST** be List [2] data type. For this assignment, you **MUST NOT** use any third-party library, package, or module like `numpy`. (The Python Standard Library [3] is OK.)

(1) Generate A Tuple of Random Integers (1 Point)

(a) Read the docstring of the instance method `Sorter.generate_new_tuple(n)` in `sorter.py` and implement the method.

(b) Run the script `doubling_test.py` without an argument to self-test if the method is implemented properly. The script should then perform a doubling test with "default" sorting method called `_default_sort()`[4] and you should see an output as shown below. You will figure out the meaning of the output by doing Task 2 – nothing to submit for this sub task.

[1] https://introcs.cs.princeton.edu/python/41analysis/
[2] https://docs.python.org/3/tutorial/datastructures.html#more-on-lists
[3] https://docs.python.org/3.7/library/
[4] The Python built-in `list.sort()` method: https://docs.python.org/3/library/stdtypes.html#list.sort

```
>>> %Run doubling_test.py

# Perform a doubling test for default sort with 100 trials
starting with n = 64
n:       64 time: 0.001257 (n:n/2) ratio: -
n:      128 time: 0.003101 (n:n/2) ratio: 2.466995
n:      256 time: 0.008518 (n:n/2) ratio: 2.746886
n:      512 time: 0.015068 (n:n/2) ratio: 1.769020
n:     1024 time: 0.024654 (n:n/2) ratio: 1.636171
n:     2048 time: 0.046303 (n:n/2) ratio: 1.878121
n:     4096 time: 0.095661 (n:n/2) ratio: 2.065970
```

(2) The `Sorter` Class and Doubling Test (0.5 Point each)

Study the `sorter.py` and `doubling_test.py`. In your `README.txt`, answer the following questions.

(a) Some instance methods of `Sorter` have the leading "_" (e.g., `_default_sort()`) and others don't (e.g., `set_algorithm()`). What do you think this convention mean? Explain it by examining `doubling_test.py` and looking up the PEP 8 Style Guide for Python Code [5].

(b) `Sorter._default_sort()` has "`lst = list(self.unsorted_tuple)`" at the beginning of its definition. Why do you think `unsorted_tuple` is defined as a tuple instead of a list?

(c) `Sorter.time_trials()` uses "`time()`". Look up the documentation at https://docs.python.org and explain what is the return value.

(d) In `Sorter.time_trials()`, explain the following part of the code mentioning `n`, `algorithm`, and `is_reverse`.

```
for i in range(n):
    is_reverse = (i%2 == 0)
    if self.algorithm == 'merge':
        self._merge_sort(is_reverse)
    elif self.algorithm == 'insertion':
        self._insertion_sort(is_reverse)
    elif self.algorithm == 'bubble':
        self._bubble_sort(is_reverse)
    else:
        self._default_sort(is_reverse)
```

(3) Insertion Sort (2 Points)

Implement the instance method `Sorter._insertion_sort()` with Insertion Sort algorithm.

(4) Bubble Sort (2 Points)

Implement the instance method `Sorter._bubble_sort()` with Bubble Sort algorithm.

(5) Merge Sort (2 Points)

Implement the method `Sorter._merge_sort()` with Merge Sort algorithm. As Merge Sort is a recursive algorithm, so you may need to add another helper method/function [6].

[5] https://www.python.org/dev/peps/pep-0008/

[6] You may write some piece of code outside the indicated block in the template.

(6) Test Doubling Hypothesis (1 Point)

Run the script `doubling_test.py` **with** an argument: `bubble`, `insertion`, and `merge`. Observe the results and compare. In your `README.txt`, put the results up to `n: 4096`, state the Doubling Hypothesis, and approximate the time increasing factor for each sorting algorithm.

Hand-in Instructions Create a file `README.txt` that includes 1.) answers to the questions, 2.) any reference that you used to complete this assignment, 3.) pitfalls you encountered and 4.) short explanations of your solution if necessary.

Compress the Python script `sorter.py` and the `README.txt` to a `zip` file "`assignment3.zip`". Upload the zip file to your exercise group's Course page on Canvas.