FIT1008 – Intro to Computer Science
Workshop Week 5

Semester 1, 2017

Objectives of this practical session

- To gain understanding of time complexity of simple algorithms.
- To learn how to compute running time for a program.

Task 1

(i) Write a Python function `sum_items(a_list)` in a new file called `sum_items.py`, which returns the sum of all the items of `a_list`, or zero if `a_list` is empty. (*You may assume that the items of the list are real numbers.*)

(ii) Compute the best and worst case time complexity for this new function and include this information in the documentation for this function.

Task 2

If you include the code:

```
import timeit
```

you can use the call `timeit.default_timer()` to compute the elapsed time as follows:

```
    start = timeit.default_timer()
    # do whatever you are doing that you need to time
    taken = (timeit.default_timer() - start)
```

1. Extend `sum_items.py` and write a Python function `time_sum_items(a_list)` that returns the time taken to call `sum_items`.
2. Write a Python function `table_time_sum_items()` that does the following:
    - For `n = 2, 4, 8, 16` and so on up to `1,024`
    - Creates a random list, `a_list` of reals between 0 and 1, whose length is `n`. (*You will need to import the module* random, *and use the functions* `random.seed()` *and* `random.random()`.)
    - Prints on a newline `n` and the value of `time_sum_items(a_list)`
3. Cut and paste the output from the previous stage into Excel and make a graph. Explain the shape of the graph. Is it what you expected? **Note**: When creating the graph, you will want to select the 2 columns of data and use a X/Y scatter graph.

Important: Don't forget to write your explanations down and submit them together with your graphs.

Task 3

1. Write a Python function `shaker_sort(a_list)` in a new file called `shaker_sort.py`, that implements *shaker sort* (see Tute 5 Exercise 2 for details).
2. Write a function `time_shaker_sort(a_list)`
3. Write a Python function `table_time_shaker_sort()` that does the following:
   - For `n = 2, 4, 8, 16,` and so on up to `1,024`
   - Creates a random list, `a_list` of reals between 0 and 1, whose length is `n`.
   - Prints on a newline `n` and the value of `time_shaker_sort(a_list)`
4. Cut and paste the output from the previous stage into Excel and make a graph. Explain the shape of the graph. Is it what you expected?
5. Write a Python function `table_avg_time_shaker_sort()` that does the following:
   - For `n = 2, 4, 8, 16,` and so on up to `1,024`
   - Creates `100` random lists, of reals between 0 and 1, whose lengths are `n`.
   - Prints on a newline `n` and the average value of `time_shaker_sort`
6. Cut and paste the output from the previous stage into Excel and make a graph. Explain the shape of the graph. Is it what you expected?

Important: Don't forget to write your explanations down and submit them together with your graphs.

Task 4

Using the experimental method above compare the time complexity of a tail recursive and a non-tail recursive version of a function to compute the Fibonacci numbers.

Extra challenge

Don't like excel to do the plots? Give *matplotlib* a chance:
`http://matplotlib.org/users/pyplot_tutorial.html`
If you have installed the Python tools following the video on Moodle, *matplotlib* should be part of your installed Python libraries.