FIT1008 Introduction to Computer Science
Practical Session 3

Semester 2, 2014

Objectives of this practical session

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

For all the programs you are required to provide documentation and testing for each piece of functionality.

Task 1 [3 marks]

(i) Write a Python function `sum_items(a_list)`, 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.

(iii) Write a function `test_sum_items()` that calls `sum_items` with at least two different test cases and extend your program to include the following code:

```python
if __name__ == "__main__":
    test_sum_items()
```

Task 2 [3 marks]

If you include the code:

```python
import time
```

you can use the call `time.time()` to compute the elapse time as follows:

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

1. 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:
   - Creates a random list, of reals between 0 and 1, whose length is greater than `1,000,000`. (*You will need to import the module* random, *and use the functions* `random.seed()` *and* `random.random()`.)
   - Prints a blank line (*This will make it easier to paste the results of the first loop into your graphing program or spreadsheet*).
   - For `n = 2, 4, 8, 16,` and so on up to at least `1,000,000`, print out on separate lines `n` and the value of `time_sum_items(a_list[:n])`

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?

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

Task 3 [4 marks]

1. Write a Python function `sum_until_negative(a_list)` that sums items in `a_list`, beginning at `a_list[0]` and ending *as soon as a negative number is reached*. If the first item of `a_list` is negative or `a_list` is empty, the function should return 0.
2. Write a function `table_time_sum_until_negative_1()` similar to the function you wrote for Task 2. Measure the running times and graph the results. Compare the graph with the one you obtained for Task 2 and explain their similarities and differences.
3. Now write a function `table_time_sum_until_negative_2()` similarly to the one above but, this time, make sure that the list you pass to `sum_until_negative()` *has a negative number in the first item of the list*. Measure the running times and graph the results. Compare the graph with the one you obtained for Task 2 and explain their similarities and differences. In particular, think about what do they mean in terms of best/worst cases.

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

Advanced Question [Bonus 2 marks]

Write the Python function `find_max_sum_interval(a_list)` to find the Max Sum Interval of the `a_list`: you want to find indices `i_min` and `i_max` such that

```
a_list[i_min] + a_list[i_min+1] + a_list[i_min+2] + ... + a_list[i_max-1] + a_list[i_max]
```

is as large as possible.

The function has to loop over all possible values of `i_min` and `i_max` working out the sum for each, and keeping track of the biggest you've seen so far. (Note that the `a_list` items are allowed to be negative, so it's not always best to just take the whole `a_list`.) When you have found `i_min` and `i_max`, you should return a list containing these two indices.

Once again, first estimate your time complexity, then time `find_max_sum_interval` as you did in Task 2, for `n = 2, 4, 8, 16,` and so on up to at least `100`, and graph your results.

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

Hall of Fame

Write a Python function `quick_find_max_sum_interval(a_list)` that implements a more efficient algorithm for finding the Max Sum Interval. Once you've written and tested the function, time it as you did in the Advanced Question, and graph your results.

- What is its worst-case complexity, in big-O notation?
- What about its best-case complexity?

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