IT5003 Oct-Dec 2023
Data Structures and Algorithms

Tutorial+Lab 01
Basic Python

Document is last modified on: October 2, 2023

1 Introduction and Objective

The purpose of this first tutorial+lab session (all onsite) is to recap the first two (or three, if you count the recitation 1 too) sessions of IT5003: Introduction, basic Python, basic analysis of algorithm, and to ensure that all students can code a simple Python program using their own computer/laptop at home and submit code to Kattis for automatic judging. The first half of the session is generally the 'tutorial' part and the second half of the session is generally the 'hands-on/lab' part. The tutors will control the timings and they don't have to divide the sessions exactly by half. There will be a short break during the transition.

As this is the first session, we will do a quick ice breaking at the start of the session.

To get the most out of the tutorial part of these sessions, please try out all the questions in the tutorial component and give some answer even if you encounter difficulties in answering some of them. Before, during, or after the tutorial session, don't hesitate to clear up all doubts and questions you might have, with the tutor.

Every week, you will try to solve one selected Kattis problem during the 'hands-on/lab' component. The tutors already know the selected Kattis problems for this semester. However, these selected problems will be revealed to you on the spot each week (if you happen to already solve it, then you are free to just leave the session or actually you can stay back to help your peers – you can learn more things by observing how others approached the problem that you have solved – possibly with another way). Tutor will guide all students to get (near) Accepted solution for each problem. These problems are not graded but attempting them during the hands-on time (and possibly to fully complete them afterwards) is beneficial to better understand IT5003 material.

The tutorial/lab participation marks are there to encourage class participation. These marks will be given by the tutor **at the end of the semester** using the following guideline:

- 0% if you only attend ≤ 4 out of 8 tutorial/lab sessions (likely 7 this semester due to long well-being day/weekend/Deepavali PH-in-lieu),

- 1% for **at most the bottom three** most-passive students (assuming these students attend > 4 tutorial/lab sessions),
- 3% for **at least the top three** most-active students (answering questions when asked by TA – the correctness of your answers are secondary; or even just by asking your own questions to TA before/during/after class/during consultation); in each tutorial group, and
- 2% for the rest.
- PS: For a few lab groups with ≥ 30 students, this can be bottom FOUR and top FOUR instead.

2 Questions

There is no specific tutorial questions this time, but full hands-on. See below.

Hands-on 1

TA will run this session with a few to do list:

- First, we will continue tinkering with `https://www.comp.nus.edu.sg/~stevenha/cs2040/demos/SpeedTest.py` to highlight a few more common time complexities,
- Next, this is the time to review the optional easy Python/coding challenges (PS0 at `https://nus.kattis.com/courses/IT5003/IT5003_S1_AY2324/assignments/amqr3d/standings`. This time, we also analyze the time complexities of our solutions.
- Finally, live solve one chosen Kattis problem involving basic Python.

In Lecture, Steven *usually* show the $O(N^2)$ (default), $O(N \log N)$ (replacing the inner-loop with the fast growing one), $O(N)$ (removing the inner loop), and $O(N^3)$ (adding one more inner loop) versions of `https://www.comp.nus.edu.sg/~stevenha/cs2040c/demos/SpeedTest.py`.

TAs are supposed to try showing a few more, e.g., the faster ones $O(1)$ (removing both loops), $O(\log N)$ (removing the outer loop), $O(N^4)$ (add yet one more inner loop), $O(2^N)$ `https://github.com/stevenhalim/cpbook-code/blob/master/ch3/cs/itertools2.py`, $O(N!)$ `https://github.com/stevenhalim/cpbook-code/blob/master/ch3/cs/itertools1.py`), etc. Note that for this semester, the itertools (combinations/permutations) and factorial can be important for PS1a task.

The solutions for all problems in PS0 are known to all TAs. TA can choose any subset of problem(s) to be re-discussed live with your live group. These are the worst case (Big O) time complexities of those 10 problems:

- Clearly $O(1)$

  - /metronome (simple $O(1)$ divide by 4.0),
  - /addingtrouble (simple $O(1)$ if-else, 2 cases),
  - /internationaldates (simple $O(1)$ if-else, 3 cases),

- Can be interpreted as $O(1)$ (terms and conditions[1] apply):
  - /codetosavelives ($O(t \times max(m, n))$, but all variables are very small),
  - /undeadoralive (simple $O(1)$ if-else, 4 cases, the string lengths are small, not more than 160 characters),

- $O(\log m)$:
  - /babypanda ($O(\log m)$ bits of integer m),

- $O(n)$:
  - /coffeecupcombo ($O(n)$, n is up to 10^5),
  - /electionparadox ($O(n)$, a bit of observation),

- Non-standard:
  - /sifferprodukt (maximum number of iterations is 5 (try $679 \to 378 \to 168 \to 48 \to 32 \to 6$) and each digit product function call is short),
  - /bokhyllor (hard to analyse, but on the smallest S and largest s, m, b (max 20 each), we will have at most 60 shelves, 'small'),

The hands-on for today is: `https://nus.kattis.com/problems/vaccineefficacy`

This is a nice simulation problem and it is very relevant during this COVID-19 pandemic (DORSCON green now...).

Problem Set 1

We will end the tutorial with high level discussion of PS1 A+B.

For /contingencyplanning, the problem is not that hard (just a bit mathematical), let's use $n = 2$ (Sample Input/Output 2 with the illustration). If there are $n = 2$ zombies, let's call them A and B, there exist *four* different ways that at least one can rise from the graveyard and attack. If only one of them appear (A only or B only), then there are 1 way each. If both of them appears together, then it can be A, B or B, A (two sub-ways), hence $2 \times 1 + 1 \times 2 = 2 + 2 = 4$. Can you generalize this formula? Try $n = 3$ yourself.

[1]Some 'low constants, rule of thumbs: still in hundreds in the worst case' problems may still be interpreted as $O(1)$ by some Computer Scientists. For our final paper, the time complexity requirements should be precise enough to reduce potential different interpretations, i.e., specify the time complexity in using certain specific variables.

For $n = 4$ (Sample Input/Output 3), the answer 64 is found via
$C(4,1) \times 1! + C(4,2) \times 2! + C(4,3) \times 3! + C(4,4) \times 4! = 4 \times 1 + 6 \times 2 + 4 \times 6 + 1 \times 24 = 4 + 12 + 24 + 24 = 64$.

The answer grows very fast, for a certain small n (thus we can pre-calculate the answers and have a three-liner solution), the answer will already be far larger than the limit and we print "JUST RUN!!".

For /cocktail, the problem is also not that hard, after we sort the time in non-increasing order, a simple greedy strategy should emerge (drink each potion from the one with the longest duration, and simulate the rest, hoping to get the "Furious Cocktail" status). We are not in advanced algorithm course, but this Greedy strategy can be proven to be always correct. The code for /cocktail is 'not that short' (half page), and plagiarism checking starts from this task onwards.