COMP2054-ADE: FORMATIVE LAB

"Implementing Vector, counting operations and doing an experimental amortised analysis"

Description

Firstly, you should download the file Main.java and Vector.java from Moodle.
Then:

A. "**Implement Vector**". In Vector.java
   1. Thoroughly read and understand the implementations. In particular, how to edit Vector::resize() operation to switch between incremental and doubling strategies
   2. Read/modify the "primitive operation counting" in the code – i.e. consider the lines of the form "c += ??" and ensure that you understand their motivations. Make sure that the total value of c is output after each push operation
   3. Note: if output the value of c after each push, then after having done n pushes, then the value of c will count as the T(n) of lectures. That is, a single run of doing many pushes, can produce all of the values of T(n); there is no need to do a separate experiment for each value of n.

B. "**Experiments**": Run experiments on incremental and doubling strategies

C. "**Graph**". Using the results of the experiments, for each strategy, plot some graphs of the costs of push as a function of the total number, n, of elements added to the Vector. The main point is to plot T(n), and also T(n)/n, as functions of n.
   Aim to show that the amortised cost is indeed O(1) when using the doubling strategy.

   Also, given the "weird functions represented as graphs", you should be able to relate them to the Big-Oh and Big-Omega definitions, and be able to argue in terms of the graphs for why they are O(1), O(n), Ω(1), Ω(n), Θ(1), Θ(n), etc. E.g. to show a function is O(1) you just need to be able to "*put a horizontal line above it, and that is above it – at least, for n >= some n0*". These lines to demonstrate the Big-Oh family behaviour should also be plotted/drawn on the graphs. The learning objective from this is to be able to link together the formal definitions with lines and interpretations of graphs – in particular to emphasise how the Big-Oh family is also very useful for classifying the scaling properties of "weird functions" that do not have a "nice smooth behaviour".

D. "**Extension**": Try other strategies for the new size on the resize operation. E.g. A constant increment in the size, or try trebling the size, etc. AIM: If you were designing a "Vector" for a library then what strategy would you use?

NOTES: The code is all given, only minor editing needs to be done, Hence, it is mostly a matter of just running to produce the data, and then copying it to Excel, and being able to plot a graph. To help, an example Excel file is give. But you are recommended to produce your own.