COMP1201 Assignment 1

(due 21st February, 4pm)

This assignment is worth 5% of the module marks. Submission is via the Handin system.

Q1 Consider the java program TestSort found here:

`https://secure.ecs.soton.ac.uk/notes/comp1201/assignment1/question1.shtml`

This program generates an array of 100 random doubles and sorts them using three different algorithms: Insertion sort, Shell sort and Quick sort (the java default), outputting the time taken by each algorithm.

(a) Modify TestSort to measure the running time on different sizes of arrays. Plot a graph of the average run time against the size of the input. For this, you will need to run the three algorithms for *several* arrays of the same size.

Note: Your answer should only include a brief summary of the modifications made to the code, and the graph mentioned above. (2 marks)

(b) If a program runs in $\Theta(n^a)$, (that is, $T(n) \approx c * n^a$), then the logarithm of the run time grows linearly with the logarithm of the size of the input. Take the logarithm of the run time and array size for the previous data and re-plot your graph as a log-log graph. (1 mark)

(c) Use the log-log graph to estimate the average-case time complexity of insertion sort. (1 mark)

(d) Estimate the average running time of insertion sort on an array of size 10^{10}. (1 mark)

Q2 Graph colouring is a classic problem described here:

`http://en.wikipedia.org/wiki/Graph_coloring`

Consider the Java program found here:

`https://secure.ecs.soton.ac.uk/notes/comp1201/assignment1/question2.shtml`

This program generates a random graph and then tries all possible colourings of the nodes. It then shows the best colouring possible.

(a) Write a new class to measure the average run time of this program for problems of size 12 to 17. Plot a graph of the logarithm of the run time against the problem size.

Note: Your answer should only include a brief description of your new class and the graph mentioned above. (2 marks)

(b) From your measurements, estimate the average-case time complexity of the graph-colouring solver. Explain your working in detail. Is the answer what you expected, and why? (3 marks)

Q3 This question considers a version of binary search trees which store elements of the form $\langle \texttt{key}, \texttt{value} \rangle$, with the `key` being used for comparisons (for the purpose of arranging elements in the tree), and with duplicate keys being allowed.

(i) A simple way to implement such a binary search tree is to modify the insert operation so that the new element is added to the left subtree when the key being added is below that of the root, and to the right of the subtree when the key being added is greater than *or equal to* that of the root. As with standard binary search trees, new elements are inserted by creating new leaves. What is the time complexity of inserting n elements with identical keys into an initially empty binary search tree? Explain your answer in detail. (1 mark)

(ii) Now consider a different version of the insert operation, which keeps a boolean flag at each node, and inserts an element with the same key either to the left subtree or to the right subtree, depending on the value of the flag. The value of the flag alternates between 0 and 1 every time a node is visited while inserting an element with the same key. What is the time complexity of inserting n identical elements into an initially empty binary search tree, with this modified insert operation. Explain your answer in detail. (2 marks)

(ii) Finally, consider yet another implementation of binary search trees which allow duplicate keys, which keeps a (singly) linked list of values associated to the same key, and inserts into the list. What is the time complexity of inserting n identical elements into an initially empty binary search tree, with this new implementation? Explain your answer in detail. (2 marks)