ELL781: Software Fundamentals for Computer Technology

Minor Test, Maximum marks: 35

Instructions:

- **Please clearly indicate the question number, and part number if applicable, at the start of each response.**
- **Please read all questions carefully.**
- **Please ensure that your responses are to-the-point and that you write only what is asked for on the answer script you submit.**
- **While the exam is open-notes, all your answers must be written entirely in your own words, without any copying from anywhere.**
- **Please try to be clear and careful with all mathematical notation, so that there is no ambiguity in the expressions/formulae you write down. Try to stick to the kind of notation used in class as far as possible.**

Section 1. Multiple Choice Questions

Each question may have any number of correct choices. List (in clear handwriting) the letters corresponding to all choices you believe to be correct (3 marks will be awarded for each correct choice you list, -1.5 marks for each incorrect choice). No justification is required.

1. Which of the following recurrences correspond to polynomial-time complexity?
   (a) $T(n) = 3T(n/2) + n(n-1)$; $T(1) = 1$
   (b) $T(n) = 2T(n-2) + 2$; $T(1) = T(2) = 1$
   (c) $T(n) = 8T(3n/4) + n^3$; $T(1) = 1$
   (d) $T(n) = T(n-1) + T(n-2)$; $T(1) = T(2) = 1$
   (e) $T(n) = nT(n/2)$; $T(1) = 1$
2. Consider an undirected graph with 7 vertices and 19 edges (no self-loops). Which of the following are true?
   (a) The graph can never be coloured with 5 colours.
   (b) The graph can be coloured with 5 colours if and only if it contains a node with degree 4.
   (c) The graph can be coloured with 5 colours if and only if it does *not* contain a node with degree 4.
   (d) The graph can always be coloured with 5 colours.

Section 2. Short Answer Questions

3. We discussed two definitions of $\Omega(g(n))$ in class, which are repeated below to aid your memory:
   Definition I: $T(n)$ is $\Omega(g(n))$ if $\exists c > 0$ such that $T(n) \geq cg(n)$ for infinitely many values of n.
   Definition II: $T(n)$ is $\Omega(g(n))$ if $\exists c > 0, n_0$ such that $\forall n \geq n_0, T(n) \geq cg(n)$.
   Consider

$$T(n) = \begin{cases} n^4 & \text{if } n \leq 100 \\ n^2 & \text{if } n > 100 \end{cases}$$

   Prove whether or not $T(n)$ is $\Omega(n^2)$, under each of Definition I and Definition II. **[4]**

4. Show that if $T(n) = T(2n/5) + T(3n/5) + cn$ (c is a constant), then $T(n)$ is $\Omega(n \log n)$, by:

   (a) Taking the given solution to be a guess and showing its correctness. **[3]**

   (b) Getting to a closed form via substitution (draw the recursion tree). **[3]**

5. Consider the below array.

   `[3, 1, 6, 4, 9, 7, 8, 2, 5]`

   (a) Suppose the quicksort partition function is called on this entire array, with pivot value `3`. Show the state of the array through the sequence of iterations in the partition function. For each iteration, the final locations of the two cursors (`i` and `j`) should be clearly indicated, and then the state of the array after the swap should be shown for the next iteration. **[3]**

   (b) What is the total number of comparisons that happen during the sequence of iterations of the partition function? And the total number of swaps? **[2]**

   (c) What is the final value (index) returned by the partition function? Assume the above array is indexed starting from `0`. **[1]**

   (d) Suppose we use mergesort instead of quicksort on the above array, making the first two recursive calls on the subarrays from index `0` to index `4`, and from index `5` to index `8`. Obtain the total number of comparisons that happen *only in the final merge* of these two subarrays, after each has been recursively sorted. **[2]**

   (e) Compare your answers to parts (b) and (d). Based on these, which algorithm appears to be more efficient in terms of the number of comparisons/swaps involved? Is this consistent with what was mentioned in class, in terms of which of the two has better average-case complexity? If not, why do you think that is? Is there some other kind of cost that is not being accounted for here? **[3]**

6. Consider a series of numbers $X_0, X_1, X_2, X_3, ...$ which are defined as follows:

$$
\begin{aligned}
&X_0 = 0; \\
&X_1 = 1; \\
&X_2 = 2; \\
&X_n = X_{n-1} - X_{n-2} + X_{n-3},\ \forall n \geq 3.
\end{aligned}
$$

   (a) Give a recursive algorithm for computing the n^{th} number in this series which directly makes use of the above recurrence relation. Write clear pseudocode, and analyse the time complexity of your algorithm. **[2]**

   (b) Illustrate the execution of the algorithm for $n = 5$, by drawing a recursion tree of all the recursive calls made, and how the values returned from the base cases are propagated up the tree. **[2]**

   (c) Based on the answer to (b), do you think this is the most efficient way to compute these numbers? If not, why not? Point out specifically where you think the inefficiency is. **[1]**