Special instructions: You may work either individually or in a group of exactly two persons. Write your solutions out on paper and deliver them to SSO by 14:00 on Friday, 18th October, 2013. If you are working in a group, one script should be submitted for the group, with both names at the top; both members of the group will then receive the same mark. Clearly write your name(s), student ID number(s) and the words “Comp36111 Sec. A Coursework” on the front (cover) sheet and staple all sheets together. Submitted solutions must be entirely the effort of the persons whose names appear at the top of the submission.

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Advanced Algorithms I: Coursework for Sec. A

Time: This should take you a few hours

Please answer all questions.
Marks will be awarded for clarity and succinctness as well as correctness.

The use of electronic calculators is not recommended.
1. The following algorithm computes the length of the shortest paths in a weighted directed graph $G$ with no negative cycles. The vertices of $G$ are the integers $0, \ldots, n-1$, and the weight on an edge from $i$ to $j$ is represented as the entry $A[i, j]$ of a 2-dimensional square matrix. If there is no such edge, then $A[i, j] = \infty$. The diagonal entries of $A$ are ignored. The length of the shortest path from node $i$ to node $j$ is represented as the entry $C[i, j]$ in an array $C$, which is returned.

\begin{verbatim}
begin allPairsShortestPath(A)
    n ← size(A)
    C ← an $n \times n$ array
    for $i = 0$ to $n - 1$
        for $j = 0$ to $n - 1$
            if $i = j$
                $C[i, j] \leftarrow 0$
            else
                $C[i, j] \leftarrow A[i, j]$
    for $h = 0$ to $n - 1$
        for $i = 0$ to $n - 1$
            for $j = 0$ to $n - 1$, $j \neq i$
                $C[i, j] \leftarrow \min(C[i, j], C[i, h] + C[h, j])$
    return C
end
\end{verbatim}

Evidently, allPairsShortestPath runs in time $O(n^3)$.

a) Prove that this algorithm is correct. (4 marks)

b) Explain, giving pseudo-code if necessary, how the algorithm can be adapted so as to return not only the length of the shortest path from $i$ to $j$, but also a path that achieves that minimum length, in a way which allows the path in question to be read off efficiently. (4 marks)

c) Could the algorithm be adapted to yield all paths achieving the minimum length, in case of ties? (2 marks)

2. Let a context-free grammar $G = \langle V, N, S, P \rangle$ be given, where $V$ is the set of terminals, $N$ the set of non-terminals, $S \in N$ the distinguished non-terminal and $P$ a set of productions. Assume that $G$ is in Chomsky-normal form, that is: every production is either
of the form $A \rightarrow BC$ or of the form $A \rightarrow a$, where $A$, $B$ and $C$ are non-terminals and $a$ is a terminal.

a) Using pseudo-code, write an algorithm which takes a non-empty string $x \in V^*$ as input and returns Yes if $x$ is accepted by $G$ and No otherwise. Your algorithm must run in time $O(|x|^3)$, where $|x|$ denotes the length of $x$.

(4 marks)

b) By modifying the algorithm if necessary, show how it can be made to return not only a judgement as to whether $x$ is accepted by $G$, but also an encoding of all the parse-trees which $G$ assigns to $x$. This modification must not compromise the cubic running time of the algorithm.

(4 marks)

c) Show that the grammar

\[
\langle \{a\}, \{A\}, A, \{A \rightarrow AA, A \rightarrow a\} \rangle
\]

yields at least $2^{n/2} - 1$ parse trees for the string $x = a^n$ with $n \geq 2$. (You may be easily able to obtain a higher bound!) Why does this not contradict your answer to the previous part of this question?

(2 marks)

3. Let $A$ be a matrix of dimensions $\ell \times m$ and $B$ a matrix of dimensions $m \times n$. The most straightforward algorithm for computing the product $AB$ employs approximately $2\ell mn$ arithmetic operations. Ensure that you understand why this is so. For the remainder of this exercise, we shall assume that this straightforward algorithm is employed to multiply a pair of matrices.

Let $A_1, \ldots, A_n$ be matrices such that, for all $i$ ($1 \leq i \leq n$), $A$ has dimensions $a_{i-1} \times a_i$. Thus, the matrix product $A_1 \cdots A_n$ makes sense, and has dimensions $a_0 \times a_n$. Of course, matrix multiplication is associative: it does not matter for the end result how the matrices are grouped in pairs for successive multiplication.

a) Suppose the matrices are multiplied in strict left-to-right order:

\[
((\cdots (A_1 A_2) A_3 \cdots) A_n).
\]

That is: first we multiply $A_1$ and $A_2$, then we multiply the result by $A_3$, etc., finally multiplying our computed value of $A_1 \cdots A_{n-1}$ by $A_n$. How many arithmetic operations are required?

(2 marks)
b) Suppose now the matrices are multiplied in strict right-to-left order:

\[ A_1 ( A_2 (A_3 \cdots (A_{n-1} A_n) \cdots ) \). \]

How many arithmetic operations are required this time?

(2 marks)

c) In general, a grouping of multiplications takes the form of a binary tree: each vertex represents some intermediate product \((A_i, \ldots, A_{j-1})\), with the left daughter representing a still smaller intermediate product \((A_j, \ldots, A_{h-1})\) and the right daughter the residual intermediate product \((A_h, \ldots, A_{j-1})\). Write an algorithm to output a grouping that \textit{minimizes} the total number of arithmetic operations required to compute \(A_1 \cdots A_n\), and explain why it runs in polynomial time (as a function of \(n\)).

(6 marks)