One and a half hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Algorithms and Imperative Programming

Date: Thursday 17th January 2013

Time: 09:45 - 11:15

Please answer any TWO Questions from the THREE Questions provided

Use a SEPARATE answerbook for each QUESTION

Marks will be awarded for reasoning and method as well as being correct.

This is a CLOSED book examination

The use of electronic calculators is permitted provided they are not programmable and do not store text.

1. Algorithm design.

   For each of the following computational tasks

   (i) describe an algorithm for the task. Your description may be a program in a standard language, in pseudocode, or a clear and precise step-by-step description. You should explain your algorithm briefly. Marks are awarded for a correct algorithm. Some marks will also be awarded for efficiency: the more efficient your algorithm is, the more marks it will be awarded.

   (ii) give the worst-case time complexity of your algorithm in terms of the size of the input and the number of operations required. Explain your answer.

   a) Given a list of integers as input, determine whether or not two integers (not necessarily distinct) in the list have a product k. For example, for $k = 12$ and list $[2,10,5,3,7,4,8]$, there is a pair, 3 and 4, such that $3 \times 4 = 12$. (6 marks)

   b) A *majority element* in a list of integers of length N, is an element that occurs strictly more than $N/2$ times in the list. Determine whether or not a list of integers has a majority element. (7 marks)

   c) Choosing k-items from a list of n distinct integers *at random and without repetition* (i.e. an item must not be chosen more than once). Assume you are given a function *random(m)* which chooses an integer at random between 1 and m. (7 marks)

2. Sorting

   a) What is the *worst case* time complexity of each of the following sorting algorithms in terms of the number of items N to be sorted? Give your answer in Big-Oh notation.

   (i) merge sort,
   (ii) bucket sort,
   (iii) quick sort,
   (iv) radix sort,
   (v) insertion sort,
   (vi) selection sort.

   (3 marks)

b) Look at the following pseudocode for a sorting algorithm.

Sorting algorithm (based on pseudocode from Wikipedia)

```
input: array of numbers A[0,...,N-1]
for i := 1 to i := N-1
{
    //comment: save A[i] to make a hole at index iHole
    item := A[i]
    iHole := i
    //comment: keep moving the hole to next smaller index until
              A[iHole-1] is <= item
    while iHole > 0 and A[iHole-1] > item
    {
        //comment: move hole to next smaller index
        A[iHole] := A[iHole-1]
        iHole := iHole-1
        }
    //comment: put item in the hole
    A[iHole] := item
}
output: A[] in sorted order
```

i) Does the algorithm sort the items into ascending or descending order? (1 mark)

ii) What type of input would lead to a best-case behaviour in terms of the number of assignments done? Give the precise number of assignments done in this case (do *not* use Big Oh). (2 marks)

iii) Can the sorting algorithm shown be described as 'in-place'? Explain your answer. (1 mark)

c) QuickSort and MergeSort are two Divide and Conquer algorithms for sorting.

i) For *each* algorithm, briefly describe how it recursively divides up the input array. (4 marks)

ii) Explain how the merge operation in MergeSort works, and give its time complexity. (4 marks)

iii) Which of the two algorithms cannot be implemented in place? (1 mark)

d) QuickSort is not a stable algorithm. However, if the number in $A[i]$ is changed to $A[i] \times n + i - 1$, then the new numbers are all distinct. After sorting, which transformation will restore the numbers to their original values? Explain your answer. (4 marks)

3. Complexity

a) Look at the following pairs of functions. In each case state whether (asymptotically) $f(n)$ grows faster, $g(n)$ grows faster, or they are of the same order. (Remember: asymptotically means for sufficiently large n).

For example, for the pair $f(n) = 5n$, and $g(n) = 10n$, the answer would be: THE SAME (since they are of the same *order*, both $O(n)$) .

i) $f(n) = n^2 + n$, and $g(n) = 6n^2 + 2$ (1 mark)

ii) $f(n) = n \log_2 n$, and $g(n) = n\sqrt{n}$ (1 mark)

iii) $f(n) = 2^{\sqrt{n}}$, and $g(n) = n^{20}$ (1 mark)

iv) $f(n) = \log \log n$, and $g(n) = \log n$ (1 mark)

b) Which of the following statements (A-E) are true about the four functions given below? In each case, indicate *all* the statements that apply.

A. The function has exponential growth
B. The function is constant, it does not depend on n
C. The function is $O(n^2)$ (Big-Oh of n^2, it grows at the same order or more slowly than n^2)
D. The function has linear complexity, it is $\Theta(n)$
E. The function is $\Omega(n^5)$ (it grows at the same order or faster than N^5)

i) 3^{10} (2 marks)

ii) $17n^2 + 5n + 4$ (2 marks)

iii) $2n \log n$ (2 marks)

iv) 3^n (2 marks)

c) Give *three* different reasons why computer scientists tend to give the time complexity of algorithms using Big-Oh notation, instead of e.g. quoting exact numbers of operations used. (Do not say ‘tradition’ or the ‘status-quo’). (3 marks)

d) The following is the celebrated Gale-Shapley algorithm for 'stable marriage' (which is actually used to assign guests to hotel rooms, places to university applicants, etc.) The input to the algorithm is a list of men M and an equal-sized list of women W. Moreover, each man knows which is his most preferred woman, second favourite, and so on. Each woman likewise has a ranking of the men.

```
function stableMarriage {
    Initialize all m ∈ M and w ∈ W to free
    while there exists a free man m who still has a woman w to propose to {
        w = m's highest ranked such woman to whom he has not yet proposed
        if w is free
            (m, w) become engaged
        else some pair (m′, w) already exists
            if w prefers m to m′
                (m, w) become engaged
                m′ becomes free
            else
                (m′, w) remain engaged
    }
    for each engaged man m and his fiancee w {
        marry (m, w)
    }
    output: complete list of married couples
}
```

Notice: in the algorithm, men sometimes propose to engaged women, and engaged women always swap if they prefer the proposer !

Give an argument that the algorithm stops (with everyone sure to be married), and use this argument also to give an estimate of the worst-case time complexity (using Big-Oh). (Hint: consider what happens to the women as the algorithm progresses).

(5 marks)