One and a half hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Algorithms and Imperative Programming

Date: Thursday 23rd January 2014

Time: 14:00 - 15:30

Please answer any TWO Questions from the THREE Questions provided

Use a SEPARATE answerbook for each QUESTION

This is a CLOSED book examination

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

[PTO]

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. 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 how you calculated your answer.

   a) Given a list of integers as input, determine whether or not two integers (not necessarily distinct) in the list have a **sum** k. For example, for list $[2,10,5,3,7,4,8]$ and $k = 17$, there is a pair, 10 and 7 such that $10+7 = 17$. (6 marks)

   b) The intersection of lists: Given two lists of integers, compute a list of integers which consists of those integers which appear in both of the two lists. The order of the resulting list does not matter and, if numbers appear several times in the lists, the result need not reflect this multiplicity - though it may. Thus, given lists $[2,5,3,8,2,4,7]$ and $[6,7,2,4,9,1]$, one possible intersection list is $[2,4,7]$. (7 marks)

   c) Consider an array of characters and a division point between two characters in the array. The task is to interchange the two parts of the array, before and after the division point, maintaining the order of characters in each of the parts. Thus if the array of characters is *abcdefg* and the division point is between *c* and *d*, then the interchange results in the array of characters *defgabc*. Extra marks will be awarded if you can do this interchange **in-place**, i.e. by swapping pairs of elements in the array. [Hint: One approach to an in-place algorithm is to consider how the operation of reversing an array may contribute to the interchange of parts of the array. A different approach to an in-place algorithm is to observe that interchanging parts of equal length can be done easily by swapping pairs of elements, and then using this operation repeatedly on smaller parts, if necessary.] (7 marks)

2. Sorting

   a) Which algorithm?

      i) Which sorting algorithm works by comparison of key values, can easily be implemented as stable, and has a worst-case complexity of $O(n \log n)$? (2 marks)

      ii) Which sorting algorithm can be implemented in-place, and might make use of a 'median-of-three' item ? (2 marks)

      iii) Which sorting algorithm is suited to items with keys that are strings of symbols from a limited alphabet? (2 marks)

   b) A lecturer at the University of Madchester thinks he has made his fortune by inventing the following randomized comparison-based algorithm for sorting an array A containing *n* items:

```
elegant_sort( A[ ], $n$)
{
    while (true)
        $i$ := the value from an $n$-sided fair dice roll;
        $j$ := the value from an $n$-sided fair dice roll;
        if $i > j$
            swap $i$ and $j$;
        if A[$i$] > A[$j$]
            swap A[$j$] and A[$i$];
        end while
}
```

      i) There is a specific reason why the pseudocode does NOT describe a (correct) sorting algorithm. What is it? (1 mark)

      ii) After some time, the array A will almost certainly be sorted. Show that the expected (or average) number of comparisons to reach a sorted array is $O(n^3)$ on a standard unsorted input. Hint: begin by considering how long it takes for the smallest key to reach (and fix at) its correct position. (3 marks)

      iii) The lecturer claims that independent of the input, the algorithm *may* sort the array in $O(n)$ time. Verify if this is true, showing your reasoning. (2 marks)

   c) Many sorting algorithms are based on pairwise comparison of key values.

      i) In order for a comparison-based sorting algorithm to work properly on a set of items X, three conditions must generally hold for all a, b and $c \in X$. Name or describe **two** of them. (2 marks)

ii) Give an expression (using aymptotic notation) for the Lower Bound of the Running time of Comparison-Based Sorting. (2 marks)

iii) Briefly justify this lower bound. (4 marks)

3. Algorithm Analysis

a) **Definition:** a function $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c.g(n)$ for all $n \geq n_0$, where n and n_0 are integers.

i) By the definition above, which of the following functions of n are $O(n^{10})$? List all that apply.

A. $50n^2$ B. $n \log n$ C. 2^n D. $n!$ E. $10,000$

(2 marks)

ii) By the definition above, which of the following functions of n are $O(n^{1/2})$? List all that apply.

A. $3\sqrt{n}$ B. $\log n$ C. $n/100$ D. $20n/(\sqrt{5n})$ E. $0.01^{n/2}$

(2 marks)

iii) The big theta notation, $f(n)$ is $\Theta(g(n))$, allows us to say that two functions $f(n)$ and $g(n)$ are asymptotically equal, up to a constant factor.
Which of the following functions are $\Theta(n)$? List all that apply.

A. $5n^2$ B. $n \log n$ C. $n + 100$ D. $15n$ E. $2^{\log_2 n}$

(2 marks)

b) An algorithm has a running time of $O(n.m^2)$ for input parameters $n \geq 1$ and $m \geq 1$. Answer TRUE of FALSE to the following questions

i) The running time of the algorithm can be described as 'polynomial-time'. (1 mark)

ii) For n constant, the running time of the algorithm can be described as 'constant'. (1 mark)

iii) For m constant, the running time can be described as 'linear in n'. (1 mark)

c) Below is some pseudocode for an efficient method for calculating x^n for integer x, and nonnegative integer n.

```
binary_exponentiation($x$, $n$)
1: if($n = 0$)
2:     return 1;
3: else if($n = 1$)
```

```
4:     return $x$;
5: else if $n$ mod $2 = 0$
6:     return binary_exponentiation($x * x$, $n/2$);
7: else
8:     return $x *$ binary_exponentiation($x * x$, $(n-1)/2$);
```

i) Work out the number of muliplications the algorithm does to compute 5^{10}. Show your working. (2 marks)

ii) Use Big-Oh to express the asymptotic running time in terms of the number of multiplications done by the algorithm. Show your working. (3 marks)

iii) How much space is used by the algorithm, expressed in big-Oh? Explain your answer briefly. (1 mark)

d) A stack is an abstract data type (ADT) that implements two operations, push (putting something on the stack) and pop (removing something from it). The push and pop operations can be combined into a single operation called popush(n, v), which takes two arguments, $0 \leq n \leq current\_stack\_depth$, the number of items to be popped off the stack, followed by v (which may be null), an item to be pushed onto the stack.

i) What is the asymptotic worst-case time complexity of the operation popush(n, v) (assuming either an array or a linked-list impementation of the stack) ? (1 mark)

ii) What is the amortized worst-case complexity of popush(n, v) for any series of n operations, starting from an empty stack? Justify your answer. You may use an informal answer, or a more formal amortized analysis technique. (3 marks)

iii) In general, what is the difference between amortized analysis and average-case analysis? (1 mark)

END OF EXAMINATION