UNIVERSITY OF BRISTOL

January 2015 Examination Period

FACULTY OF ENGINEERING

Second Year Examination for the Degrees of BSc, BEng and MEng

COMS21103J
Data Structures and Algorithms

TIME ALLOWED:
2 hours

This paper contains three questions:
All questions carry 20 marks each
All three questions will be used for assessment
The maximum for this paper is 60 marks

Other Instructions
1. Calculators must have the Engineering Faculty seal of approval.

TURN OVER ONLY WHEN TOLD TO START WRITING

Q1. a) Express each row in the following table as $f(n) = \Delta(g(n))$ by choosing the best applicable bound (Ω, $\mathcal{O}$, or Θ) to replace Δ. You do NOT need to prove your answer.

	$f(n)$	$g(n)$
i)	$n^{\log\log n}$	n^2
ii)	$n^2 + 100n$	n^3
iii)	$n \log n$	$\log(n^n)$
iv)	2^n	2^{2n}

[4 marks]

b) This question is about sorting algorithms:

i) What measures in MergeSort should be taken (and where) to ensure that the algorithm is stable?

ii) You have to sort a small array which is almost sorted (at most two elements are out of place). Which one of the following algorithms is the most suitable for this task?

- Selection Sort - Insertion Sort - QuickSort

iii) What is the running time of QuickSort (that sorts in an ascending order) when the pivot is the first element and the input contains distinct elements sorted in an ascending manner?

[4 marks]

c) An algorithm A solves a problem of size n in time $\Theta(n^2)$. Algorithm B for the same problem solves a problem of size n by dividing it into 16 subproblems of size $\frac{n}{4}$, recursively solving each of them, and then combining the solutions in time $\mathcal{O}(n^2)$.

i) What is the recurrence for the time complexity of algorithm B? *[1 marks]*

ii) What is the time complexity of algorithm B? Clearly state the method you used and the steps involved. *[4 marks]*

iii) Which of the two algorithms is asymptotically better? *[1 marks]*

d) Given the following specifications:
Precondition: $x, y \geq 2$.
Postcondition: $y > 5$.
Why does the following code segment not satisfy total correctness?

```
While x ≥ 2
    x := x − 2
    y := y + 3
```

[2 marks]

e) Clearly describe a $\Theta(n \log n)$-time algorithm (you do not have to give it in pseudocode) for determining whether or not an array $A[1..n]$ of *distinct* natural numbers (excluding 0) has any elements $A[i]$ and $A[j]$ satisfying $A[i] = 3 * A[j]$. Clearly show that your algorithm is $\Theta(n \log n)$. *[4 marks]*

A1. a) i) $f(n) = \Omega(g(n))$.

ii) $f(n) = \mathcal{O}(g(n))$.

iii) $f(n) = \Theta(g(n))$.

iv) $f(n) = \mathcal{O}(g(n))$.

b) i) In the merge algorithm when comparing equal elements, preference should be given to items from the left (i.e. the lower indexed) sub array. *[2 marks]*

ii) Insertion Sort. Because in this case it runs in $\mathcal{O}(n)$. *[1 marks]*

iii) $\Theta(n^2)$ or just $\mathcal{O}(n^2)$. *[1 marks]*

c) i)

$$T(n) = 16\frac{n}{4} + \mathcal{O}(n^2)$$

or

$$T(n) = 16\frac{n}{4} + n^2$$

ii) Using the Master Method: This is Case 2: $a = 16$, $b = 4$, $f(n) = n^2$, $n^{\log_b a} = n^{\log_4 16} = n^2$. Clearly, $n^2 = \Theta(n^2)$. Thus, the answer in this case is $T(n) = \Theta(f(n) \log n) = \Theta(n^2 \log n)$.
Remark: Solutions clearly using a recursion tree or the iteration method are also fine as long as the answer is $\Theta(n^2 \log n)$ or $\mathcal{O}(n^2 \log n)$.

iii) Algorithm A because it has a better asymptotic running time.

d) Because it is not partially correct. The input $x = 2$, $y = 2$ is a valid input that satisfies the precondition, but the output $y = 5$ does not satisfy the postcondition.

e) We make use of a second array $B[1..2n]$ of size $2n$. In the first n positions in B, i.e. $B[1..n]$ we copy the elements of the original array A. We fill the remaining n elements of B, i.e. $B[n + 1..2n]$ by setting $B[n + i] := 3 * A[i]$. This has linear complexity, i.e. $\Theta(n)$. We sort B by calling MergeSort, which will cost $2n \log 2n = \Theta(n \log n)$ work. Now, all is left is to scan B (in linear time) to see if two consecutive elements are equal. Note that A has distinct elements so if we find equal elements in B we return True. Otherwise, we return False. Clearly, the time complexity of the algorithm is $\Theta(n \log n)$ which is the cost needed for calling MergeSort.

Q2. a) For a large unsorted array, you need to find the 90-th percentile (i.e. the value x such that 90% of the numbers are less than or equal to x). A colleague of yours produced an algorithm that sorts the array *using MergeSort* first in ascending order, then retrieves the m^{th} element where $m = \lceil \frac{9n}{10} \rceil$.

i) What would be the time complexity of your colleague's algorithm? *[1 marks]*

ii) Recall that there is median finding algorithm that uses random partitioning and for every input runs in linear time on average. Using a similar approach, write a pseudo code to find the 90-th percentile of an array. Comment your code for readability where needed. *[6 marks]*

iii) Discuss the time complexity of your algorithm including the potential worst case. *[2 marks]*

b) You were asked to design an algorithm that efficiently searches for a binary string of 10 bits in a folder of binary files. The algorithm should find all occurrences of the binary string within all the files in the folder. Explain whether you would (or would not) use each of these algorithms to search for the binary string.

i) Finite State Machine

ii) Boyer-Moore-Horspool

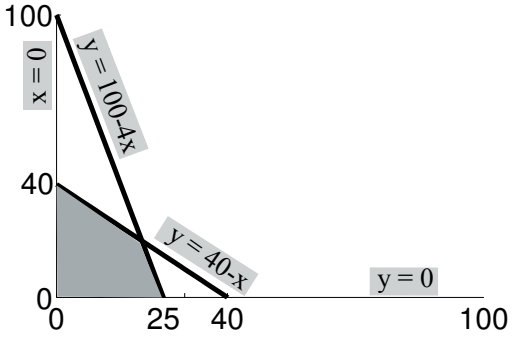
[4 marks]

c) Consider the linear program

$$\begin{array}{ll} \textbf{minimise} & -2x + -y \\ \textit{subject to} & x + y \leq 40 \\ & 4x + y \leq 100 \\ & x \geq 0,\ y \geq 0 \end{array}$$

i) Use hand calculations to solve this problem by the simplex method. Clearly show the steps of your solution. *[5 marks]*

ii) Copy an approximation of the figure below onto your answer sheet, and describe graphically the path you followed in (i) to find the solution. *[2 marks]*



A2. a) For order statistics:

i) $O(n \log n)$

ii) The algorithm should choose a pivot, then swap elements around the pivot so all elements to the left are smaller and all elements to the right are larger.

If the right partition is of size $\frac{n}{10}$, end.

If the right partition of size greater than $\frac{n}{10}$, recursively choose a pivot from the right partition.

If the right partition of size $m < \frac{n}{10}$, recursively choose a pivot from the left hand size and search for the largest $\frac{n}{10} - m$ elements.

– The students can choose their preferred psudo-code style.

iii) The algorithm is linear in best time. In the worst case we would partition the array into one element to one side, leaving an array of size $(n-1)$. In this case. $T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$.

b) For string matching:

i) Yes, particularly for a small alphabet (as in the case of binary strings 0,1), the finite machine can encode the within-pattern dependencies so the files are scanned once.

ii) No, this algorithm is suitable for large alphabets. It won't be helpful in the case of binary strings.

c) For linear programming:

i) In standard form:

$$\begin{aligned} z &= 2x_1 + x_2 \\ x_3 &= 40 - x_1 - x_2 \\ x_4 &= 100 - 4x_1 - x_2 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Initial solution $(x_1, x_2, x_3, x_4, z) = (0, 0, 40, 100, 0)$

Replace x_1 by x_4, the tightest constraint

$$\begin{aligned} z &= 2(25 - \tfrac{1}{4}x_4 - \tfrac{1}{4}x_2) + x_2 \\ x_3 &= 40 - (25 - \tfrac{1}{4}x_4 - \tfrac{1}{4}x_2) - x_2 \\ x_1 &= 25 - \tfrac{1}{4}x_4 - \tfrac{1}{4}x_2 \\ z &= 50 - \tfrac{1}{2}x_4 + \tfrac{1}{2}x_2 \\ x_3 &= 15 + \tfrac{1}{4}x_4 - \tfrac{3}{4}x_2 \\ x_1 &= 25 - \tfrac{1}{4}x_4 - \tfrac{1}{4}x_2 \end{aligned}$$

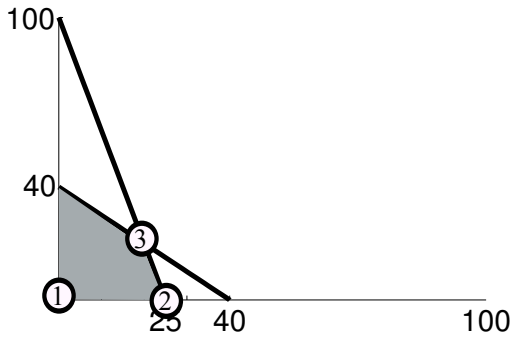
solution $(25, 0, 15, 0, 50)$.

Replace x_2 by x_3, the tightest constraint

$$\begin{aligned} z &= 60 - \tfrac{1}{3}x_4 - \tfrac{2}{3}x_3 \\ x_2 &= 20 + \tfrac{1}{3}x_4 - \tfrac{4}{3}x_3 \\ x_1 &= 20 - \tfrac{1}{6}x_4 - \tfrac{1}{3}x_3 \end{aligned}$$

solution $(20, 20, 0, 0, 60)$. The objective function cannot be increased further without breaking the nonnegativitiy constraints.

ii) Initial solution is at (1). Second solution (25,0) is at (2). Final solution (20,20) is at (3).



Q3. a) This question is about choosing the right algorithm for computing shortest paths in different types of graphs. In each of the following scenarios state which graph algorithm you would use to minimise the worst case time complexity. You should give the time complexity of your chosen algorithm in terms of the number of vertices, V and edges, E in the graph.

i) The graph is unweighted (equivalently all edges have weight one). You want to find the shortest path between two vertices given in the input. *[2 marks]*

ii) The graph has positively and negatively weighted edges. You want to find the shortest path between two vertices given in the input. *[2 marks]*

iii) The graph has at most $10V$ edges. You want to find the shortest path between every pair of vertices. In this part, for full marks you should also give the time complexity just in terms of V (simplifying where possible). *[2 marks]*

b) This question is about skip lists and focuses on the INSERT and FIND operations. You can ignore any other supported operations for the purposes of this question.

i) Briefly describe the structure of a skip list and explain how the operations INSERT and FIND are implemented. You may wish to draw a diagram to aid your explanation. *[6 marks]*

ii) What is the expected time complexity of these operations? *[1 marks]*

iii) Is the following statement true or false? Justify your answer. *[2 marks]*

"Skip lists are randomised therefore there are some sequences of operations which skip lists always perform poorly on"

c) Your friend tells you that they have designed a priority queue which supports INSERT, DECREASEKEY and EXTRACTMIN operations in $O(\sqrt[3]{n})$ time. Here n is the number of elements in the priority queue.

i) What is the time complexity of Dijkstra's algorithm if it is implemented using this priority queue? Give your answer in terms of the number of vertices, V and edges, E in the graph. Justify your answer. *[4 marks]*

ii) Your friend tells you that they have just improved the complexity of the DECREASEKEY operation to be $O(1)$ time. How does this affect the answer you gave in i)? *[1 marks]*

A3. a) i) Breadth First Search. The time complexity is $O(V + E)$.

ii) Bellman-Ford. The time complexity is $O(VE)$.

iii) Johnson's algorithm. The time complexity in terms of V and E is $O(VE + V^2 \log V)$ when implemented with a Fibonacci heap. When implemented with a binary heap its time complexity is $O(VE \log V)$ (either answer is fine). In either case when $E \leq 10V$ this simplifies to $O(V^2 \log V)$.

b) i) A skip list is comprised of a number of linked lists called levels. The elements in each level are sorted from the smallest to the largest. The bottom level contains every element in the skip list. The second level up contains some (typically about half) of the elements from the bottom level. The third level up contains contains some of the elements (again typically about half) from the second level up. In general, the elements in any level are always a subset of the elements in the level below.

Each element is linked to the next element in the same level and to the representation of itself in the level below.

The FIND operation works as follows. Start at the smallest end of the highest level (the one with the fewest elements). Move right through the level (which is a linked list) and stop when either you find what you wanted or you reach a element which is bigger than the one you are looking for. In the latter case, drop down a level by following the link from the element where you are. Continue this process until you find what you were looking for. If you reach an element which is larger than what you are looking for but are on the bottom level. The element you seek isn't in the skip list at all (but you have found the predecessor and successor).

The INSERT operation works as follows. First perform the FIND operation. As Assuming the element isn't already present you will find the predecessor and successor as commented above. Insert the new element into the bottom level between the predecessor and successor (which is easy as it's a linked list). We now need to decide whether to insert the new element into the next level up. We decide this by flipping a coin. If it is a heads, we insert, if not we stop. We continue this process repeatedly (inserting into higher and higher levels) until we first see a tails.

ii) Both operations take $O(\log n)$ expected time where n is the number of elements in the skip list.

iii) False. The randomness in the Skip list data structure comes from flipping coins to decide whether to promote an inserted element. The probability of poor performance is completely unrelated to the values of the elements inserted.

c) i) Dijkstra's algorithm INSERTs one element into the queue for each vertex at the start (and never performs any more inserts). It also performs one DECREASEKEY operation for each edge and performs EXTRACTMIN once for every vertex. As there are V INSERTs, we have that n above equals V (and not E). As all operations have the same cost, this is dominated by the E DECREASEKEY operations. The overall time complexity is therefore $O(E\sqrt[3]{V})$.

ii) With the improved DECREASEKEY operation, the complexity becomes

$$O(V \cdot \sqrt[3]{V} + E) = O(V^{4/3} + E)$$

END OF PAPER