Department of Computing and Information Systems
COMP20007 Design of Algorithms
Semester 1, 2014
Mid Semester Test

Instructions

- **Do not open this paper until instructed to do so.** You may read this page now.
- You may fill out your name and student number now.
- You must have your student card on display during this test.
- The test will start at 11:10pm and finish at 11:45pm.
- The total time allowed for this test is 35 minutes.
- This is a closed book exam. You should **not** have any study notes of any kind, including electronic (no calculators, phones, etc).
- Any student seen looking at their phone (or similar) or another student's paper during the test will have their paper removed immediately, and will be referred to the School of Engineering for a breach of academic honesty.
- Throughout you should assume a RAM model of computation where input items fit in a word of memory, and basic operations such as $+ - \times /$ and memory access are all constant time.
- Answer all questions on this paper.
- Remember, -1 mark for an incorrect answer in Question 1 True/False (advised not to guess).

Name: _____

Student Number: _____

Question 1 [9 marks, minimum 0 marks]

Answer True or False for each of these statements. You will gain one mark for a correct answer, and **lose** one mark for an incorrect answer.

1.	$f(n) = \log(10n)$ is in $O(n^2)$	
2.	$f(n) = n^2$ is in $O((\log n)^2)$	
3.	$f(n) = n$ is in $\Omega(\sqrt{n})$	
4.	If the oracle makes no errors, then the MOBS algorithm will always ask more questions than binary search on an input range that is bigger than 10.	
5.	Inserting a new item into a heap of n items requires $\Theta(\log n)$ time	
6.	A Huffman tree, if stored explicitly with left and right pointers, requires $\Omega(n^2)$ space.	
7.	A union-find structure based on linked lists can be engineered to allow $\Theta(1)$ time for union.	
8.	An algorithm that can divide in $O(\log n)$ time, and conquer in $O(n)$ time will have a worst case running time that is in $\Omega(1)$.	
9.	A brute-force implementation of the greedy set cover algorithm on an input with m sets, n distinct items, and N total items will require $O(nN)$ time.	

Question 2.1 [3 marks]

Write a recurrence relation that describes the running time of the following (not very smart) algorithm. The input is an array, `A`, and some constant integer `k`. Assume a comparison based sort in Step 2.

```
my_divide_and_conquer(A, k)
    0. Let n be the number of elements in A
    1. If n <= k, return A
    2. Sort A.
    3. Copy A[k..n-1] into a new array C
    4. Call my_divide_and_conquer(C, k)
```

What is a tight upper bound on the worst case running time of the algorithm?

Question 2.2 [3 marks]

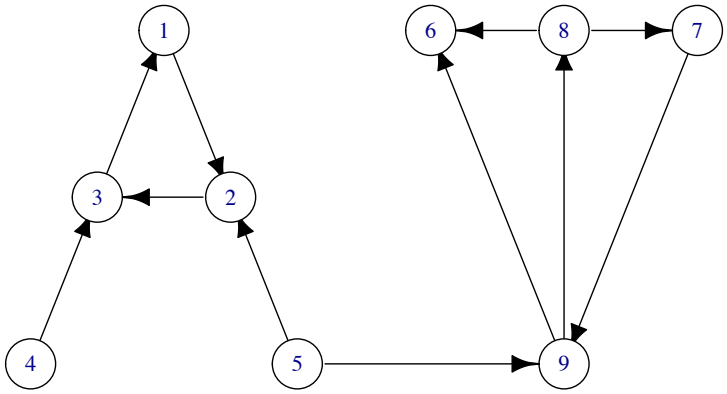
Write a recurrence relation that describes the running time of the following (also not very smart) algorithm. The input is an array, `A`, and some constant integer `k`. Assume a comparison based sort in Step 2, and that `n/k` rounds down to the nearest integer.

```
my_divide_and_conquer(A, k)
    0. Let n be the number of elements in A
    1. If n/k <= 1, return A
    2. Sort A.
    3. Copy A[1..n/k] into a new array C
    4. Call my_divide_and_conquer(C, k)
```

What is a tight upper bound on the worst case running time of the algorithm?

Question 3 [8 marks]

Label this graph with the pre and post numbers that would be assigned with Depth First Search. Assume the numbers begin at 1, and always prefer a lower labelled vertex if there is a choice.



Which edge is a back edge, and which is a cross edge?

If all edges in the graph were undirected and of weight 1, draw a possible Minimum Cost Spanning Tree for the graph.

Question 4 [4 marks]

Draw the Huffman tree for the symbol weights 1,1,3,4,4,7.

Assuming that their are n sorted input weights, and Huffman's algorithm is implemented using a sorted linked list of subtrees as in the Workshop, give big-O for the following.

Finding the two smallest subtrees in the list	
The total time over the whole algorithm for inserting new subtrees into the list.	

Question 5 [3 marks]

The algorithm we have discussed for finding Strongly Connected Components (SCCs) uses the highest post numbered vertex from a DFS of the reverse of the input graph to find a sink SCC in the input graph. Why is it necessary to reverse the graph: why not just take the lowest post number in the original graph? (Hint: show a graph where this doesn't work - include pre and post numbers of DFS.)