UNIVERSITY OF ABERDEEN SESSION 2012/2013

Examination in CS2521 Algorithmic Problem Solving

29 May 2013 09.00 – 11.00

Candidates are not permitted to leave the examination room during the first or last half hours of the examination.

Answer TWO questions. Each question is worth 25 marks; the marks for each part of a question are shown in brackets.
Note that there are some helpful formulae and theorems in an appendix to the paper.

1. (a) Prove that the following algorithm results in a sorted list of elements after running:

```
1: function bubbleSort(list)
2:     for i = n down to 1 do
3:         for j from 1 to i − 1 do
4:             if list[j] > list[j+1] then
5:                 swap the values of A[j] and A[j+1]
6:             end if
7:         end for
8:     end for
9: end function
```

[4]

(b) Prove by induction that $\sum_{i=1}^{n} i = n(n+1)/2$ **[4]**

(c) Find a simple $g(n)$ such that $f(n) = \Theta(g(n))$ for $f(n) = \sum_{i=1}^{n}(5)(5^i)$ **[4]**

(d) Describe the following pseudocode fragment in summations, and simplify them as far as you can. Give an exact upper and lower bound to the complexity of this algorithm (in terms of n)?

```
1: for all i=1 to n do
2:     for all j=i to n do
3:         output "hello world"
4:     end for
5: end for
```

[3]

(e) Determine whether 1.1^n dominates $n \log n$ or vice-versa. **[1]**

(f) Describe how to modify any balanced tree data structure so that search, insert, delete, minimum, and maximum each take $O(\log n)$ time, but successor and predecessor now each take $O(1)$ time. Which operations must be modified to support this? Provide the pseudocode for these modified operations. **[5]**

(g) When running randomised Quicksort, how many times do we call the randomisation function in the worst case? In the best case? **[4]**

2. (a) Provide pseudo-code to reverse an array of length n *in place*. What is the complexity of your code? What would the total complexity be if you were to use a singly-linked list as the underlying data structure of your algorithm? Is there a faster way to reverse a singly-linked list? If so, provide the pseudocode for this algorithm, if not, explain why not. **[4]**

(b) What is a stable sort? When is stability important? **[2]**

(c) The pivot operation within quick sort normally takes $\Theta(n)$ time. Is it theoretically possible to find a way to speed up the pivot to take time $\log n$ without altering the rest of the algorithm? Why/Why not? What if the new pivot would take $\Omega(\log n)$ time instead? **[3]**

(d) Can the recurrence $T(n) = 2T(n/4) + \sqrt{n}$ be solved via the Master method? If so, solve it, if not, show why not. **[3]**

(e) Recall the 0-1 Knapsack Problem: given integers $0 < w_1, \ldots, w_n, W$ and $0 < v_1, \ldots, v_n$, choose $I \subseteq \{1, \ldots, n\}$ to maximize $\sum_{i \in I} v_i$, subject to the constraint $W \geq \sum_{i \in I} w_i$. State the Fractional Knapsack Problem. **[3]**

(f) Using pseudocode, give a greedy algorithm that solves the Fractional Knapsack Problem. Make sure that you explain all variables used. **[7]**

(g) Give an example that illustrating that the standard greedy algorithm which solves the Fractional Knapsack Problem can fail to produce the optimal solution on an instance of the 0-1 Knapsack Problem. **[3]**

3. Throughout this question, you should assume that the graphs discussed are weighted and directed, with no loops, multi-edges or negative edge weights.

(a) Why are adjacency lists very often a better representation of graphs than adjacency matrices? **[3]**

(b) Here is Dijkstra's algorithm for pathfinding, with a fagment of code, marked '...', removed:

```
function DIJKSTRA(G,w,s)
    for all v ∈ vert(G) do
        d[v] = ∞
        π[v] = null
    end for
    d[s] = 0
    S = ∅
    Q = ∅
    Q = INSERT-ALL(Q,vert(G))
    while Q ≠ ∅ do
        u = EXTRACT-MIN(Q)
        S = S ∪{u}
        ...
    end while
end function
```

The argument G here is a graph, w is an array holding its edge weights for G, and s is the source vertex.

i. Explain precisely the problem that Dijkstra's algorithm solves. **[1]**

ii. Explain the roles of d, π, Q and S. **[6]**

iii. What does it mean to *relax* an edge from some vertex u to some vertex v? When is it *eligible*? **[2]**

iv. Prove that a necessary condition for a solution to the problem from part i. is that there are no remaining eligible edges to be relaxed. **[4]**

v. Supply the missing lines of code to Dijkstra's algorithm. **[5]**

vi. Explain why Dijkstra's algorithm becomes equivalent to Breadth-First Search when each edge has the same weight. **[4]**

Appendix

The following formulae and theorems may be of use to you.

Summations

$$\sum_{i=1}^{n} i = n(n+1)/2 \qquad \sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6 \qquad \sum_{i=0}^{n-1} ra^i = r(1-a^n)/(1-a)$$

Logarithms

$$\log_a 1 = 0 \qquad x^{\log_x a} = a \qquad x = y \leftrightarrow \log_a x = \log_a y \qquad \log_a xy = \log_a x + \log_a y$$

$$\log_a x^y = y \log_a x \qquad \log_a x/y = \log_a x - \log_a y \qquad \log_a x = \log_b x / \log_b a$$

Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ be a function, and $T(n)$ defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

Where n/b includes $\lfloor n/b \rfloor$ and $\lceil n/b \rceil$. Then T(n) has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

END OF EXAM PAPER