Ollscoil na hÉireann
The National University of Ireland

Coláiste na hOllscoile, Corcaigh
University College, Cork

End-of-Term Examination 2010
Practice Exam

CS4407 Analysis of Algorithms

Prof. G. Provan

Attempt all questions

Total marks: 100

60 minutes

Please answer all questions
Points for each question are indicated by [xx]

1. [**20**] f(n) and g(n) are asymptotically positive functions. Prove or disprove the following:
   $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.

We must prove that $f(n) \leq c\, g(n)$ for some positive constant c and $n0 > n$.
Consider the first relation, *i.e.*, $f(n) = O(g(n))$. *For this we can take* $f(n) = 2n$ *and* $g(n) = n$, and the relation must hold.
Now, consider the second relation, , *i.e.*, $2^{f(n)} = O(2^{g(n)})$. *We must show that* $2^{f(n)} \leq c\, 2^{g(n)}$.
For the left-hand-side, we have $2^{f(n)} = 2^{2n} = 4^n$.
For the right-hand-side, we have $2^{g(n)} = 2^n$.
There is no positive constant c such that $4^n \leq c\, 2^n$
Hence the claim is false.

2. [**20**] Write the most efficient algorithm you can think of (in C, Java, pseudo-code) for the following:
   a. Given an array of n integers A[n], return a sorted sub-list of the k smallest integers, where $1 < k < n$.
   b. What is the running time in terms of big-oh, big-theta, or big-omega? Explain your answer.

There are many approaches. Consider a quicksort variation: we need not recursively sort partitions which only contain elements that would fall after the k^{th} place in the end. Thus, if the pivot falls in position k or later, we recur only on the left partition:

```
function quicksortFirstK(list, left, right, k)
    if right > left
        select pivotIndex between left and right
        pivotNewIndex := partition(list, left, right, pivotIndex)
        quicksortFirstK(list, left, pivotNewIndex-1, k)
        if pivotNewIndex < k
            quicksortFirstK(list, pivotNewIndex+1, right, k)
```

The algorithm takes an expected time of $O(n + k\log k)$, and is quite efficient in practice, especially if we substitute selection sort when k becomes small relative to n.

3. **[20]** Write out an algorithm that takes as input a directed acyclic graph $G=(V,E)$ and two vertices v and z, and returns a path from v to z in G. What is the complexity of this algorithm?

We can use a variant of depth-first search (DFS) to compute this path.
The complexity of the algorithm is the same as that of DFS, i.e, $\Theta(|V|+|E|)$.

```
Algorithm pathDFS(G, v, z)
  setLabel(v, VISITED)
  S.push(v)
  if v = z
    return S.elements()
  for all e ∈ G.incidentEdges(v)
    if getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if getLabel(w) = UNEXPLORED
        setLabel(e, DISCOVERY)
        S.push(e)
        pathDFS(G, w, z)
        S.pop(e)
      else
        setLabel(e, BACK)
  S.pop(v)
```

4. **[20]** Given a graph G and a minimum spanning tree T, suppose that we decrease the weight of one of the edges not in T. Give an algorithm for finding the minimum spanning tree in the modified graph. What is the complexity of this algorithm?

Call the edge whose cost we decreased e=(u,v), and its new cost is w*. The only possible change resulting from this decrease is that e may now belong to the new MST, and instead another edge should be thrown out.

Method 1:

First, we first test if the value of $w^* > w_k$, where w_k is the most expensive edge in T; if so, then e will not be included in T and we just return T. Else, we test if e can replace some edge in T, possibly in a cycle induced by including e. First, we must test the cycle property Let T be the MST of G before the cost decrease. Then, we run BFS in time O(n) to find the unique path connecting u and v on T; call it P. Once we have found the path, we can check in time O(n) whether any edge on P has cost more than w*. If so, we replace the most expensive edge of P with e to obtain the new MST. Otherwise, we know that even after the cost decrease, e should not be part of the MST by the cycle property (because it is most expensive on at least one cycle, namely $P \cup \{e\}$).

Method 2:

First, we first test if the value of $w^* > w_k$, where w_k is the most expensive edge in T; if so, then e will not be included in T and we just return T.

Else,

Set $T' = T - \{a \in T \mid w(a) > w^*\}$, where a is an edge in T, and w(a) is the weight of edge a.

Starting with T', run Kruskal's algorithm to generate the new MST that will augment T'.

5. **[20]** The input consists of n skiers with heights $p_i, ..., p_n$, and n skis with heights $s_1, ..., s_n$. The problem is to assign each skier a ski to minimize the AVERAGE DIFFERENCE between the height of a skier and his/her assigned ski. That is, if the skier i is given the ski ai, then you want to minimize:

$$(|p_i - s_{ai}|)/n$$

(a) Consider the following greedy algorithm. Find the skier and ski whose height difference is minimized. Assign this skier this ski. Repeat the process until every skier has a ski. Prove or disprove that this algorithm is correct.

This algorithm is INCORRECT for the problem of minimizing the average difference between the heights of skiers and their skis. We provide a counter-example to show that it is not correct.

Consider an instance with the following values:

$p1 = 5$; $p2 = 10$; and $s1 = 9$; $s2 = 14$.

The algorithm would pair $p2$ with $s1$ and $p1$ with $s2$ for a total cost of $1=2(1 + 9) = 5$. Pairing $p1$ with $s1$ and $p2$ with $s2$ yields a total cost of $1=2(4 + 4) = 4$.

(b) Consider another greedy algorithm. Give the shortest skier the shortest ski, give the second

shortest skier the second shortest ski, give the third shortest skier the third shortest ski, etc. Prove or disprove that this algorithm is correct.

This algorithm is CORRECT. The proof is by contradiction. Assume the people and skis are numbered in increasing order by height. If the greedy algorithm is not optimal, then there is some input $p_1,\ldots, p_n; s_1,\ldots, s_n$ for which it does not produce an optimal solution. Let the optimal solution be $T = \{(p_1; s_{j(1)}),\ldots,(p_n; s_{j(n)})\}$, and let the output of the greedy algorithm be $G = \{(p_1; s_1),\ldots, (p_n; s_n)\}$. Beginning with p_1, compare T and G. Let p_i be the first person who is assigned different skis in G than in T . Let s_j be the pair of skis assigned to p_i in T. Create solution T_0 by switching the ski assignments of p_i and p_j . By the definition of the greedy algorithm, $s_i \le s_j$: The total cost of T_0 is given by

$Cost(T_0) = Cost(T) – 1/n(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|)$:

There are six cases to be considered. For each case, one needs to show that $(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|) \ge 0$:

Case 1: $p_i \le p_j \le s_i \le s_j$.

$(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|) = (s_j - p_i) + (s_i - p_j) - (s_i - p_i) - (s_j - p_j) = 0.$

Case 2: $p_i \le s_i \le p_j \le s_j$.

$(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|) = (s_j - p_i)+(p_j - s_i)-(s_i - p_i)-(s_j - p_j) = 2(p_j - s_i) \ge 0$:

Case 3: $p_i \le s_i \le s_j \le p_j$.

$(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|) = (s_j - p_i)+(p_j - s_i)-(s_i - p_i)-(p_j - s_j) = 2(s_j - s_i) \ge 0$

Case 4: $s_i \le s_j \le p_i \le p_j$.

$(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|) = (p_i - s_j) + (p_j - s_i) - (p_i - s_i) - (p_j - s_j) = 0.$

Case 5: $s_i \le p_i \le s_j \le p_j$.

$(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|) = (s_j - p_i)+(p_j - s_i)-(p_i - s_i)-(p_j - s_j) = 2(s_j - p_i) \ge 0$

Case 6: $s_i \le p_i \le p_j \le s_j$.

$(|p_i - s_j| + |p_j - si| - |p_i - si| - |p_j - s_j|) = (s_j - p_i)+(p_j - s_i)-(p_i - s_i)-(s_j - p_j) = 2(p_j - p_i) \ge 0$