Ollscoil na hÉireann
The National University of Ireland

Coláiste na hOllscoile, Corcaigh
University College, Cork

Quiz1
Practice Exam

CS4407 Analysis of Algorithms

Prof. G. Provan

Attempt all questions

Total marks: 60

50 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 $f(n) \le c\, g(n)$ for some positive constant c and $n_0 > n$.
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* $2^{f(n)} \le c\, 2^{g(n)}$.
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 $4^n \le c\, 2^n$
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)
```