CSL 630 Analysis and Design of Algorithms

Minor I, Sem I 2014 -15 Max 40, Time 1 hr

Name _____ **Entry No.** _____ **Group**

Note (i) The answers must be written in the space provided under each question including the back of the sheet. Do all roughwork in separately provided paper.

(ii) Write your answers neatly and precisely. You won't get a second chance to explain what you have written.

(iii) Every algorithm must be accompanied by proof of correctness and a formal analysis of running time and space bound. Feel free to quote any result from the lectures without proof - for any anything new, you must prove it first.

1. Consider the following method for multiplying two non-negative integers a, b represented in base 2. If $b = 0$ or $a = 0$ then mult $(a, b) = 0$, else mult (a, b) = mult $(2a, b/2)$ if b is even, else a + mult $(a, b-1)$.
   (i) Prove the correctness of this approach **(5)** By induction on b. Base case $b = 0$. Trivial.
   Suppose true for $b < n$. Then for $b = n$, consider two cases -
   b is even : $mult(a, b) = mult(2a, b/2) = 2a \times b/2 = a \times b$
   b is odd :$mult(a, b) = a + mult(a, b-1) = a + a \times (b-1) = a \times b$.
   Most solutions either didn't use induction or didn't use it properly. The proof of the correctness of this recursive algorithm cannot be done directly.

   (ii) Assuming that multiplying/dividing by 2 takes $O(1)$ time (using shifting and appropriate data structure), what is the bit complexity of this procedure ? **(10)**
   Let $|a| = m$, $|b| = n$ and let $T(m, n)$ be the time to multiply an m bit number by n bit number using *mult*.

   $$T(m, n) \leq T(m+1, n-1) + O(n+m)$$

   This is by considering the cases b is odd followed by $b-1$ is even ($mult(a, b) = mult(2a, (b-1)/2) + a$) and the case b is even. The cost of addition an $O(n+m)$ bit number which is the length of the result. You can now verify that $T(m, n) = O(n(m+n))$ which is $O(n^2)$ for $m = n$.

2. Given a set S of n numbers (may not be integers), determine if there are elements $a, b, c \in S$ such that $c = b + c$. For example, if $S = \{2.5, 3.4, 4.5, 6.8, 7.9, 10.0\}$, then $a = 3.4, b = 4.5, c = 7.9$ is one such tuple. Describe an $O(n^2)$ algorithm for this problem. **(10)**
   Sort the given n elements in $O(n \log n)$ time - denote the sorted sequence as $x_1, x_2 \ldots x_n$.Consider an $n \times n$ array A where $A_{i,j} = x_i + x_j$ (the array is not stored explicitly). Note that the elements are increasing across rows and columns since they were defined on the sorted elements.

   For each element x_i, we want to determine if $x_i = x_{i_1} + x_{i_2}$ and this is done as follows. Given any element X (not necessarily any of the x_i), to check if X occurs in the array A, we will sequentially locate two elements $A_{1,t} \leq X \leq A_{1,t+1}$ in the first row in $O(n)$ steps. If X equals either, we are done, else, we start searching in the second row of A upto column t since $A_{2,t} \geq A_{1,t}$. We continue this way till we find X or we have reached the last row. Since we are searching in non-overlapping columns, the total search time is $O(n)$. So to search for all x_i, we need at most $O(n^2)$ steps.
   This algorithm can also be thought of as moving three pointers in the sorted array.
   Many answers were along the right lines but didn't prove why it was sufficient to search the array in a certain order (increasing for one index and decreasing for the other). Some marks were deducted.
   About 5 marks were given for answers having $O(n^2 \log n)$ running time (with proof of correctness).
   No marks were given for brute force $O(n^3)$ solution.

3. A weighted directed graph $G = (V, E)$ $|V| = n, |E| = m$ has a weight function $w(i, j) = 2$ if $i > j$ and 1 otherwise (no self loops). The vertices are labelled as $1, 2, \ldots n$.

(i) Which of the known SSSP algorithm would you use for this problem ? **(2)**
Dijkstra since weights are non-negative and it is the fastest.

(ii) Design an $O(n+m)$ SSSP algorithm for this weight function. **(13)**
We can implement Dijkstra's algorithm without the normal heap data structure. Note that all the path lengths are between $1..2n$, We maintain an array of size $2n$, where $A[i]$ contains a list of all vertices with label i. As weight labels change, when we relax edges, we insert it in the corresponding list. To find the smallest label, we maintain a pointer to location j, if j is the length of the most recent shortest path output by Dijkstra's algorithm. The next shortest paths must be of length j or more. If $A[j]$ is empty we find the next non-empty location for the next smallest label. The total work is clearly $O(m+n)$.
Marks were given for solutions using BFS with proper proof of correctness. Note that we can convert the graph to G' where an edge with weight 2 was converted to a sequence of two edges with weight 1 with a new vertex in between.
Many tried to use topological sort that doesn't work for general directed graphs.
Some of the answers went in the wrong direction by assuming that the graph is complete since weights were defined as $w(i,j)$. But this is not for all i,j and only for edges that are present in the graph. This interpretation makes the problem trivial since the graph becomes a fixed graph and then there is no need to design an algorithm.