COMP 361 Exam Solutions

Trimester 2, 2016

1. Divide and Conquer [30 marks]

a) (10 marks) How many lines, as a function of n (in $\Theta()$ form), does the following program print? You may assume n is a power of 2.

From the given function we can see that the base case occurs when $n \leq 1$, and therefore $n_0 = 1$. At each time step in the normal case, we recurse on two sub problems each of size $n/2$. Therefore, a recurrence for the given function is:

$$T(n) = \begin{cases} \Theta(1), & n \leq n_0 = 1 \\ 2T(\frac{n}{2}) + \Theta(1), & n > n_0 \end{cases}$$

Since this recurrence relation is of the form $T(n) = aT(n/b) + f(n)$, we can determine an asymptotic cost for the recurrence using the Master Theorem (given on Page 2). Let's begin by defining some intermediary parameters:

- $a = 2$
- $b = 2$
- $f(n) = \Theta(1)$
- $\alpha = \log_b a = \log_2 2 = 1$

With these parameters defined, consider the three cases of the master method:

1. Applies if $f(n) \in O(n^{\alpha-\epsilon})$ for some $\epsilon > 0$.
2. Applies if $f(n) \in \Theta(n^{\alpha})$. Since $f(n) = \Theta(1) \notin \Theta(n)$, this case doesn't apply here.
3. Applies if $f(n) \in \Omega(n^{\alpha+\epsilon})$. Similarly, this case doesn't apply here.

We can see that for $\epsilon = \alpha = 1$, $f(n) = \Theta(1) \in O(n^0) \in O(1)$. Therefore case 1 says that $T(n) \in \Theta(n^{\alpha}) \in \Theta(n^1) \in \Theta(n)$. Since $T(n) \in \Theta(n)$, we can conclude that the given function will print out n lines.

b) (10 marks) You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time $O(n \log n)$.

This can in fact be solved in linear time by employing smart use of hash tables, but the fact that the question specified a bound of $O(n \log n)$ gives us a rather large hint that the marker expects some sort of sorting based algorithm.

Consider the following proposal:

1. Sort the elements using merge sort

2. Perform a linear pass through the sorted array appending the element at index i in the sorted array to an intermediary array only if it isn't equal to the element that directly preceded it in the sorted array i.e. the element at index $t-1$

The first step of this algorithm takes $O(n \log n)$ (refer to CLRS for a proof of this). The second requires $\Theta(n)$ to scan through the sorted array. We know that in asymptotic analysis we drop lower order terms, and so $O(n \log n) + \Theta(n) \approx O(n \log n)$.

Note that for full marks you should a) propose an $O(n \log n)$ solution and b) provide a brief discussion as to why this algorithm has a running time of $O(n \log n)$.

c) (10 marks) (Hard) Suppose you are given matrices A, B, C which are each n by n and you wish to check whether $AB = C$. You can do this on $O(n^{\log_2^7})$ steps using Strassen's algorithm. In this subquestion we will explore a much faster $O(n^2)$ randomised test.

(i) Let v be an n-dimensional vector whose entries are randomly and identically chosen to be 0 or 1 (each with probability 1/2). Prove that if M is a non-zero n by n matrix (i.e. at least one element is not a zero), then $Pr[Mv = 0] \leq 1/2$.

As an introduction, let's assume that every element in M is non-zero, and then work our way down to the requirement that only one element need not be zero.

For all positive M, the only situation in which $Mv = 0$ arises if v is an n-dimensional vector of all zeros. Why? Assume not. That is, that $Mv = 0$ and there is at least one element of v that is equal to 1. At some point in our matrix multiplication process, we would have multiplied this positive element in v with a positive element in M (by our assumption that M is all positive). The product of two strictly positive elements is strictly positive. By way of matrix multiplication being the sum of products along rows and columns, this contradicts the assumption that $Mv = 0$.

Therefore, when all elements in M are strictly positive, the only situation when $Mv = 0$ is when v is an n-dimensional vector of all zeros. Since the elements of v are identically and independently distributed, $P(v = (0, \ldots, 0)) = P(0) \cdot \ldots P(0) = 1/2 \cdot \ldots 1/2 = (1/2)^n < 1/2$ and therefore $Pr(Mv = 0) < 1/2$.

Now we wish to consider the case where only one element in M is non-zero. Let's denote this strictly positive element m_{ij}, the j-th element in row i of M. By way of matrix multiplication, the only element of v that will be multiplied with m_{ij} is element v_j, the j-th element in v. If $v_j = 0$, $Mv = 0$. Similarly, if $v_j = 1$, $Mv \neq 0$. Since each element in v is independent and identically distributed, $Pr(v_j = 0) = 1/2$, implying that $Pr[Mv = 0] = 1/2$ when only one element of M is considered to be non-zero.

Comparing the results of the first part of our proof with the results of the second part, we can see that as the number of non-zero elements in M increases, the probability of $Mv = 0$ decreases. As such, $Pr[Mv = 0] \leq 1/2$.

(ii) Show that $Pr[ABv = Cv] \leq 1/2$ fi $AB \neq C$. Why does this give an $O(n^2) randomised test for checking whether $AB = C$?

2. Greedy Algorithms [30 marks]

(a) [15 marks] Give an efficient algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied.

Consider a graph $G = (E, V)$, where each variable $x_i \in V$ (i.e. $V = \{x_i | i = 1, \ldots, n\}$). There exists an edge $e_{ij} = (x_i, x_j) \in E$ if and only if the equality $x_i = x_j$ or $x_j = x_i$ is presented as part of the input set. By the constraints of the question, we know that $|E| \leq m$ and $|V| = n$.

An efficient greedy algorithm is as follows:

1. Initialise the graph with an empty edge set and no vertices
2. For each equality (x_i, x_j) in the input set:
   - Add both x_i and x_j as vertices in the graph if they haven't already been added
   - Add an undirected edge $e_{ij} = (x_i, x_j)$ to E
3. For each inequality (x_i, x_j) in the input set:
   - If either of $x_i, x_j \notin V$, skip this iteration as by construction of the graph an equality (or chain of equalities) between x_i and x_j cannot exist
   - Begin a depth first search from x_i
   - If we reach x_j in this depth first search, return false as this inequality is inconsistent with the specified set of inequalities
4. Return true

(b) [10 marks] Argue or prove that your algorithm is correct.

The framework for proving greedy algorithms correct that was presented during lectures doesn't fit especially well with the algorithm proposed above. Instead we will perform a more constructive proof.

(c) [5 marks] State the asymptotic cost of your algorithm and justify why it's correct.

Constructing the graph requires inserting at most n variables and at most m edges. Each variable and edge can be inserted in constant time using hash tables to represent the graph, giving the cost of this step as $O(n) + O(m)$. In the worst case our algorithm performs a depth first search for every inequality constraint in the input set. Each depth first search costs at most $O(|v| + |E|)$ i.e. $O(n + m/2)$, so we have a total running cost of $\Theta(n) + \Theta(m) + O(nm + m^2/2)$.

This is a rather crude analysis due to the fact that in the worst case that all m of our input constraints are inequalities, our graph will contain no edges and therefore each depth first search will take $\Theta(1)$ time. However, this analysis does give us an upper bound on the asymptotic cost of the proposed algorithm.

3. Dynamic Programming [30 marks]

4. Guest Lectures [10 marks]

Omitted.

5. Approximation Algorithms (Hard) [20 marks]