Jeff Hungerford
A22129599
1/28/2000

1.

A)

A technique for multiplying two $n \times n$ matrices with $O(n^3)$ processing elements is given in section 9.1.3 of the textbook. This technique does not operate in constant time because the summation at the end is not of constant time.

A technique for multiplying two $n \times n$ matrices with $O(n^3)$ processing elements in constant time with the assumption that concurrent reads and concurrent writes cause no problem is the following:

Start with 3 $n \times n$ matrices: A and B which hold the input data, and a matrix C which is initialized to 0.

Processing elements are addressed as nodes in a three-dimensional space, $P[x, y, z]$. Each processing element $P[x, y, z]$ preforms the following action in parallel $C[y, z] = C[y, z] \vee A([x, y] \wedge B[z, x])$.

B)

The technique for multiplying two $n \times n$ matrices with $O(n^3)$ processing elements given in section 9.1.3 of the textbook functions under the Exclusive Read / Exclusive Write constraint, and operates in $O\log n$ time.

C)

The boolean matrix multiplication technique given in part A of this question can be used to construct an algorithm capable of preforming transitive closure on a $n \times n$ boolean matrix in $O((\log n)^2)$ time with $O(n^3)$ Processing elements.

as described in section 9.3.1 of the text, we preform the following steps:

First, we OR the input matrix with the $n \times n$ identity matrix

next, we use the technique given in part A to multiply the input matrix with itself $\log n$ times.

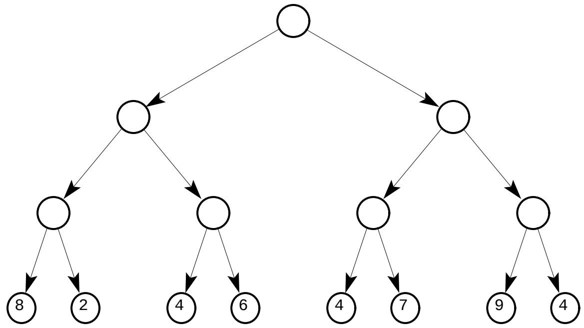
The result, after this computation is the transitive closure of the input matrix.

2.

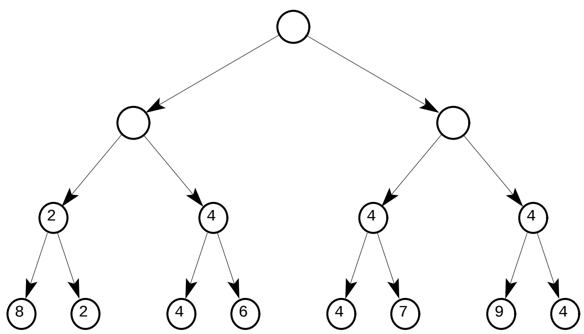
A technique for transposing a matrix of two dimensional date which is stored in a hypercube structured array of processing elements is given in section 9.2 of the text book.

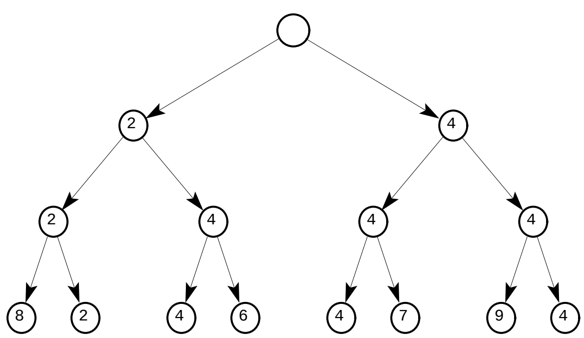
3.

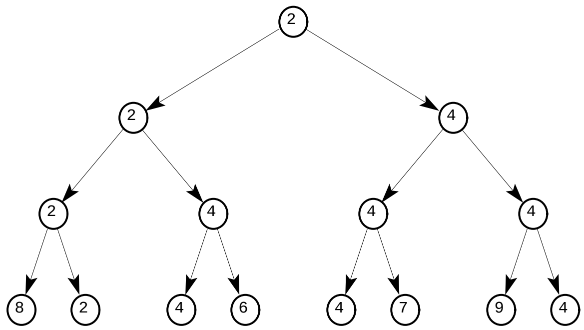
In the following figure, you can see the initial state of the data. The following fig-



ures show the movement of the data through the tree.







4.

I am unable to come up with a $O(\log n)$ sorting algorithm using $O(n^2)$ Processing elements. As far as I can tell, the best that can be done with $O(n^2)$ Processing Elements is $O(n\log n)$ time. I did find a solution for constant time, using $O(n^3)$ processing elements in the textbook.

5.

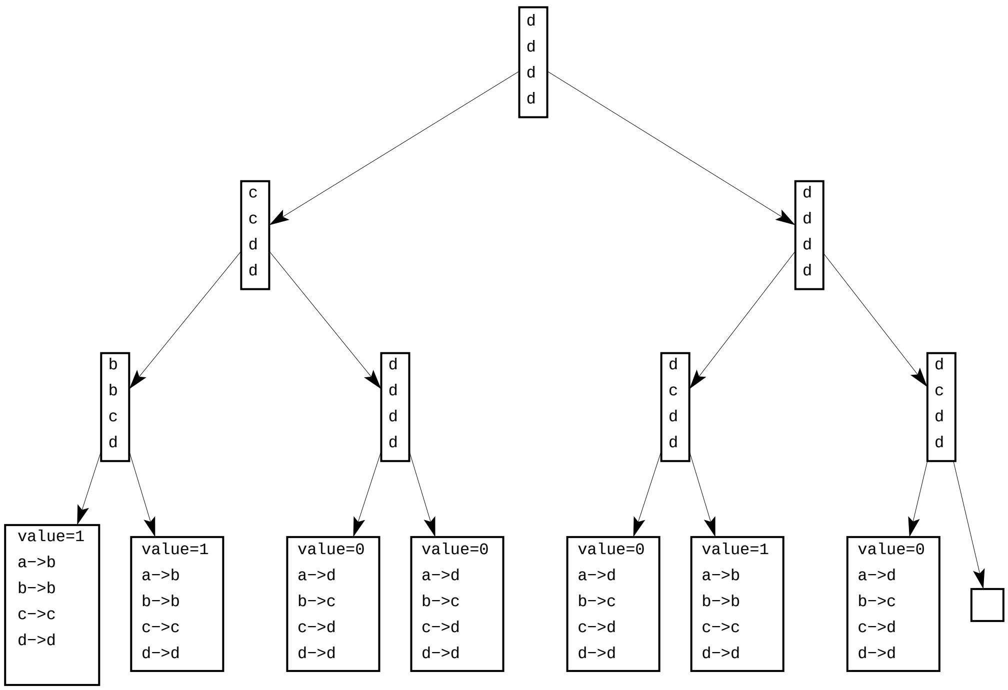
In a perfect shuffle based communication network of size n, one node is (on average) $\log n$ hops away from any other node.

Assuming that on a hypercube a computation takes t time to complete, and that for a message to travel down one link between processors takes one time unit, every time a signal would have to be sent from one node to another, instead of taking 1 time unit in the hypercube, it would take $\log n$ time units in the shuffle network. As the computational time is the same, the total time taken by the shuffle network would be $tn + \log n$.

6.

A)

B)



C) Yes, this can be preformed in $O(\log n)$ time using $O\left(\dfrac{n}{\log n}\right)$ processing elements, as I have done this in section B above.