

CSE 101 Homework 6
Spring, 2013, Due THURSDAY, May 16
Divide-and-Conquer
80 points total = 10 %

Binary Conversion The following recursive algorithm uses the divide and conquer method to convert an n bit binary integer $x_{n-1}...x_0$ into decimal. It uses the $O(n^{\log_2 3})$ time divide-and-conquer multiplication algorithm *Multiply2* from class and the text; and the grade school linear time ($O(n)$) *Add* algorithm as sub-routines. We assume *Add* and *Multiply* are defined to take decimal integers as input and output. Note that 2^n , in binary, is a 1 followed by n 0's, so is easy to construct as a binary integer in linear time. Let *ConstructPower2*, given n , construct 2^n in binary in time $O(n)$.

ConvertToDecimal($x_{n-1}...x_0$: Binary integer represented as an array of bits): decimal integer;

1. IF $n = 1$ return x_0 .
2. $y \leftarrow x_{n-1}...x_{n/2}$
3. $z \leftarrow x_{n/2-1}...x_0$
4. $w \leftarrow \text{ConstructPower2}(n/2 - 1)$ (in binary)
5. $a \leftarrow \text{ConvertToDecimal}(y)$
6. $b \leftarrow \text{ConvertToDecimal}(z)$
7. $c \leftarrow \text{ConvertToDecimal}(w)$
8. $c \leftarrow \text{Add}(c, c)$
9. $d \leftarrow \text{Multiply2}(a, c)$
10. $e \leftarrow \text{Add}(d, b)$
11. Return e

First, give a proof that this algorithm is correct, by strong induction (5 points). Second, give a recurrence for the time this algorithm takes (5 points). Third, solve the recurrence to give a time analysis for this algorithm (5 points). Finally, think of a modification to this algorithm that would improve its running time (5 points, somewhat tricky).

Least Common Ancestor: 20 points Consider the following recursive algorithm that takes as input a binary tree T .

Each non-leaf in T , x , has left-child $x.left$, and right child $x.right$, and each non-root has parent $x.parent$. (Child pointers at leaves and the parent pointer at the root return NIL). It uses a depth-first search procedure *DFS* that is linear-time in the size of the sub-tree and returns the list of

nodes in the sub-tree. It computes, for each pair of nodes x and y in T , the deepest node that is an ancestor of both x and y , and stores it in an array $LCA[x, y]$. The main idea is that if x is in the left sub-tree of the root, and y is in the right sub-tree, then the only common ancestor of x and y is the root. Otherwise, the least common ancestor is in the subtree that contains both x and y .

LeastCommonAncestor(r : node)

1. $LCA[r, r] \leftarrow r$
2. IF $r.left \neq NIL$ THEN
 3. $LeastCommonAncestor(r.left)$;
 4. $L_1 \leftarrow DFS(r.left)$;
5. IF $r.right \neq NIL$ THEN
 6. $LeastCommonAncestor(r.right)$;
 7. $L_2 \leftarrow DFS(r.right)$.
8. FOR each $x \in L_1$
9. FOR each $y \in L_2$
10. $LCA[x, y] \leftarrow r$

First, give a recurrence relation for the time of this algorithm when the input is a *complete binary tree* of size $n = 2^d - 1$, where d is the depth of the tree. (Note that such a complete binary tree is always perfectly balanced, with left and right sub-trees of the same size.), and solve it to give a time analysis for the algorithm in the complete binary tree case. Then give a *worst-case* analysis for the time, not making any assumptions about the input tree.

Triangle– 20 points A *triangle* in an undirected graph G is a triple of nodes u, v, w so that any two of them are adjacent in the graph. Use Strassen’s $O(n^{\log_2 7})$ time matrix multiply algorithm to determine whether an undirected graph, in adjacency matrix format, has a triangle, in the same order of time. (10 points efficient algorithm, 10 points correctness argument)

Implementation: 20 pts Often, divide-and-conquer algorithms only become superior to asymptotically slower algorithms for large inputs, and are slower for smaller. A simple technique for improving their performance on small inputs is to use a threshold. Put in a larger base case in the recurrence, using a simpler but asymptotically slower algorithm when we fall below this threshold. In other words, for some threshold T , we would use the recurrence if $n > T$ and use a simpler algorithm if $n \leq T$. Implement the $O(n^{\log_2 3})$ time divide-and-conquer multiplication algorithm from class, and the grade-school multiplication algorithm. Then consider

a hybrid algorithm using the technique above, where you replace the base-case of the recursion with the grade-school method for inputs of size less than some threshold T . For the different thresholds T , plot the average times to multiply random n bit numbers using the two methods (on log-log scale). Experimentally determine the best value of T . Does this method only improve the running time on small inputs, or on all inputs? Explain.