

CSE3358 Problem Set 3

01/28/05

Due 02/04/05

Problem 1: Multiplying large numbers

We often assume that multiplying two integers is a $\Theta(1)$ -time operation. This assumption becomes unrealistic if the numbers grow very large. For instance, to multiply two n -bit integers, u and v , the traditional algorithm (do it by hand to see) requires $\Theta(n^2)$ bit multiplications and additions. In this problem we will investigate a divide-and-conquer algorithm for multiplying two numbers. This will be similar to Strassen's algorithm, but applied to n -bit numbers instead of matrices of size $n \times n$.

We will divide each number into two parts, the high order bits and the low order bits. Thus u can be written as $a2^{n/2} + b$ where a represents the high order bits of u and b represents the low order bits of u . Similarly, v can be written as $c2^{n/2} + d$. Therefore,

$$uv = (a2^{n/2} + b)(c2^{n/2} + d) = ac2^n + (ad + bc)2^{n/2} + bd$$

Multiplying by a power of 2 is a simple shifting operation that can be done in linear time $\Theta(n)$. So the dominating operations are the 4 multiplications ac , ad , bc , and bd .

- (a) (10 points) Write pseudocode for a divide-and-conquer algorithm that recursively computes the products ac , ad , bc , and bd , and combines them to compute the product uv . Write a recurrence describing the algorithm running time and solve it.
- (b) (10 points) Design a different divide-and-conquer algorithm: compute ac and bd , then compute $ad + bc$ using only one multiplication along with addition and subtraction, thus reducing the number of sub-problems in each level of the recursion to 3 instead of 4. Write the recurrence describing the algorithm running time and solve it.

Problem 2: Solving Recurrences

Use the Master Theorem to solve the following recurrences. State which case of the Master Theorem you are using (and justify) for each of the recurrences.

- a. (2 points) $T(n) = 2T(\frac{n}{2}) + n^3$
- b. (2 points) $T(n) = T(\frac{9n}{10}) + n$
- c. (2 points) $T(n) = 16T(\frac{n}{4}) + n^2$
- d. (2 points) $T(n) = 7T(\frac{n}{3}) + n^2$
- e. (2 points) $T(n) = 7T(\frac{n}{2}) + n^2$
- f. (2 points) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$

Problem 3: Partitioning in Quicksort

Consider the following code for partitioning an array in Quicksort.

```

PARTITION(A,p,r)
x←A[p]    ▷ this is the pivot
i←p-1    ▷ i starts at the far left
j←r+1    ▷ j starts at the far right

while TRUE
  do repeat j←j-1    ▷ move j down until it finds an element ≤ x
    until A[j]≤ x
  repeat i←i+1    ▷ move i up until it finds an element ≥ x
    until A[i]≥ x
  if i<j          ▷ if they did not cross
    then swap A[i]↔A[j]
    else return j

```

The Quicksort algorithm used with that partitioning code is the following:

```

QUICKSORT(A,p,r)
if p<r
  then q←PARTITION(A,p,r)
      QUICKSORT(A,p,q)
      QUICKSORT(A,q+1,r)

```

When PARTITION terminates, every element in $A[1..j]$ is less than or equal to every element in $A[j+1..r]$, for $p \leq j < r$. Moreover, it always places the pivot value (originally in $A[p]$) into one of these two partitions. Since $p \leq j < r$, this split is always nontrivial i.e. both partitions are non-empty.

(a) (3 points) Show the result of above PARTITION on the array $A = [13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21]$ step by step.

There are technicalities that make the pseudocode of PARTITION a little tricky. For example, the choice of $A[p]$ as pivot is not arbitrary.

(a) (5 points) Show that if $A[r]$ is used as a pivot element in the PARTITION code, the QUICKSORT algorithm above might never terminate.

(b) (5 points) The PARTITION code here is faster in practice than the one we saw in class. Construct an array of size n for any n such that the PARTITION code here performs zero swaps while the one we saw in class performs n swaps.

(c) (5 points) The partitioning code we saw in class does not yield a balanced partitioning when the elements are not distinct. Construct an array of size n for any n such that the PARTITION code here splits the array in two halves while the the one we saw in class produces a worst-case partition.