1 Integer multiplication

In this and the next problem, we use the number of bit operations (e.g. bit addition) in the time analysis.

In Lecture 1, we discussed the problem of integer multiplication: given two binary integers $x$ and $y$ (each with $n$ bits), compute $z = x \cdot y$. We had an $O(n^2)$ time algorithm back then. We now apply divide and conquer to this problem. As the starting point, we divide $x$ (and $y$) into half: the high order $n/2$ bits and the low order $n/2$ bits. That is, we write $x = x_1 \cdot 2^{n/2} + x_0$ and $y = y_1 \cdot 2^{n/2} + y_0$.

Now give a divide and conquer algorithm based on the above division, which will call four sub-problems recursively. What is the running time?

2 Integer multiplication, continued

The previous divide and conquer approach does not seem to work too well. We want to improve the algorithm to make it run faster. The idea is similar to the matrix multiplication: we want to reduce the number of sub-problems to call recursively. Give an improved divide and conquer algorithm by improving upon the previous algorithm. Analyze the running time of your algorithm. Hint: this observation should help: $(x_1 + x_0) \cdot (y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0$.

3 Divide and conquer

An array $A[1,n]$ is said to have a majority element if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form like: is $A[i] > A[j]$? This occurs often: for example, there is no natural way of saying which of two images are larger, but we can tell whether they are the same bitmap image. On the positive side, you can get answer for questions of the form: is $A[i] = A[j]$? in constant time. Let me repeat: you can only examine whether two array elements are the same or not, but you will not be able to know which one is smaller and larger.

Now, show how to solve this problem in $O(n \log n)$ time using a divide and conquer approach. You may use a natural dividing way: divide $A$ into two arrays of half size. Argue why your algorithm is correct, and analyze its running time. Again, you can only rely on queries like $A[i] = A[j]$? but not $A[i] \leq A[j]$? (which means you can not sort the list by say merge sort).

Extra Credits. Now give a linear time algorithm for this problem. Hint: create $n/2$ pairs by pairing up $A[2i-1]$ and $A[2i]$. Discard both element in a pair if the two elements are different, and keep only one if the two are the same. Now, why does the reduced array help? You need to (i) describe the full algorithm, (ii) prove the algorithm is correct and (iii) analyze the running time. For simplicity, we assume $n$ is a power of 2.