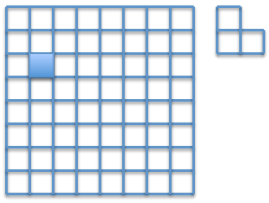
Algorithms Tutorial 2
Solutions

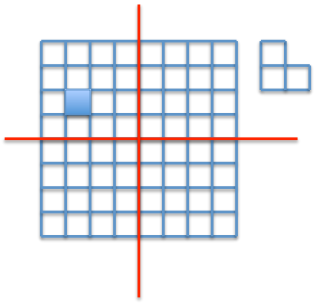
Divide and Conquer and polynomial multiplication

1. You are given a $2^n \times 2^n$ board with one of its cells missing (i.e., the board has a hole); the position of the missing cell can be arbitrary. You are also given a supply of "dominoes" each containing 3 such squares; see the figure:
   Your task is to design an algorithm which covers the entire board with such

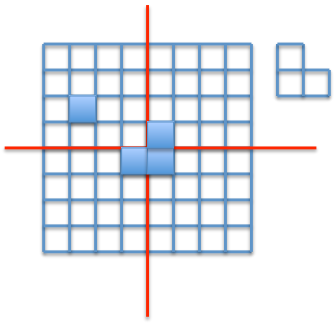


   "dominoes" except for the hole. Each dominoe can be rotated.

   Solution: We proceed by a divide and conquer recursion; thus, we split the board into 4 equal parts:



We can now apply our recursive procedure to the top left board which has a missing square. To be able to apply recursion to the remaining 3 squares we place a domino at the centre as shown below.



We treat the covered squares in each of the three pieces as a missing square and can proceed by recursion applied on all 4 squares whose sides are half the size of the size of the original square.

2. You and a friend find yourselves on a TV game show! The game works as follows. There is a **hidden** $N \times N$ table A. Each cell $A[i, j]$ of the table contains a single integer and no two cells contain the same value. At any point in time, you may ask the value of a single cell to be revealed. To win the big prize, you need to find the N cells each containing the **maximum** value in its row. Formally, you need to produce an array $M[1..N]$ so that $A[r, M[r]]$ contains the maximum value in row r of A, i.e., such that $A[r, M[r]]$ is the largest integer among $A[r, 1], A[r, 2], \ldots, A[r, N]$. In addition, to win, you should ask at most $2N\lceil \log N \rceil$ many questions. For example, if the hidden grid looks like this:

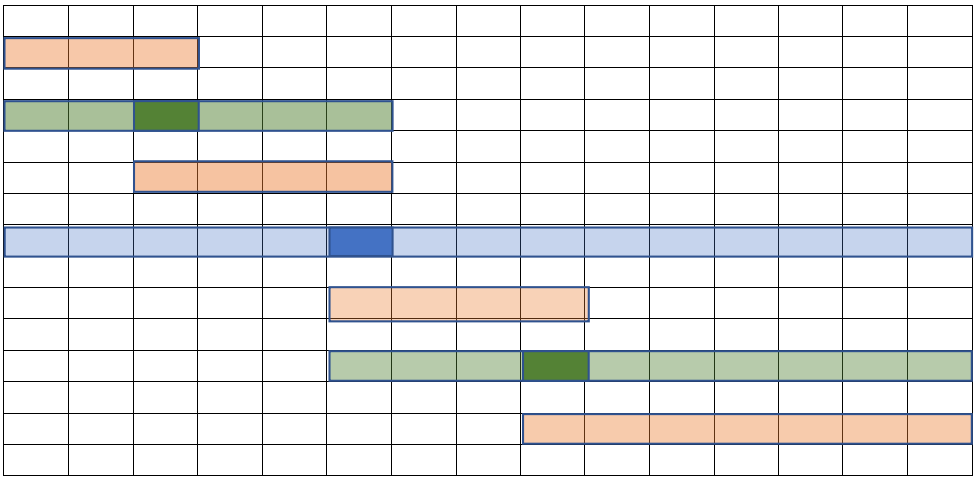
	Column 1	Column 2	Column 3	Column 4
Row 1	**10**	5	8	3
Row 2	1	**9**	7	6
Row 3	-3	**4**	-1	0
Row 4	-10	-9	-8	**2**

then the correct output would be $M = [1, 2, 2, 4]$.
Your friend has just had a go, and sadly failed to win the prize because they asked N^2 many questions which is too many. However, they whisper to you a hint: they found out that M is **non-decreasing**. Formally, they tell you that $M[1] \leq M[2] \leq \cdots \leq M[N]$ (this is the case in the example above).

Design an algorithm which asks at most $\mathbf{O(N \log N)}$ many questions that produces the array M correctly, even in the very worst case.

Hint: Note that you do not have enough time to find out the value of every cell in the grid! Try determining $M[N/2]$ first, and see if divide-and-conquer is of any assistance.

Solution: We first find $M[N/2]$. We can do this in N questions by simply examining each element in row $N/2$, and finding the largest one. Suppose $M[N/2] = x$. Then, we know for all $i < N/2$, $M[i] \leq x$ and for all $j > N/2$, $M[j] \geq x$. Thus, we can recurse to solve the same problem on the sub-grids $A[1..(N/2)-1][1..x]$ and $A[(N/2)+1..N][x..N]$. It remains to show that this approach uses at most $2N\lceil \log N \rceil$ questions.



Note that at each stage of recursion in every column at most two cells are investigated; in the first call of recursion only one cell has been investigated in each column (blue cells, max found in the dark blue cell); in the second call of recursion two cells were investigated in only one column (green cells), in the third call of recursion two cells were investigated only in three columns etc. So in each recursion call at most $2N$ cells were investigate and there are at most $\log_2(N)$ many recursion calls thus resulting in inspection of at most $2N \log_2 N$ many cells as required. Additional exercise: using the above reasoning try to find a bound for the total number of cells investigated which is sharper than $2N \log_2 N$.

3. Given positive integers M and n compute M^n using only $O(\log n)$ many multiplications. (15 pts)

Solution: Note that when n is even, $M^n = (M^{\frac{n}{2}})^2$, and when n is odd, $M^n = (M^{\frac{n-1}{2}})^2 \times M$. Hence, we can proceed by divide and conquer. If n is even, we recursively compute $M^{\frac{n}{2}}$ and then square it. If n is odd, we recursively compute $M^{\frac{n-1}{2}}$, square it and then multiply by another M. Since n is (approximately) halved in each recursive call, there are at most $O(\log n)$ recursive calls, and since we perform only one or two multiplications in each recursive call, the algorithm performs $O(\log n)$ many multiplications, as required.

Alternative Solution: Any positive integer n is the sum of a subset of the powers of 2 ($\{1, 2, 4, 8, 16, ...\}$). Thus, M^n is the product of a subset of powers of M where the power is a power of 2 ($\{M, M^2, M^4, M^8, ...\}$). We can obtain these powers of M in $O(\log n)$ time by repeated squaring and then multiply together the appropriate powers to get M^n. The appropriate powers to multiply are the powers M^{2^i} such that the i^{th} least significant bit of the binary representation of n is 1. For example, to obtain M^{11}, the binary representation of 11 is 1011, and hence we should multiply together M, M^2, and M^8.

4. Let us define the Fibonacci numbers as $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Thus, the Fibonacci sequence looks as follows: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

   (a) Show, by induction or otherwise, that

   $$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

   for all integers $n \geq 1$.

   (b) Hence or otherwise, give an algorithm that finds F_n in $O(\log n)$ time.

   Hint: You may wish to refer to Example 1.5 on page 28 of the "Review Material" booklet.

   Solution

   (a) When $n = 1$ we have

   $$\begin{aligned} \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1 \end{aligned}$$

so our claim is true for $n = 1$.

Let $k \geq 1$ be an integer, and suppose our claim holds for $n = k$ (Inductive Hypothesis). Then

$$\begin{aligned}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{k+1} &= \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{k}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}\\ &= \begin{pmatrix}F_{k+1} & F_k\\ F_k & F_{k-1}\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}\end{aligned}$$

by the Inductive Hypothesis. Hence

$$\begin{aligned}\begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^{k+1} &= \begin{pmatrix}F_{k+1}+F_k & F_{k+1}\\ F_k+F_{k-1} & F_k\end{pmatrix}\\ &= \begin{pmatrix}F_{k+2} & F_{k+1}\\ F_{k+1} & F_k\end{pmatrix}\end{aligned}$$

by the definition of the Fibonacci numbers. Hence, our claim is also true for $n = k + 1$, so by induction it is true for all integers $n \geq 1$.

(b) Let

$$G = \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}$$

and suppose we know G^x. Then, we can compute G^{2x} by simply squaring G^x, which takes $O(1)$ time. Since it is enough to compute G^n, we can do so by first computing G^{2^t} for all $t \leq \lfloor \log_2 n \rfloor$: we can do this in $O(\log n)$ steps by repeatedly squaring G.

Then, we can consider the base 2 representation of n: this tells us precisely which G^{2^t} matrices we should multiply together to obtain G^n.

As an alternative (and equivalent) solution, we can proceed by divide and conquer. To compute G^n: if n is even, recursively compute $G^{n/2}$ and square it in $O(1)$. If n is odd, recursively compute $G^{(n-1)/2}$, square it and then multiply by another G: this last step also occurs in $O(1)$. Since there are $O(\log n)$ steps of the recursion only, this algorithm runs in $O(\log n)$.

5. Design an algorithm which multiplies a polynomial of degree 16 with a polynomial of degree 8 using only 25 multiplications in which both operands (which both depend on the coefficients of the polynomial) can be arbitrarily large.

Solution: The product of a polynomial $A(x)$ of degree 16 and a polynomial $B(x)$ of degree 8 is a polynomial $C(x)$ of degree 24; thus, to uniquely determine

$C(x)$ we need its values at 25 inputs, so we choose

$$x = -12, -11, \ldots, -1, 0, 1, \ldots, 12.$$

We now evaluate $A(i)$ and $B(i)$ for all i such that $-12 \leq i \leq 12$. Note that this does not involve any multiplication of two arbitrarily large numbers but only multiplications of one arbitrarily large number (a coefficient of $A(x)$ or $B(x)$) with a constant. Notice that the constant involved can be very large, such as 12^{16} occurring when we evaluate $A(12) = A_0 + A_1 \times 12 + \ldots + A_{16} \times 12^{16}$. We now perform 25 large number multiplications which produce the values of the product polynomial $C(x) = A(x)B(x)$ at inputs ranging from -12 to 12. Having found $C(-12) = A(-12)B(-12), \ldots, C(12) = A(12)B(12)$, we now solve the following system of 25 linear equations in variables $C_0, \ldots, C_{24}$:

$$\{C_0 + C_1 i + C_2 i^2 + \ldots + C_{24} i^{24} = C(i) = A(i)B(i) \ : \ -12 \leq i \leq 12\}$$

Solving such a system expresses the solutions for $C_0, \ldots, C_{24}$ as a linear combination of values of $C(i)$ and thus involves only constants times large values multiplications. So in total we have used only 25 multiplications where both numbers can be arbitrarily large, because they depend on the values of the coefficients of polynomials $A(x)$ and $B(x)$.

6. Multiply the following pairs of polynomials using at most the prescribed number of multiplications of large numbers (large numbers are those which depend on the coefficients and thus can be arbitrarily large).

   (a) $P(x) = a_0 + a_2 x^2 + a_4 x^4 + a_6 x^6$; $Q(x) = b_0 + b_2 x^2 + b_4 x^4 + b_6 x^6$ using at most 7 multiplications of large numbers;

   (b) $P(x) = a_0 + a_{100} x^{100}$ and $Q(x) = b_0 + b_{100} x^{100}$ with at most 3 multiplications of large numbers.

   Solution:

   (a) Note that using the substitution $y = x^2$ reduces $P(x)$ to $P^*(y) = a_0 + a_2 y + a_4 y^2 + a_6 y^3$ and $Q(x)$ to $Q^*(y) = b_0 + b_2 y + b_4 y^2 + b_6 y^3$. The product $R^*(y) = P^*(y)Q^*(y)$ of these two polynomials is of degree 6 so to uniquely determine $R^*(y)$ we need 7 of its values. Thus, we evaluate $P^*(y)$ and $Q^*(y)$ at seven values of its argument x, by letting $x = -3, -2, -1, 0, 1, 2, 3$. We then obtain from these 7 values of $R^*(y)$ its

coefficients, by solving the corresponding system of linear equation in coefficients $r_0, \ldots, r_6$ such that $R^*(x) = r_0 + r_1 x + \ldots + r_6 x^6$. Thus we solve the system $\{\sum_{j=0}^{6} r_j i^j = R^*(i) \ : \ -3 \leq i \leq 3\}$. We now form the polynomial $R^*(x) = r_0 + r_1 x + \ldots + r_6 x^6$ with thus obtained r_j and finally substitute back y with x^2 obtaining $R(x) = P(x)Q(x)$.

(b) We use (essentially) the Karatsuba trick:

$$\begin{aligned}(a_0 + a_{100}x^{100})(b_0 + b_{100}x^{100}) &= a_0 b_0 + (a_0 b_{100} + b_0 a_{100})x^{100} + a_{100} b_{100} x^{200} \\ &= a_0 b_0 + ((a_0 + a_{100})(b_0 + b_{100}) - a_0 b_0 - a_{100} b_{100})x^{100} + a_{100} b_{100} x^{200}\end{aligned}$$

Note that the last expression involves only three multiplications: $a_0 b_0$, $a_{100} b_{100}$ and $(a_0 + a_{100})(b_0 + b_{100})$.

7.

(a) Multiply two complex numbers $(a + i\,b)$ and $(c + i\,d)$ (where a, b, c, d are all real numbers) using only 3 real number multiplications.

(a) Find $(a + i\,b)^2$ using only two multiplications of real numbers.

(b) Find the product $(a + i\,b)^2(c + i\,d)^2$ using only five real number multiplications.

Solution:

(a) This is again the Karatsuba trick:

$$(a + i\,b)(c + i\,d) = ac - bd + (bc + ad)\,i = ac - bd + ((a+b)(c+d) - ac - bd)\,i$$

so we need only 3 multiplications: ac and bd and $(a + b)(c + d)$.

(a) $(a + i\,b)^2 = a^2 - b^2 + 2ab\,i = (a + b)(a - b) + (a + a)b\,i$.

(b) Just note that $(a + i\,b)^2(c + i\,d)^2 = ((a + i\,b)(c + i\,d))^2$; you can now use (a) to multiply $(a + i\,b)(c + i\,d)$ with only three multiplications and then you can use (b) to square the result with two additional multiplications.

8. Assume that you are given a polynomial $P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ and a polynomial $Q(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$ whose coefficients can be arbitrarily large numbers. Let $R(x) = P(x)^2 - Q(x)^2$. Compute the coefficients of $R(x)$ using only 7 large number multiplications.

Solution: Just note that $R(x) = P(x)^2 - Q(x)^2 = (P(x) - Q(x))(P(x) + Q(x))$. Now just compute $P(x) - Q(x)$ and $P(x) + Q(x)$ which are both polynomials of degree 3, so their product can be obtained using only 7 multiplications as explained in question 5 in this tutorial.

9. You are given a polynomial $P(x) = A_0 + A_1 x^{100} + A_2 x^{200}$ where A_0, A_1, A_2 can be arbitrarily large integers. Design an algorithm which squares $P(x)$ using only 5 large integer multiplications. (15 pts)

Solution: Note that using the substitution $y = x^{100}$ reduces $P(x)$ to $P^*(y) = A_0 + A_1 y^1 + A_2 y^2$, and it is clear that $P^*(y)^2$ will have the same coefficients as $P(x)^2$. The polynomial $Q^*(y) = P^*(y)^2$ is of degree 4 so to uniquely determine $Q^*(y)$ we need five of its values. Thus, we evaluate $Q^*(y)$ at five values: -2, -1, 0, 1, and 2 (this is where the 5 large integer multiplications occur). Then, using these five values, we obtain the coefficients of $Q^*(y)$ by solving the corresponding system of linear equations. After we have found the coefficients of $Q^*(y)$, we can substitute y back with x^{100} to obtain $P(x)^2$. **Alternative, cleverer solution by a student:** Note that

$$(A_0 + A_1 x^{100} + A_2 x^{200})^2 = A_0^2 + 2A_0A_1 x^{100} + (A_1^2 + 2A_0A_2)x^{200} + 2A_1A_2 x^{300} + A_2^2 x^{400}$$

and that

$$(A_0 + A_1 + A_2)^2 - A_0^2 - A_2^2 - 2A_0A_1 - 2A_1A_2 = A_1^2 + 2A_0A_2$$

Thus, we only need 5 multiplications of large numbers to compute A_0^2, A_2^2, A_0A_1, A_1A_2 and $(A_0 + A_1 + A_2)^2$ to obtain all the needed coefficients.