**Assignment 1, COMP251, Winter 2019. Jan 10, 2019.**

**Exercise 1.** ALGORITHM DESIGN, ORACLES. We have an oracle that takes two inputs, $x$ and $y$, and reports whether $x = y$. Complexity in this exercise is measured by the number of uses of this oracle. A set $A$ of $n$ (not necessarily different) elements is given. The number of occurrences of an element is called its frequency. We also know that there is an element with frequency $> n/2$. Show how you can find that element by using the oracle not more than $O(n)$ times.

SOLUTION. With one majority element, we can find all others in time $n - 1$. So, let's find one element by divide-and-conquer in a recursive manner. If $n \leq 2$, we can output the first element. If $n > 2$ is odd, then take the first element $\alpha$ and compare it with the $n - 1$ others to determine whether it is a majority element (by a majority vote by the $n - 1$ others; a tie is broken in favor of "a majority element"). If $\alpha$ is not a majority element, then throw it out, and proceed with the $n - 1$ other elements, where now $n - 1$ is even.

If $n > 2$ is even, pair all elements, and throw away all pairs with distinct elements (so we are throwing out at least as many minority as majority elements). In addition, throw out the second element in each similar pair. This procedure preserves a strict majority among the leftover elements, and there is at least one surviving element (by the pigeon hole principle) and at most $n/2$. Proceed recursively with the remaining elements.

Arguing as in the chip testing problem, and writing $T_n$ for the time taken on a problem of size $n$, we have $T_0 = T_1 = T_2 = 0$, and

$$T_n \leq T_{\lfloor n/2 \rfloor} + \lfloor n/2 \rfloor, n > 2.$$

The solution is $T_n = O(n)$.

**Exercise 2.** DESIGN OF A DIVIDE-AND-CONQUER ALGORITHM. Consider an $n \times n$ chessboard, where $n$ is a power of two. Define a trimino as a 3-piece tile (i.e., a tile that would cover positions $(1, 1)$, $(1, 2)$ and $(2, 1)$ on the chess board). Triminos can be rotated. Develop a divide-and-conquer algorithm for covering the chessboard with triminos such that exactly one of the $n^2$ spaces on the chessboard is not covered.

SOLUTION. Write $A_n$ for a perfectly tiled $n \times n$ chessboard with an empty slot in the top right corner. Let $R(A_n, \theta)$ be $A_n$ rotated by $\theta$ degrees in clockwise fashion. Let $C(A_n, A'_n, A''_n, A'''_n)$ be the composition of four $n \times n$ perfectly tiled chessboards, possibly rotated, with $A_n$ top right, and the three other arguments of $C$ placed in clockwise order. Let TRIMINO$(n)$ output a perfectly tiled $n \times n$ chessboard. The recursive procedure TRIMINO can be described as follows:

TRIMINO $(n)$

```
if n = 2, then output A_2 by placing one trimino
          such that the top right corner is empty
     else A_{n/2} ← TRIMINO(n/2)
          B_n = C(A_{n/2}, R(A_{n/2}, 270), A_{n/2}, R(A_{n/2}, 90))
          place a trimino in the three empty slots at the center
          (note that B_n is perfectly tiled)
          output B_n
```

**Exercise 3.** THE FIBONACCI EXAMPLE GENERALIZED. How would you compute $x_n$ in a uniform cost model, where we have $x_0 = 2$, $x_1 = 0$, $x_2 = 7$, and for $n > 3$, $x_n = 7x_{n-1} - 11x_{n-2} + 5x_{n-3}$?

SOLUTION. The standard solution uses the fact that

$$\begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \end{bmatrix} = M \begin{bmatrix} x_{n-1} \\ x_{n-2} \\ x_{n-3} \end{bmatrix}, \text{ where } M = \begin{bmatrix} 7 & -11 & 5 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Thus,

$$\begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \end{bmatrix} = M^{n-2} \begin{bmatrix} x_2 \\ x_1 \\ x_0 \end{bmatrix}.$$

We can compute $M^{n-2}$ in $O(\log n)$ time in the uniform cost model using fast exponentiation.

**Exercise 4.** BIT COMPLEXITY. How would you compute $x_n$ efficiently, where we have $x_0 = 7$, $x_1 = 13$, $x_n = x_{n-1}^3 x_{n-2}^5$, in a bit model of complexity? Only integer arithmetic is permitted. What is your complexity? If it is $O(n^c)$, give $c$, and if it is $O(c^n)$, give $c$.

SOLUTION. This is the hardest question—by far—in assignment 1. Note that the eventual solution is of the form

$$x_n = 7^{y_n} 13^{z_n}.$$

Let us first compute $y_n$ and $z_n$. The recursion gives us

$$y_n = 3y_{n-1} + 5y_{n-2}, y_0 = 1, y_1 = 0, z_n = 3z_{n-1} + 5z_{n-2}, z_0 = 0, z_1 = 1.$$

These are two generalized Fibonacci sequences. Let

$$M = \begin{bmatrix} 3 & 5 \\ 1 & 0 \end{bmatrix}.$$

Then

$$\begin{bmatrix} y_n \\ y_{n-1} \end{bmatrix} = M^{n-1} \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

and

$$\begin{bmatrix} z_n \\ z_{n-1} \end{bmatrix} = M^{n-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

We compute $M^{n-1}$ by fast exponentiation, and note that $y_n$ and $z_n$ are just two elements of $M^{n-1}$. Given $y_n$ and $z_n$, we compute $7^{y_n}$ and $13^{z_n}$ separately by fast exponentiation. Finally, we compute the product of these two powers by Karatsuba's method.

FAST EXPONENTIATION IN THE BIT MODEL. When we compute $x^n$ for a fixed constant $x$ by fast exponentiation, we are working with $O(n)$-bit numbers. Thus, Karatsuba's method gives a recurrence for the bit complexity $T_n$ like

$$T_n = T_{n/2} + n^{\log_2 3},$$

which yields $T_n = O(n^{\log_2 3})$ by the master theorem.

THE FINAL TALLY. The number of bits in $x_n$ is $O(\log x_n) = O(y_n + z_n)$. Noting that $7^{y_n}$ has $O(y_n)$ bits, and $x_n$ has $O(y_n + z_n)$ bits, our total complexity has four components:

$$\begin{aligned} O\left(n^{\log_2 3}\right) &\text{ (computation of } M^{n-1}) \\ O\left(y_n^{\log_2 3}\right) &\text{ (computation of } 7^{y_n}) \\ O\left(z_n^{\log_2 3}\right) &\text{ (computation of } 13^{z_n}) \\ O\left((y_n + z_n)^{\log_2 3}\right) &\text{ (computation of } x_n = 7^{y_n} 13^{z_n}) \end{aligned}$$

It should be clear that the last contribution dominates. We are now forced, somehow, to compute the order of $y_n$ and $z_n$. An equation like

$$y_n = 3y_{n-1} + 5y_{n-2}, y_0 = 1, y_1 = 0,$$

has a general solution of the form

$$y_n = c\alpha^n + c'\beta^n,$$

where $c, c' > 0$ are constants and $\alpha, \beta$ are solutions of the quadratic equation

$$x^2 = 3x + 5.$$

Thus, $\alpha = \frac{3+\sqrt{29}}{2}$ and $\beta = \frac{3-\sqrt{29}}{2}$. The largest solution dominates, and since we do not have to compute the constants $c$ and $c'$ (as we are only after a big oh result), we have

$$y_n + z_n = O\left(\left(\frac{3+\sqrt{29}}{2}\right)^n\right).$$

The bit complexity of this method, therefore, is $O(\gamma^n)$, where

$$\gamma = \left(\frac{3+\sqrt{29}}{2}\right)^{\log_2 3}.$$

**Exercise 5.** COMPUTING HAMMING DISTANCES. The Hamming distance between two binary vectors is the number of components that differ. Let $x_1, \ldots, x_n$ be vectors of $\{0,1\}^n$. We would like to compute the Hamming distances between $x_i$ and $x_j$ for all $i \neq j$. Using a RAM model of computation, how fast can you do this (in $O(.)$ notation)? The idea is to beat the obvious $\Theta(n^3)$ algorithm by a good margin, i.e., by achieving $O(n^\alpha)$ complexity for some $\alpha < 2.9$. Partial credit for only getting $o(n^3)$. Hint: You may want to recast this problem in terms of one or more other problems.

SOLUTION. First replace all zeroes in the data by $-1$'s. Note that the inner product (or "dot product") $x_i \cdot x_j$ equals the number of identical components minus the Hamming distance, which we denote by $H(i,j)$, so

$$x_i \cdot x_j = \sum_{k=1}^{n} x_i(k)x_j(k) = n - 2H(i,j).$$

Let $A$ be the matrix with $i$-th row $x_i$, and let $A^t$ be its transpose (i.e., its $j$-th column is $x_j$). Then the element at position $(i,j)$ in the matrix product $AA^t$ is $x_i \cdot x_j = n - 2H(i,j)$. We can compute $AA^t$ by Strassen's method in worst-case time $O(n^{\log_2 7})$, and are done, since $\log_2 7 = 2.8073549220576041074\ldots$.