COMP3121/9101/3821/9801 Assignment 2
Due date: Friday, 7 of April at Noon

1. Fibonacci numbers are defined by $F(0) = 0$, $F(1) = 1$ and $F(n) = F(n-1)+F(n-2)$ for all $n \geq 2$. Thus, the Fibonacci sequence looks as follows: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$.
   (a) Show that $\begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$.
   (b) Find $F(n)$ in $O(\log n)$ many steps.

2. (a) Multiply two complex numbers $a + i\,b$ and $c + i\,d$ using only three real numbers multiplications and any number of real number additions.
   (b) Find the square of a complex number $a + i\,b$ using only two real numbers multiplications and any number of real number additions.
   (c) Evaluate $(a + i\,b)^2(c + i\,d)^2$ using only 5 large number multiplications.

3. Multiply polynomials $P(x) = a_0 + a_{17}x^{17} + a_{19}x^{19} + a_{21}x^{21} + a_{23}x^{23}$ and $Q(x) = b_0 + b_{17}x^{17} + b_{19}x^{19} + b_{21}x^{21} + b_{23}x^{23}$ using only 16 multiplications of large numbers (i.e., multiplication of two numbers which both depend on the coefficients which can be arbitrarily large)

4. Simplify the polynomial
$$P(x) = (x-1)(x-\omega_{15})(x-\omega_{15}^2)(x-\omega_{15}^3)\ldots(x-\omega_{15}^{14})$$
where $\omega_{15} = e^{i\cdot 2\pi/15}$ is a primitive root of unity of order 15.

   Hint: two polynomials of the same degree which have the same coefficient in front of the largest power and have the same zeros are equal.

5. To apply the FFT to the sequence $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ we apply recursively FFT and obtain $FFT(a_0, a_2, a_4, a_6)$ and $FFT(a_1, a_3, a_5, a_7)$. Proceeding further with recursion, we obtain $FFT(a_0, a_4)$ and $FFT(a_2, a_6)$ as well as $FFT(a_1, a_5)$ and $FFT(a_3, a_7)$. Thus, from bottom up, $FFT(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ is obtained using permutation $(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$ as the leaves of the recursion tree of the original input sequence. Given any input $(a_0, a_1, a_2, \ldots, a_{2^n-1})$ describe the permutation of the leaves of the recursion tree.

   Hint: write indices in binary and see what the relationship is of the bits of the i^{th} element of the original sequence and the i^{th} element of the resulting permutation of elements as they appear on the leaves on the recursion tree.

6. Assume you are given a sequence $(q_0, q_1, \ldots, q_n)$. Compute all elements of the sequence

$$\left(\sum_{i+j=m} (j+1)q_j q_i, \quad 0 \leq m \leq 2n\right)$$

in time $O(n \log n)$.

Hint: how do we call sequences of this form?

Additional problems for extended classes only (COMP3821/9801)

7. Assume that you are given an $n \times n$ table; each square of the table contains a distinct number. A square is a local minimum if the number it contains is smaller than the numbers contained in all neighbouring squares which share an edge with that square. Thus, each of the four corner squares has only two neighbours sharing an edge; $4(n-2)$ non corner squares along the edges of the table have three neighbouring squares and all internal squares have four neighbouring squares. A square is a local minimum if the number it contains is smaller than the numbers contained in all of its neighbouring squares. You can only make queries which given numbers m and k, $1 \leq m, k \leq n$, return the value in the square (m, k).

   (a) Show by an example that if you search for a local minimum by picking a square and then moving to a neighbouring square with a smaller value (if there is such) might take $\Theta(n^2)$ many queries.

   (b) Design an algorithm which finds a local minimum and makes only $O(n)$ many queries.

   Hint: a surface $z = f(x, y)$ where (x, y) belongs to a bounded closed subset S of the plane attains its minimal height z either along the curve which is the edge of S or strictly in the interior of S. Can you use this fact to design a divide-and-conquer algorithm?

8. Assume you are given two data bases A and B; A contains n numbers, B contains $n+1$ numbers, all numbers are distinct. You can query each database only in the following way: you specify a number k and the database (A or B); the query returns the value of the k^{th} smallest element in that database. Design an algorithm which produces the median of the set of all elements which belong to either A or B using at most $O(\log n)$ queries.

   Hint: Let m be the median of all elements which belong to either A or B; then if k many elements of A are smaller than m, then exactly $n-k$ elements of B must be smaller than m. How can you find such m efficiently?