Algorithms Tutorial 2
Solutions

Divide and Conquer, polynomial multiplication and the FFT

1. You are given an array A containing a very long sequence of N integers which form an arithmetic progression, except that one number has been repeated twice (consecutively). Thus, the sequence looks like this:

   $$a, a+b, a+2b, \ldots, a+(k-1)b, a+kb, a+kb, a+(k+1)b, \ldots, a+Mb$$

   You are also given the length N of the array.

   (a) If only one number has been repeated in such a sequence, what is the value of M? Keep in mind that the starting element is $a = a + 0 \times b$.

   (b) What would have been the value of M in a sequence corresponding to such an arithmetic progression of length N if no element were repeated twice?

   (c) Find the repeated value by accessing only $O(\log_2 N)$ many entries of A. (Hint: you might find (a) and (b) useful.)

   Solution:

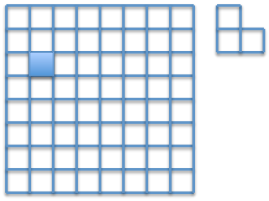
   (a) If a number has been repeated, then $M = N - 2$; note that we start with $a = a + 0 \times b$.

   (b) If no number is repeated and all numbers are distinct then $M = N - 1$.

   (c) This is a typical divide and conquer. We first find the value of $A[1]$ which gives the starting point a of the sequence. We then look at the value $A[2]$; if $A[1] = A[2]$ we are done, otherwise $A[2] - A[1]$ will give us the value

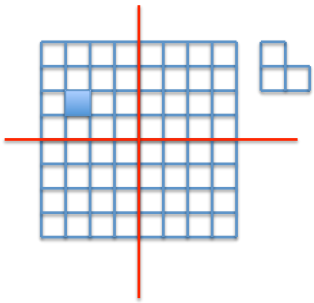
b of the difference of two consecutive terms. We then look at $A[\lfloor N/2 \rfloor]$ and compute $R = (A[\lfloor N/2 \rfloor] - A[1])/b$ and compare it with $\lfloor N/2 \rfloor$ which is the length of the subsequence $A[1 \ldots \lfloor N/2 \rfloor]$ and apply (a) and (b): if $R = \lfloor N/2 \rfloor - 1$ then the duplicate is in the subarray $A[\lfloor N/2 \rfloor \ldots N]$; if $R = \lfloor N/2 \rfloor - 2$ then the duplicate is in the subarray $A[1 \ldots \lfloor N/2 \rfloor]$. We now continue such a divide and conquer procedure with the sub-array containing the duplicate. Clearly, since such a process lasts only $\log N$ many steps, we will inspect only $\log N$ many values of A.

2. 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

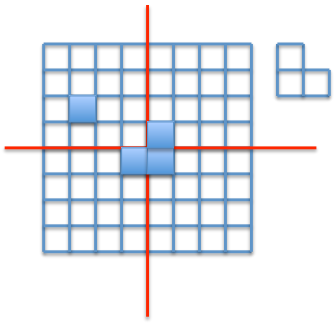


"dominoes" except for the hole. Each dominoe can be rotated.

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.

3. 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:

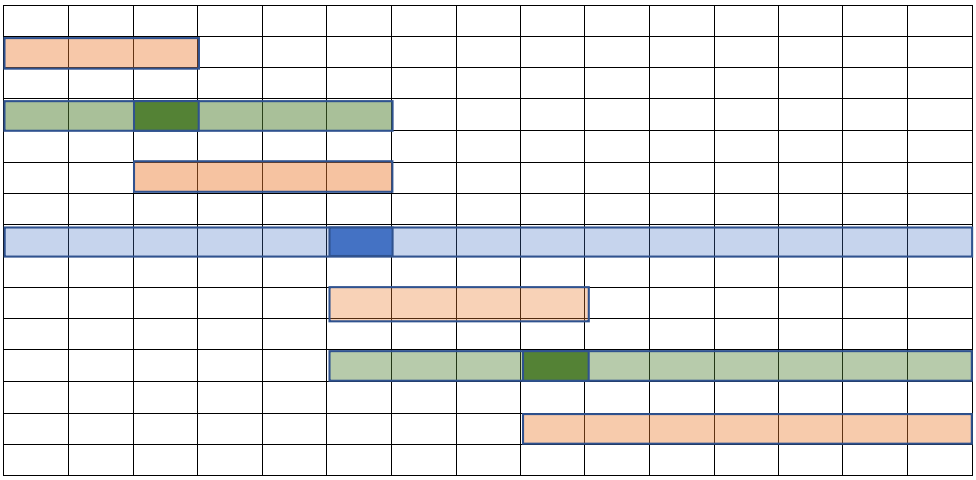
	Column 1	Column 2	Column 3	Column 4
Row 1	**10**	5	8	3
Row 2	1	**9**	7	6
Row 3	-3	**4**	-1	0
Row 4	-10	-9	-8	**2**

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 $2N \log_2 N$.

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) $P(x) = a_0 + a_2x^2 + a_4x^4 + a_6x^6$; $Q(x) = b_0 + b_2x^2 + b_4x^4 + b_6x^6$ 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 $P^*(y) = a_0 + a_2y + a_4y^2 + a_6y^3$ and $Q(x)$ to $Q^*(y) = b_0 + b_2y + b_4y^2 + b_6y^3$. 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 $R^*(x) = r_0 + r_1x + \ldots + r_6x^6$. Thus we solve the system $\{\sum_{j=0}^{6} r_j i^j = R^*(i) \ : \ -3 \le i \le 3\}$. We now form the polynomial $R^*(x) = r_0 + r_1x + \ldots + r_6x^6$ 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:

$$\begin{aligned}(a_0 + a_{100}x^{100})(b_0 + b_{100}x^{100}) &= a_0b_0 + (a_0b_{100} + b_0a_{100})x^{100} + a_{100}b_{100}x^{200} \\ &= a_0b_0 + ((a_0 + a_{100})(b_0 + b_{100}) - a_0b_0 - a_{100}b_{100})x^{100} + a_{100}b_{100}x^{200}\end{aligned}$$

   Note that the last expression involves only three multiplications: a_0b_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. Describe all k which satisfy $i\,\omega_{64}^{13}\omega_{32}^{11} = \omega_{64}^{k}$ (i is the imaginary unit).

Solution: This problem only tests your understanding of the roots of unity: note that $i = \omega_4$ and, by the cancelation lemma, that $\omega_{32}^{11} = \omega_{64}^{22}$ and $\omega_4 = \omega_{64}^{16}$. Thus

$$i\,\omega_{64}^{13}\omega_{32}^{11} = \omega_4\omega_{64}^{13}\omega_{64}^{22} = \omega_{64}^{16}\omega_{64}^{13}\omega_{64}^{22} = \omega_{64}^{16+13+22} = \omega_{64}^{51} = \omega_{64}^{51+64m}$$

where m is an arbitrary integer (positive or negative). Thus, the solutions are all k of the form $k = 64m+51$ where m is an arbitrary integer.

9. What are the real and the imaginary parts of $\mathrm{e}^{i\frac{\pi}{4}}$? Compute the DFT of the sequence $A = (0,1,2,3,4,5,6,7)$ by applying the FFT algorithm by hand.

Solution: Note that

$$\omega_8 = \mathrm{e}^{i\frac{2\pi}{8}} = \mathrm{e}^{i\frac{\pi}{4}} = \cos\frac{\pi}{4} + i\sin\frac{\pi}{4} = \frac{\sqrt{2}}{2} + i\,\frac{\sqrt{2}}{2} = \frac{\sqrt{2}}{2}(1+i).$$

The polynomial associated with the given sequence is

$$P(x) = x+2x^2+3x^3+4x^4+5x^5+6x^6+7x^7$$

We split it into even and odd powers

$$P(x) = (2x^2+4x^4+6x^6)+x(1+3x^2+5x^4+7x^6)$$

We now let

$$P^{[0]}(y) = 2y+4y^2+6y^3; \quad P^{[1]}(y) = 1+3y+5y^2+7y^3$$

Then we have

$$P(x) = P^{[0]}(x^2)+xP^{[1]}(x^2)$$

We have to evaluate $P(x)$ at all roots of unity of order 8. Thus we have for $k = 0, 1, 2, 3$

$$P(\omega_8^k) = P^{[0]}(\omega_4^k) + \omega_8^k P^{[1]}(\omega_4^k) \tag{1}$$
$$P(\omega_8^{4+k}) = P^{[0]}(\omega_4^k) - \omega_8^k P^{[1]}(\omega_4^k) \tag{2}$$

We now have to compute $P^{[0]}(\omega_4^k)$ and $P^{[1]}(\omega_4^k)$ for $k = 0, 1, 2, 3$. We could again split the two polynomial into even and odd powers, but since their degree is low we can compute these values directly, also because the roots of unity of order 4 are so simple: $\omega_4^0 = 1$, $\omega_4^1 = i$, $\omega_4^2 = -1$ and finally $\omega_4^3 = -i$. We now have for $P^{[0]}(y) = 2y + 4y^2 + 6y^3$:

$$\begin{aligned}
&P^{[0]}(\omega_4^0) = P^{[0]}(1) = 2 + 4 + 6 = 12;\\
&P^{[0]}(\omega_4^1) = P^{[0]}(i) = 2i + 4i^2 + 6i^3 = 2\,i - 4 - 6\,i = -4 - 4\,i;\\
&P^{[0]}(\omega_4^2) = P^{[0]}(-1) = -2 + 4 - 6 = -4;\\
&P^{[0]}(\omega_4^3) = P^{[0]}(-i) = -2\,i - 4 + 6\,i = -4 + 4\,i;
\end{aligned}$$

and similarly for $P^{[1]}(y) = 1 + 3y + 5y^2 + 7y^3$:

$$\begin{aligned}
&P^{[1]}(\omega_4^0) = P^{[1]}(1) = 1 + 3 + 5 + 7 = 16;\\
&P^{[1]}(\omega_4^1) = P^{[1]}(i) = 1 + 3\,i + 5i^2 + 7i^3 = 1 + 3\,i - 5 - 7i = -4 - 4\,i;\\
&P^{[1]}(\omega_4^2) = P^{[1]}(-1) = 1 - 3 + 5 - 7 = -4;\\
&P^{[1]}(\omega_4^3) = P^{[1]}(-i) = 1 - 3\,i + 5i^2 - 7i^3 = 1 - 3\,i - 5 + 7\,i = -4 + 4\,i
\end{aligned}$$

we also compute: $\omega_8^0 = 1$; $\omega_8^1 = \frac{\sqrt{2}}{2}(1 + i)$; $\omega_8^2 = i$; $\omega_8^3 = \frac{\sqrt{2}}{2}(-1 + i)$.

We can now substitute $k = 0, 1, 2, 3$ in (1) and (2):

$$\begin{aligned}
P(\omega_8^0) &= P^{[0]}(\omega_4^0) + \omega_8^0 P^{[1]}(\omega_4^0) = 12 + 16 = 28 \\
P(\omega_8^{4+0}) &= P^{[0]}(\omega_4^0) - \omega_8^0 P^{[1]}(\omega_4^0) = 12 - 16 = -4 \\
\hline
P(\omega_8^1) &= P^{[0]}(\omega_4^1) + \omega_8^1 P^{[1]}(\omega_4^1) = -4 - 4\,i + \frac{\sqrt{2}}{2}(1+i)(-4-4\,i) = -4 - 4(1+\sqrt{2})\,i \\
P(\omega_8^{4+1}) &= P^{[0]}(\omega_4^1) - \omega_8^1 P^{[1]}(\omega_4^1) = -4 - 4\,i - \frac{\sqrt{2}}{2}(1+i)(-4-4\,i) = -4 - 4(1-\sqrt{2})\,i \\
\hline
P(\omega_8^2) &= P^{[0]}(\omega_4^2) + \omega_8^2 P^{[1]}(\omega_4^2) = -4(1+i) \\
P(\omega_8^{4+2}) &= P^{[0]}(\omega_4^2) - \omega_8^2 P^{[1]}(\omega_4^2) = -4(1-i) \\
\hline
P(\omega_8^3) &= P^{[0]}(\omega_4^3) + \omega_8^3 P^{[1]}(\omega_4^3) = -4 + 4\,i + \frac{\sqrt{2}}{2}(-1+i)(-4+4\,i) = -4 + i(4-4\sqrt{2}) \\
P(\omega_8^{4+3}) &= P^{[0]}(\omega_4^3) - \omega_8^3 P^{[1]}(\omega_4^3) = -4 + 4\,i - \frac{\sqrt{2}}{2}(-1+i)(-4+4\,i) = -4 + i(4+4\sqrt{2}) \\
\hline
\end{aligned}$$

Thus, $\widehat{A} = \langle 28,\ \ -4 - 4(1+\sqrt{2})\,i,\ \ -4(1+i),\ \ -4 + 4(1-\sqrt{2})\,i,\ \ -4,\ \ -4 - 4(1-\sqrt{2})\,i,\ \ -4(1-i),\ \ -4 + (4+4\sqrt{2})\,i \rangle$

10. Describe how you would compute all elements of the sequence $F(0), F(1), F(2), \ldots, F(2n)$ where

(a)
$$F(m) = \sum_{\substack{i+j=m \\ 0 \le i,j \le n}} i^3 j^2$$

(b)
$$F(m) = \sum_{\substack{i+j=m \\ 0 \le i,j \le n}} \log(j+1)^i$$

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

Solution: Note that the first sequence is just the convolution of sequences $A = \langle 0, 1, 2^3, 3^3, 4^3, \ldots, n^3 \rangle$ and $B = \langle 0, 1, 2^2, 3^2, 4^2, \ldots, n^2 \rangle$, while the second sequence can be written as

$$F(m) = \sum_{\substack{i+j=m \\ 0 \le i,j \le n}} \log(j+1)^i = \sum_{\substack{i+j=m \\ 0 \le i,j \le n}} i \log(j+1),$$

so it is a convolution of sequences

$$C = \langle 0, 1, 2, 3, 4, \ldots, n \rangle \quad \text{and} \quad D = \langle \log 1, \log 2, \log 3, \log 4, \log 5, \ldots, \log n \rangle.$$

Both convolutions can be efficiently evaluated by computing their DFT of their zero paddings to length $2^{\lceil \log_2(2n-1) \rceil}$, using the FFT algorithm, multiplying the corresponding values and then taking the inverse DFT of the product.

11. (a) Compute by any method you wish the (linear) convolution $s * s$ of the sequence $s = \langle 1, 2, 0, 4 \rangle$ with itself. (Note that there is no requirement on the efficiency of your method, and that the sequence is really short!)

    (b) If a sequence x has n terms and sequence y has k terms, how many terms does the convolution $x * y$ of these two sequences have?

    (c) Is it true that $s * t = t * s$ for any two sequences s and t? Explain why or why not.

    (d) Describe how we compute **efficiently** the convolution of two (long) sequences?

    Solution:

    (a) For $s = \langle 1, 2, 0, 4 \rangle$ the associated polynomial is $S(x) = 1 + 2x + 4x^3$; thus the convolution of s with itself is the sequence of the coefficients of the polynomial $S^2(x) = (1 + 2x + 4x^3)^2 = 1 + 4x + 4x^2 + 8x^3 + 16x^4 + 16x^6$, i.e., the sequence $\widehat{s} = \langle 1, 4, 4, 8, 16, 0, 16 \rangle$.

    (b) If a sequence has n terms the corresponding polynomial is of degree $n-1$; thus the product of the two polynomials is of degree $n-1+k-1 = n+k-2$ and consequently it has $n + k - 1$ many coefficients.

    (c) Since the product of the two corresponding polynomials is commutative, so is the convolution.

    (d) A convolution of a sequence of length n and a sequence of length m can be efficiently evaluated by computing their DFT of their zero paddings to length $2^{\lceil \log_2(n+m-1) \rceil}$, using the FFT algorithm, multiplying the corresponding values and then taking the inverse DFT of the product.

12. In this part we will extend the convolution algorithm described in class to multiply multiple polynomials together (not just two).
    Suppose you have K polynomials $P_1, \ldots, P_K$ each of degree at least one, and that

$$\text{degree}(P_1) + \cdots + \text{degree}(P_K) = S$$

(i) Show that you can find the product of these K polynomials in $O(KS \log S)$ time.
Hint: consider using divide-and-conquer; a tree structure might be helpful here as well. Also, remember that if x, y, z are all positive, then $\log(x + y) < \log(x + y + z)$

(ii) Show that you can find the product of these K polynomials in $O(S \log S \log K)$ time. Explain why your algorithm has the required time complexity.
Hint: consider using divide-and-conquer!

Solution

(i) We obtain the products $\Pi(i) = P_1(x) \cdot P_2(x) \ldots \cdot P_i(x)$ for all $1 \leq i \leq K$ by a simple recursion. Initially, $\Pi(1) = P_1(x)$, and for all $i < K$ we clearly have $\Pi(i + 1) = \Pi(i) \cdot P_{i+1}(x)$. At each stage, the degree of the partial product $\Pi(i)$ and of polynomial $P_{i+1}(x)$ are both less than S, so each multiplication, if performed using fast evaluation of convolution (via the FFT) is bounded by the same constant multiple of $S \log S$. We perform K such multiplications, so our total time complexity is $O(KS \log S)$, as required.

Alternatively, pad all polynomials to a degree which is the smallest power κ of 2 larger than S; such κ is at most $2S$. Find the FFT of all of these polynomials, which will be K sequences of $\kappa < 2S$ many values. This requires taking K FFTs each in time $\Theta(S \log S)$ so in total $\Theta(KS \log S)$ many operations. Now multiply point-wise all these K sequences, using $(K - 1)S$ multiplications to get a single sequence of length S. Take the Inverse FFT of that sequence in time $\Theta(S \log S)$. Thus, in total the complexity is $\Theta(K\, S \log S)$.

(ii) The easiest way is, just as in the Celebrity Problem, to organize polynomials and their intermediate products into a complete binary tree, a trick which is useful in many situations which involve recursively operating on pairs of objects. To remind you of the construction of a complete binary tree, we first compute $m = \lfloor \log_2 K \rfloor$ and construct a perfect binary tree with $2^m \leq K$ leaves (i.e., a tree in which each node except the leaves has precisely two children and all the leaves are at the same depth). If $2^m < K$ add two children to each of the leftmost $K - 2^m$ leaves of such a perfect binary tree. In this way you obtain $2(K - 2^m) + (2^m - (K - 2^m)) = 2K - 2^{m+1} + 2^m - K + 2^m = K$ leaves exactly, but each leaf now has its pair, and the depth of each leaf is either

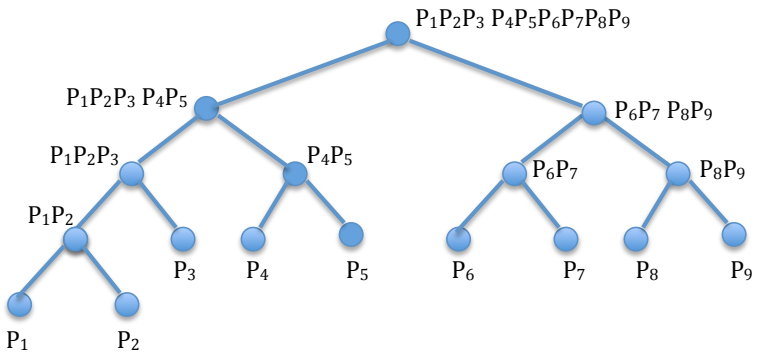


Figure 1: Here $K = 9$; thus, $m = \lfloor \log_2 K \rfloor = 3$ and we add two children to $K - 2^m = 9 - 2^3 = 1$ leaf of a perfect binary tree with 8 leaves. Thus obtained tree has 9 leaves.

$\lfloor \log_2 n \rfloor$ or $\lfloor \log_2 n \rfloor + 1$, see the picture on the next page. Each leaf is now assigned one of the polynomials and the inner nodes of the tree represent partial products of polynomials corresponding to the two children. Note that, with the possible exception of the deepest level $\lfloor \log_2 n \rfloor + 1$ of the tree (which in the example on the picture contains only two polynomials), the sum of the degrees of polynomials on each level is equal to the sum of the degrees of all K polynomials $P_i(x)$, i.e. is equal to S. Let d_1 and d_2 be the degrees of two polynomials corresponding to the children of a node at some level k; then the product polynomial is of degree $d_1 + d_2$ and thus it can be evaluated (using the FFT to compute the convolution of the sequences of the coefficients) in time $O((d_1 + d_2) \log(d_1 + d_2))$ which is also $O((d_1 + d_2) \log S)$, because clearly $\log(d_1 + d_2) \leq \log S$. Adding up such bounds for all products on level k we get the bound $O(S \log S)$, because the degrees of all polynomials at each of the levels add up precisely to S. Since we have $\lfloor \log K \rfloor + 1$ levels we get that the total amount of work is $O(\log K \cdot S \cdot \log S)$, as required.

13. You have a set of N coins in a bag, each having a value between 1 and M, where $M \geq N$. Some coins may have the same value. You pick two coins (**without replacement**) and record the sum of their values. Determine what possible sums can be achieved, in $O(M \log M)$ time.
For example, if there are $N = 3$ coins in the bag with values 1, 4 and 5 (so we could have $M = 5$), then the possible sums are 5, 6 and 9.

Hint: if the coins have values $v_1, \ldots, v_N$, how might you use the polynomial $x^{v_1} + \cdots + x^{v_N}$?

Solution: Form the polynomial $x^{v_1} + \cdots + x^{v_N}$ and use convolution to obtain its square. Drop all powers with coefficient equal to 1. The remaining powers provide the solution, because $x^{v_1}x^{v_2} = x^{v_1+v_2}$. Note that if $v_1 \neq v_2$ then this product appears with the coefficient at least 2 (once as $x^{v_1}x^{v_2}$ and once as $x^{v_2}x^{v_1}$; if v_1 appears only once, then x^{2v_1} also appears only once, thus also with a coefficient 1. So, because you pick coins without replacement you must drop all powers appearing with coefficient 1.

14. Consider the polynomial

$$P(x) = (x - \omega_{64}^0)(x - \omega_{64}^1)(x - \omega_{64}^2)\ldots(x - \omega_{64}^{63})$$

(a) Compute $P(0)$;

(b) What is the degree of $P(x)$? What is its coefficient of the highest degree of x present in $P(x)$?

(c) What are the values of $P(x)$ at the roots of unity of order 64?

(d) Can you represent $P(x)$ in the coefficient form without any computation?

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.

Solution:

(a)

$$\begin{aligned} P(0) &= (0 - \omega_{64}^0)(0 - \omega_{64}^1)\ldots(0 - \omega_{64}^{63}) \\ &= (-1)^{64}\omega_{64}^{0+1+\cdots+63} \\ &= \omega_{64}^{\frac{64\times 63}{2}} \\ &= \omega_{64}^{32*63} \\ &= \left(\omega_{64}^{32}\right)^{63} \\ &= (-1)^{63} \\ &= -1 \end{aligned}$$

(b) The degree of $P(x)$ is 64. The coefficient of the highest degree of x is 1 (there is only one term in the product of degree 64, which is all the x's multiplied together).

(c) The values of $P(x)$ at all roots of unity ω_{64}^k, $0 \leq k < 64$ are all 0.

(d) Since $P(x)$ is precisely the polynomial whose roots are the 64 roots of unity of order 64 and whose leading coefficient (in front of the largest degree) is 1, we must have $P(x) = x^{64} - 1$, by definition. Note that this clearly also provides an easier way to solve part (a)!

15. 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, $(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ 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. From there use induction to prove the general statement

Solution: In the example given above the indices of $(a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$ at the leaves are $(0, 4, 2, 6, 1, 5, 3, 7)$ or, in binary,

$$A_3 = (000, 100, 010, 110, 001, 101, 011, 111).$$

Compare this with the sequence $(0, 1, 2, 3, 4, 5, 6, 7)$ written in binary: $B_3 = (000, 001, 010, 011, 100, 101, 110, 111)$ and note that the bits of the i^{th} element of sequence A are the mirror image of the bits of the i^{th} element of sequence B. In general, to obtain a sequence B_{n+1} of integers in binary in the usual ordering we copy sequence B_n twice adding a zero to the left in the first copy of B_n and 1 to the left in the second copy of B_n. On the other hand, the sequence A_{n+1} contains first all even index terms followed by all odd index terms; thus, recursively, A_{n+1} is obtained from one copy of A_n with 0 concatenated to each element at its right end followed by another copy of A_n with 1 concatenated at the right of each element.

16. You are given a sequence

$$A = \langle a_0, a_1, \ldots, a_{n-1} \rangle$$

of length n and sequence

$$B = \langle 1, \underbrace{0, 0, \ldots, 0}_{k}, -1 \rangle$$

of length $k+2$, where $1 \leq k \leq n/4$.

(a) Compute the DFT of sequence B.

(b) Compute the convolution sequence $C = A * B$ in terms of the elements of sequence A.

(c) Show that in this particular case such a convolution can be computed in time $O(n)$.

Solution: Since $B = \langle 1, \underbrace{0, \ldots, 0}_{k}, 1 \rangle$, the corresponding polynomial is $P_B(x) = 1 - x^{k+1}$ and

$$\begin{aligned} DFT(B) &= \langle P_B(\omega_{k+2}^0), P_B(\omega_{k+2}^1), \ldots, P_B(\omega_{k+2}^{k+1}) \rangle \\ &= \langle 1 - \omega_{k+2}^{0\cdot(k+1)}, 1 - \omega_{k+2}^{1\cdot(k+1)}, \ldots, 1 - \omega_{k+2}^{(k+1)\cdot(k+1)} \rangle \\ &= \langle 0, 1 - \omega_{k+2}^{k+1}, 1 - \omega_{k+2}^{2(k+1)}, \ldots, 1 - \omega_{k+2}^{(k+1)^2} \rangle \end{aligned}$$

The polynomial corresponding to sequence A is $P_A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$; thus

$$P_A(x)P_B(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} - a_0 x^{k+1} - a_1 x^{k+2} - a_2 x^{k+3} + \ldots + a_{n-1} x^{n-1+k+1}$$

We now use the fact that $k \leq n/4$ to transform this expression into

$$\begin{aligned} P_A(x)P_B(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_k x^k + (a_{k+1} - a_0) x^{k+1} + (a_{k+2} - a_1) x^{k+2} + \\ \ldots + (a_{n-1} - a_{n-k-2}) x^{n-1} - a_{n-k-1} x^n - \ldots - a_{n-1} x^{n+k} \end{aligned}$$

Thus $C = A * B$ is given by

$$\begin{aligned} C = \langle a_0, a_1, \ldots, a_k, (a_{k+1} - a_0), (a_{k+2} - a_1), \ldots, (a_{n-1} - a_{n-k-2}), -a_{n-k-1}, \\ -a_{n-k}, \ldots - a_{n-2}, -a_{n-1} \rangle \end{aligned}$$

17. (a) Compute the DFT of the sequence $(1, -1, -1, 1)$ by **any method** you wish.

(b) Compute the linear convolution of sequences $(1, -1, -1, 1)$ and $(-1, 1, 1, -1)$ by **any method** you wish.

Solution:

(a) DFT of sequence $(1, -1, -1, 1)$ is just the sequence of values of the associated polynomial $P(x) = 1 - x - x^2 + x^3$ at the roots of unity of order 4, which are $1, i, -1, -i$. Thus $P(1) = 1-1-1+1 = 0$; $P(i) = 1-i-i^2+i^3 = 1 - i - (-1) - i = 2 - 2i$; $P(-1) = 1 - (-1) - (-1)^2 + (-1)^3 = 0$ and finally $P(-i) = 1 - (-i) - (-i)^2 + (-i)^3 = 1 + i + 1 + i = 2 + 2i$. Thus, $DFT(1, -1, -1, 1) = (0, 2 - 2i, 0, 2 + 2i)$.

(b) As defined in lecture slides #2, linear convolution of two sequences is just the sequence of the coefficients of the polynomial which is the product of the two polynomials associated with the two sequences. Thus we form $P(x) = 1-x-x^2+x^3$ and $Q(x) = -1+x+x^2-x^3$ and we simply multiply them by brute force because the polynomials are of such small degree: $P(x)Q(x) = -1+2x+x^2-4x^3+x^4+2x^5-x^6$. So linear convolution of these two sequences is $(1, -1, -1, 1) * (-1, 1, 1, -1) = (-1, 2, 1, -4, 1, 2, -1)$.

18. Assume that you are given a polynomial $P(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ and a polynomial $Q(x) = b_0 + b_1x + b_2x^2 + b_3x^3$ 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.

19. Recall that the DFT of a sequence

$$\langle a_0, a_1, \ldots, a_{n-1} \rangle$$

is the sequence of values

$$\langle P(\omega_n^0), P(\omega_n^1), P(\omega_n^2), \ldots, P(\omega_n^{n-1}) \rangle$$

of the associated polynomial $P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$. Compute directly (i.e., without using the FFT) and maximally simplify the DFT of the following sequences:

(a) $\langle 1, \underbrace{0, \ldots, 0}_{n-1} \rangle$

(b) $\langle \underbrace{0, \ldots, 0,}_{n-1} 1 \rangle$

(c) $\langle \underbrace{1, \ldots, 1}_{n} \rangle$

Solution:

(a) $\langle 1, \underbrace{0, \ldots, 0}_{n-1} \rangle$ produces a constant polynomial $P(x) = 1$. Thus $P(\omega_n^k) = 1$ for all k. Consequently, the DFT is equal to $\langle 1, 1, \ldots, 1 \rangle$.

(b) $\langle \underbrace{0, \ldots, 0,}_{n-1} 1 \rangle$ produces polynomial $P(x) = x^{n-1}$. Consequently, the DFT is equal to

$$\begin{aligned} \langle 1, \omega_n^{n-1}, \omega_n^{2(n-1)}, \ldots, \omega_n^{(n-1)(n-1)} \rangle &= \langle 1, \omega_n^n \omega_n^{-1}, \omega_n^{2n} \omega_n^{-2}, \ldots, \omega_n^{(n-1)n} \omega_n^{-(n-1)} \rangle \\ &= \langle 1, \omega_n^{-1}, \omega_n^{-2}, \ldots, \omega_n^{-(n-1)} \rangle \end{aligned}$$

(c) $\langle \underbrace{1, \ldots, 1}_{n} \rangle$ produces the polynomial

$$P(x) = 1 + x + x^2 + x^3 + \ldots + x^{n-1} = \frac{1 - x^n}{1 - x} \quad \text{for} \quad x \neq 1$$

which is 0 for all $\omega_n^k \neq 1$ because $(\omega_n^k)^n = (\omega_n^n)^k = 1$. If $x = \omega_n^0 = 1$ then $P(x) = 1 + \ldots + 1 = n$. Thus, the DFT of the sequence is equal to $\langle n, 0, 0, \ldots, 0 \rangle$.

20. 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.

21. You are given a polynomial $P(x) = A_0 + A_1x^{100} + A_2x^{200}$ 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 $P^*(y) = A_0 + A_1y^1 + A_2y^2$, 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

$$(A_0+A_1x^{100}+A_2x^{200})^2 = A_0^2+2A_0A_1x^{100}+(A_1^2+2A_0A_2)x^{200}+2A_1A_2x^{300}+A_2^2x^{400}$$

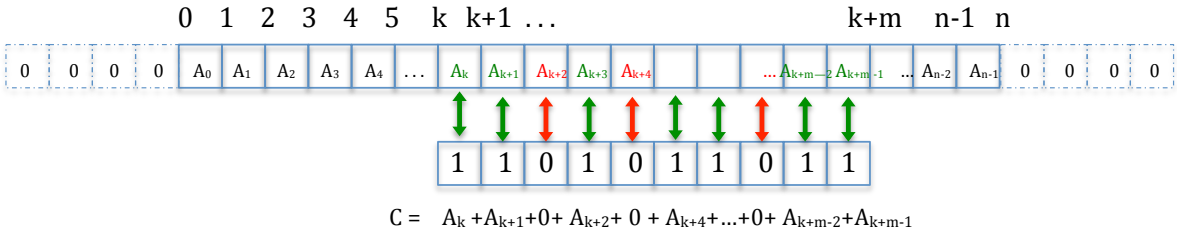
and that

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

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.

22. Assume you are given a map of a straight sea shore of length $100n$ meters as a sequence of $100n$ numbers such that A_i is the number of fish between the i^{th} meter of the shore and the $(i + 1)^{th}$ meter, $0 \leq i \leq 100n - 1$. You also have a net of length n meters but unfortunately it has holes in it. Such a net

is described as a sequence N of n ones and zeros, where 0's denote where the holes are. If you throw such a net starting at meter k and ending at meter $k+n$, then you will catch only the fish in one meter stretches of the shore where the corresponding bit of the net is 1; see the figure.



Find the spot where you should place the left end of your net in order to catch the largest possible number of fish using an algorithm which runs in time $O(n \log n)$. (30 pts)

Hint: Let N' be the net sequence N in the reverse order; look at the sequence $A * N'$.

Solution: We are given that the sea shore is described by the sequence $A = \langle A_0, A_1, \ldots, A_{100n-1} \rangle$, where A_i is the number of fish in the i^{th} meter of the shore, and the net is described by the sequence $N = \langle N_0, N_1, ..., N_{n-1} \rangle$, where N_i is 0 if there is a hole in the i^{th} meter of the net, and 1 otherwise. Hence, if we place the left end of our net at meter i, then the number of fish we will catch is given by $A_i N_0 + A_{i+1} N_1 + \ldots + A_{i+n-1} N_{n-1} = \sum_{k=0}^{n-1} A_{i+k} N_k$. Note that the left end of our net can be placed anywhere between meter $-n+1$ and meter $100n-1$ ($-n+1 \leq i \leq 100n-1$), but if any part of the net is placed outside the shore, that part does not yield any fish, so $A_i = 0$ for all values of i outside the range $[0, 100n-1]$. Since we want to catch as many fish as possible, we want to find the value of i that maximises $\sum_{k=0}^{n-1} A_{i+k} N_k$. Let us define a sequence N^* as the reversed sequence N, i.e., $N_i^* = N_{n-1-i}$ for all $0 \leq i \leq N-1$. Then the number of fish we catch if we throw the net starting at A_i is equal to

$$\begin{aligned} A_i N_0 + A_{i+1} N_1 + \ldots + A_{i+n-1} N_{n-1} &= A_i N_{n-1}^* + A_{i+1} N_{n-2}^* + \ldots + A_{i+n-1} N_0^* \\ &= \sum_{j+k=i+n-1} A_j N_k^* \end{aligned}$$

But the last sum is just the value of the convolution $A * N'$ at point $i+n-1$. Thus, we convolve sequences A and N^* and look for m such that $(A * N')[m]$

has the largest value which represents the largest number of fish you can catch, and solving $m = i + n - 1$ for i, i.e., $i = m - n + 1$ tells you where your net should start. Note that negative values of i indicate that the beginning of the net is on the shore and only a part of it is in the sea. Similarly, if $m > 100n$ this indicates that the right end of the net will be on the shore and only its left part in the sea.

23. (a) Compute the convolution $\langle 1, \underbrace{0, 0, \ldots, 0}_{k}, 1\rangle * \langle 1, \underbrace{0, 0, \ldots, 0}_{k}, 1\rangle$. (10 pts)

Solution: The sequence $\langle 1, \underbrace{0, 0, \ldots, 0}_{k}, 1\rangle$ corresponds to the polynomial $P(x) = 1 + x^{k+1}$, and the convolution of this sequence with itself is the sequence of coefficients of the polynomial $P(x)^2 = (1 + x^{k+1})^2 = 1 + 2x^{k+1} + x^{2k+2}$, i.e., the sequence $\langle 1, \underbrace{0, 0, \ldots, 0}_{k}, 2, \underbrace{0, 0, \ldots, 0}_{k}, 1\rangle$.

(b) Compute the DFT of the sequence $\langle 1, \underbrace{0, 0, \ldots, 0}_{k}, 1\rangle$. (10 pts)

Solution: The corresponding polynomial is $P(x) = 1 + x^{k+1}$, and hence

$$\begin{aligned} DFT(\langle 1, \underbrace{0, 0, \ldots, 0}_{k}, 1\rangle) &= \langle P(\omega_{k+2}^0), P(\omega_{k+2}^1), \ldots, P(\omega_{k+2}^{k+1})\rangle \\ &= \langle 1 + \omega_{k+2}^{0\cdot(k+1)}, 1 + \omega_{k+2}^{1\cdot(k+1)}, \ldots, 1 + \omega_{k+2}^{(k+1)\cdot(k+1)}\rangle \\ &= \langle 2, 1 + \omega_{k+2}^{1(k+1)}, 1 + \omega_{k+2}^{2(k+1)}, \ldots, 1 + \omega_{k+2}^{(k+1)^2}\rangle \end{aligned}$$

24. Find the sequence x satisfying $x * \langle 1, 1, -1\rangle = \langle 1, 0, -1, 2, -1\rangle$. (20 pts)

Hint: What polynomials correspond to the given sequences?

Solution: The sequence $\langle 1, 1, -1\rangle$ corresponds to the polynomial $Q(y) = 1 + y - y^2$, and the sequence $\langle 1, 0, -1, 2, -1\rangle$ corresponds to the polynomial $R(y) = 1 - y^2 + 2y^3 - y^4$. and x is simply the sequence of coefficients of the polynomial $P(y)$ that satisfies $P(y) \cdot Q(y) = R(y)$. Polynomial division gives $P(y) = 1 - y + y^2$, so $y = \langle 1, -1, 1\rangle$.