ADS Tutorial 3 - Solutions*

Instructor: Aris Filos-Ratsikas Tutor: Kat Molinet

October 20, 2025

Problem 1

Evaluate each of the following formulas involving complex numbers. Recall that $i^2 = -1$.

(a) $2i(3 - i)$

(b) $2i(i + 1)^2 + 4(i + 1)^3$

(c) $3i/(1 + i)$

Solution

(a) $2i(3 - i) = 6i - 2(i^2) = 6i - 2(-1) = 2 + 6i$

(b) $2i(i+1)^2+4(i+1)^3$ The are two approaches we can take to solving this problem. The first is to factor the polynomials and substitute $i^2 = -1$ as appropriate. But we could also convert our numbers into polar coordinates, since exponentiation is simpler (and generalises better) in polar form.

The factor and simplify approach:

$$\begin{aligned}
2i(i + 1)^2 + 4(i + 1)^3 &= 2i(i + 1)(i + 1) + 4(i + 1)(i + 1)(i + 1) \\
&= 2i(i^2 + 2i + 1) + 4(i + 1)(i^2 + 2i + 1) \\
&= 2i^3 + 4i^2 + 2i + 4(i^3 + 2i^2 + i + i^2 + 2i + 1) \\
&= 6i^3 + 12i^2 + 14i + 4 \\
&= 6i(-1) + 16(-1) + 14i + 4 \\
&= -12 + 8i
\end{aligned}$$

Cartesian → polar → Cartesian approach:

$$\begin{aligned}
2i(i + 1)^2 + 4(i + 1)^3 &= 2e^{i\pi/2}\big(\sqrt{2}e^{i\pi/4}\big)^2 + 4\big(\sqrt{2}e^{i\pi/4}\big)^3 \\
&= 2e^{i\pi/2}\big(2e^{i\pi/2}\big) + 4\big(2\sqrt{2}e^{i3\pi/4}\big) \\
&= 4e^{i\pi} + 8\sqrt{2}e^{i3\pi/4} \\
&= (-4) + 8\sqrt{2}\left(\frac{1}{\sqrt{2}}(-1 + i)\right) \\
&= -12 + 8i
\end{aligned}$$

*The solutions contain additional explanations that are not necessary, if you were to answer such a question in an exam.

(c) $3i/(1+i)$ To remove the imaginary number from the denominator of the fraction, we simply multiply the top and bottom by the *complex conjugate* of $1+i$:

$$\begin{aligned}\frac{3i}{1+i} &= \frac{3i}{1+i}\cdot\frac{1-i}{1-i}\\ &= \frac{3i-3i^2}{1-i^2}\\ &= \frac{3i-3(-1)}{1-(-1)}\\ &= \tfrac{3}{2}+\tfrac{3}{2}i\end{aligned}$$

Problem 2

Recall the *Discrete Fourier Transform (DFT)*, i.e., a sequence of (complex) numbers obtained by evaluating a polynomial of degree $n-1$ on each of the nth roots of unity. To apply a Divide-and-Conquer approach, in the lectures we assumed that n is a power of 2, i.e., that $n = 2^k$ for some non-negative integer k.

If we wanted to apply the DFT to a polynomial of degree $n-1$ which is not a power of 2, we could use *padding*. We first find n' which is the smallest power of 2 such that $n' > n$, and then we apply DFT on the polynomial of degree n', where the coefficients of the largest $n'-n$ terms are set to 0.

Provide an algorithm for the computation of n' and an argument about its running time. Also, argue that the running time of the Divide-and-Conquer FFT algorithm for computing the DFT is $O(n \log n)$.

Solution

Recall that the Discrete Fourier Transform of a sequence of n complex numbers $p_0, p_1, \cdots, p_{n-1}$ is defined to be the sequence of complex numbers

$$P(1),\ P(\omega_n),\ P(\omega_n^2), \cdots,\ P(\omega_n^{n-1})$$

obtained by evaluating the polynomial $P(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{n-1}x^{n-1}$ on each of the nth roots of unity.

If n is not a power of 2, than we can define $n' := 2^{\lceil log_n n \rceil}$ to be the smallest power of 2 greater than n. Note that this definition immediately implies that $n' < 2n$, since otherwise there must exist another power of 2 between n and n'. We construct the polynomial $p'(x)$ as follows:

$$p'(x) := p(x) + p_n x^n + p_{n+1}x^{n+1} + \cdots + p_{n'-1}x^{n'-1},$$

where $p_n = p_{n+1} = \cdots = p_{n'-1} = 0$. This padding operation takes time $O(n')$, and $p'(x) = p(x)$ for any number x. (I.e., evaluating $p'(x)$ at any number x gives you the same value as evaluating $p(x)$ at x.) From here, we can simply apply the divide-and-conquer approach outlined in lecture. In the original DFT, we evaluate a polynomial of degree $m-1$ at m distinct points in time $O(m \log m)$. Here, we evaluate a polynomial of degree $n'-1$ at n points, which, for $n' < 2n$, should have runtime no more than $O(n' \log n') \leq O(2n \log(2n)) = O(2n \log n + 2n \log 2) = O(n \log n)$.

Problem 3

A. Compute $\mathrm{DFT}_4(0,1,2,3)$ (i.e., evaluate the polynomial $x + 2x^2 + 3x^3$ on each of the 4th roots of unity). You don't have to apply the FFT algorithm, you may do the calculation directly.

B. Use the FFT algorithm to mutiply the polynomials $p(x) = x - 4$ and $q(x) = x^2 - 1$. To do this, use the following steps:

(a) First figure out what the degree of $p(x) \cdot q(x)$ is. Let $n = \deg(p(x) \cdot q(x)) + 1$. If this is not a power of two, work as in Problem 2 to pick an n that is a power of 2 by padding.

(b) Write down each of the nth roots of unity.

(c) Evaluate both $p(x)$ and $q(x)$ at each of the nth roots of unity x_j, and then compute $p(x_j) \cdot q(x_j)$ for every j. To save time, you may do this directly as in Part A of this question, without using the FFT recurrence. We will the FFT recurrence in the next step, for the polynomial interpolation.

(d) Recover the coefficients of the polynomial $p(x) \cdot q(x)$. To do that, define an appropriate polynomial $d(x)$ as presented in the lectures and apply FFT to $d(x)$ to evaluate it at the nth roots of unity.

Solution

A. The 4th complex roots of unity are the solutions to the equation $z^4 = 1$. They 1, ω, ω^2, ω^3, where $\omega = e^{i2\pi/4} = e^{i\pi/2}$, and evaluate to $\{1, i, -1, -i\}$. Substituting each of these values into the polynomial $f(x) = x + 2x^2 + 3x^3$ gives the following:

$$
\begin{aligned}
f(\omega^0) &= f(1) = 1 + 2(1)^2 + 3(1)^3 = 6 \\
f(\omega^1) &= f(i) = i + 2(i)^2 + 3(i)^3 = i + 2(-1) + 3i(-1) = -2 - 2i \\
f(\omega^2) &= f(-1) = -1 + 2(-1)^2 + 3(-1)^3 = -1 + 2 - 3 = -2 \\
f(\omega^3) &= f(-i) = -i + 2(-i)^2 + 3(-i)^3 = -i + 2(-1) - 3i(-1) = -2 + 2i
\end{aligned}
$$

B. (a) The degree of the product of any two polynomials pq is $\deg(pq) = \deg(p) + \deg(q)$. Since $p(x)$ and $q(x)$ have degrees 1 and 2, respectively, the degree of pq is simply $1 + 2 = 3$. The number n is defined to be one more than the degree of pq; thus, $n = 4$, which is indeed a power of 2.

(b) As in part A, $n = 4$. Thus, our roots of unity are the same as before:

$$\{\omega^0, \omega^1, \omega^2\, \omega^3\} = \{(e^{i\pi/2})^0, (e^{i\pi/2})^1, (e^{i\pi/2})^2, (e^{i\pi/2})^3\} = \{1, i, -1, -i\}.$$

(c) Evaluating both $p(x) = x - 4$ and $q(x) = x^2 - 1$ at each 4th root of unity gives:

$$
\begin{array}{lll}
p(1) = (1) - 4 = -3 & q(1) = (1)^2 - 1 = 0 & pq(1) = -3 \cdot 0 = 0 \\
p(i) = (i) - 4 = -4 + i & q(i) = (i)^2 - 1 = -2 & pq(i) = (-4 + i)(-2) = 8 - 2i \\
p(-1) = (-1) - 4 = -5 & q(-1) = (-1)^2 - 1 = 0 & pq(-1) = -5 \cdot 0 = 0 \\
p(-i) = (-i) - 4 = -4 - i & q(-i) = (-i)^2 - 1 = -2 & pq(-i) = (-4 - i)(-2) = 8 + 2i
\end{array}
$$

Bonus*: Even though the instructions didn't ask us to apply the FFT recurrence when evaluating p and q, below we show how to do this for the polynomial $q(x) = x^2 - 1$. First, we recursively subdivide our polynomial into even and odd components, until each component is simply a constant function. For $n = 4$, this requires $\log_2(4) = 2$ subdivisions.

$$
\begin{aligned}
q(x) &= x^2 - 1 \\
&= Q_e(x^2) + xQ_o(x^2) \\
&= [Q_{ee}(x^4) + x^2 Q_{eo}(x^4)] + x[Q_{oe}(x^4) + x^2 Q_{oo}(x^4)],
\end{aligned}
$$

for $Q_e(y) := y - 1,\ Q_o(y) := 0$ and for $Q_{ee}(z) := -1,\ Q_{eo}(z) := 1,\ Q_{oe}(z) = Q_{oo}(z) := 0$.

Sanity check:

$$
\begin{aligned}
q(x) &= [Q_{ee}(x^4) + x^2 Q_{eo}(x^4)] + x[Q_{oe}(x^4) + x^2 Q_{oo}(x^4)] \\
&= [(-1) + x^2(1)] + x[(0) + x^2(0)] \\
&= x^2 - 1.
\end{aligned}
$$

Now, we evaluate the expanded version of $q(x)$ at each the 4th roots of unity: We see that $Q_{ee}(x^4)$, $Q_{eo}(x^4)$, $Q_{oe}(x^4)$, $Q_{oo}(x^4)$ are all constants, and so evaluate to the same value at each of the 4th roots of unity. So we just need to evaluate $q(x) = Q_e(x^2) + xQ_o(x^2)$ at each of the 4th roots of unity.

$$\begin{aligned}
q(1) &= Q_e(1^2) + (1)Q_o(1^2) = Q_e(1) + Q_o(1) \\
q(i) &= Q_e(i^2) + (i)Q_o(i^2) = Q_e(-1) + iQ_o(-1) \\
q(-1) &= Q_e((-1)^2) + (-1)Q_o((-1)^2) = Q_e(1) - Q_o(1) \\
q(-i) &= Q_e((-i)^2) + (-i)Q_o((-i)^2) = Q_e(-1) - iQ_o(-1)
\end{aligned}$$

Above, we see that for even/odd evaluation pairs, we can reuse the computations for $Q_e()$ and $Q_o()$. We compute that $Q_e(1) = 0$, $Q_e(-1) = -2$, and $Q_o(1) = Q_o(-1) = 0$. Thus, we have:

$$\begin{aligned}
q(1) &= Q_e(1) + Q_o(1) = 0 \\
q(i) &= Q_e(-1) + iQ_o(-1) = -2 \\
q(-1) &= Q_e(1) - Q_o(1) = 0 \\
q(-i) &= Q_e(-1) - iQ_o(-1) = -2,
\end{aligned}$$

which is consistent with what we computed originally.

(d) For illustrative purposes, we first show how to solve this problem *without* applying the FFT recurrence, and then *with* the FFT recurrence.

Without the FFT recurrence: From lecture, we have, for $C(x) := p(x)q(x)$ that

$$\begin{aligned}
D(x) &= \sum_{s=0}^{3} C(\omega_4^s) \cdot x^s \\
&= C(\omega_4^0)x^0 + C(\omega_4^1)x^1 + C(\omega_4^2)x^2 + C(\omega_4^3)x^3 \\
&= C(1) + C(i)x + C(-1)x^2 + C(-i)x^3 \\
&= (8-2i)x + (8+2i)x^3.
\end{aligned}$$

Now that we've defined $D(x)$, we can evaluate at each of the 4th roots of unity to compute the coefficients $c_s := \frac{1}{4}D(\omega_4^{4-s})$:

$$\begin{aligned}
c_0 &:= \tfrac{1}{4}D(\omega_4^4) = \tfrac{1}{4}D(1) = \tfrac{1}{4}((8-2i)(1) + (8+2i)(1)^3) = 4 \\
c_1 &:= \tfrac{1}{4}D(\omega_4^3) = \tfrac{1}{4}D(-i) = \tfrac{1}{4}((8-2i)(-i) + (8+2i)(-i)^3) = -1 \\
c_2 &:= \tfrac{1}{4}D(\omega_4^2) = \tfrac{1}{4}D(-1) = \tfrac{1}{4}((8-2i)(-1) + (8+2i)(-1)^3) = -4 \\
c_3 &:= \tfrac{1}{4}D(\omega_4^1) = \tfrac{1}{4}D(i) = \tfrac{1}{4}((8-2i)(i) + (8+2i)(i)^3) = 1
\end{aligned}$$

This gives us an overall product of

$$\begin{aligned}
p(x)q(x) &= c_0 + c_1x + c_2x^2 + c_3x^3 \\
&= 4 - x - 4x^2 + x^3
\end{aligned}$$

With the FFT recurrence: Here, we define the polynomial $D(x)$ just as before. But now, when we evaluate $D(x)$ at each of the 4th roots of unity, we use the FFT recurrence to recursively subdivide $D(x)$. (This is just like what we did in the "bonus" section of part (c), except with $D(x)$ as the polynomial instead of $q(x)$.) To subdivide $D(x)$, we can write:

$$\begin{aligned}
D(x) &= (8-2i)x + (8+2i)x^3 \\
&= D_e(x^2) + xD_o(x^2) \\
&= [D_{ee}(x^4) + x^2D_{eo}(x^4)] + x[D_{oe}(x^4) + x^2D_{oo}(x^4)],
\end{aligned}$$

for $D_e(y) := 0$, $D_o(y) := (8-2i) + (8+2i)y$ and for $D_{ee}(z) = D_{eo}(z) := 0$ and $D_{oe}(z) := 8-2i$, $D_{oo}(z) := 8+2i$.

Sanity check:

$$\begin{aligned} D(x) &= [D_{ee}(x^4) + x^2 D_{eo}(x^4)] + x[D_{oe}(x^4) + x^2 D_{oo}(x^4)] \\ &= [(0) + x^2(0)] + x[(8-2i) + x^2(8+2i)] \\ &= (8-2i)x + (8+2i)x^3. \end{aligned}$$

Now, we evaluate the expanded version of $D(x)$ at each the 4th roots of unity: We see that $D_{ee}(x^4)$, $D_{eo}(x^4)$, $D_{oe}(x^4)$, $D_{oo}(x^4)$ are all constants, and so evaluate to the same value at each of the 4th roots of unity. So we just need to evaluate $D(x) = D_e(x^2) + xD_o(x^2)$ at each of the 4th roots of unity.

$$\begin{aligned} D(1) &= D_e(1^2) + (1)D_o(1^2) = D_e(1) + D_o(1) \\ D(i) &= D_e(i^2) + (i)D_o(i^2) = D_e(-1) + iD_o(-1) \\ D(-1) &= D_e((-1)^2) + (-1)D_o((-1)^2) = D_e(1) - D_o(1) \\ D(-i) &= D_e((-i)^2) + (-i)D_o((-i)^2) = D_e(-1) - iD_o(-1) \end{aligned}$$

Above, we see that for even/odd evaluation pairs, we can reuse the computations for $D_e()$ and $D_o()$. We compute that $D_e(1) = D_e(-1) = 0$, and $D_o(1) = 16$ and $D_o(-1) = -4i$. Thus, we have:

$$\begin{aligned} c_0 &:= \tfrac{1}{4}D(\omega_4^4) = \tfrac{1}{4}D(1) = \tfrac{1}{4}(D_e(1) + D_o(1)) = 4 \\ c_1 &:= \tfrac{1}{4}D(\omega_4^3) = \tfrac{1}{4}D(-i) = \tfrac{1}{4}(D_e(-1) - iD_o(-1)) = -1 \\ c_2 &:= \tfrac{1}{4}D(\omega_4^2) = \tfrac{1}{4}D(-1) = \tfrac{1}{4}(D_e(1) - D_o(1)) = -4 \\ c_3 &:= \tfrac{1}{4}D(\omega_4^1) = \tfrac{1}{4}D(i) = \tfrac{1}{4}(D_e(-1) + iD_o(-1)) = 1, \end{aligned}$$

which is consistent with what we computed originally. As before, this gives us an overall product of

$$\begin{aligned} p(x)q(x) &= c_0 + c_1 x + c_2 x^2 + c_3 x^3 \\ &= 4 - x - 4x^2 + x^3. \end{aligned}$$

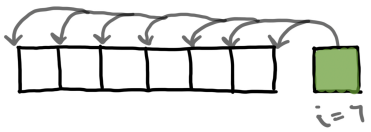
Problem 4

Recall the InsertionSort algorithm from the lectures (see also CLRS Chapter 2.1). The algorithm maintains the following invariant: at round i, when element $A(i)$ is examined, elements $A(1), \ldots, A(i-1)$ are sorted. Then element $A(i)$ is compared with $A(i-1), A(i-2), \ldots$ until the correct position for it is found, and then it is inserted in that position. The worst-case running time of InsertionSort is $\Theta(n^2)$.

What is the average-case running time of the algorithm, assuming every input sequence is equally likely? Prove an appropriate asymptotic bound. You may assume that the running time of the algorithm is the number of comparisons that it performs during its execution.

Solution

Let's start with an example. Suppose we examine an execution of InsertionSort on a list A with $i = 7$. Assume that elements $A[1], \cdots, A[6]$ are already sorted, and we're trying to figure out where $A[7]$ should be inserted. There are seven possible locations where $A[7]$ might be inserted; see the image below.
Assuming every input sequence is equally, the probability that $A[7]$ is inserted into any particular location



is 1/7. How many comparisons are required for each insert position? The first position (counting from the right) requires only a single comparison, with $A[6]$. The second position requires two comparisons, one with $A[6]$ and one with $A[5]$. In general, the kth position (counted from the right) requires k comparisons – *except* when $k = 7$, which only requires $k - 1 = 6$ comparisons. (After all, there are only 6 elements to which $A[7]$ *can* be compared.) Let X_i be a random variable indicating the number of comparisons made when inserting element $A[i]$. Then the expected number of comparisons made when inserting $A[7]$ is hence as follows:

$$\begin{aligned}\mathbb{E}[X_7] &= \left(\sum_{k=1}^{6} k \cdot \mathbb{P}(X_7 = k)\right) + 6 \cdot P(X_7 = 7) \\ &= \frac{1}{7}\left(1+2+3+4+5+6+6\right) \\ &= \frac{1}{7}\left(\left(\sum_{j=1}^{7} j\right) - 1\right)\end{aligned}$$

And more generally, we have that

$$\mathbb{E}[X_i] = \frac{1}{i}\left(\left(\sum_{j=1}^{i} j\right) - 1\right)$$

(As a sanity check, we can confirm by hand that our formula outputs the correct number of expected comparisons for both $i = 1$ and $i = n$.) By linearity of expectations, the overall number of comparisons in an execution of InsertSort is thus:

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] &= \sum_{i=1}^{n} \mathbb{E}[X_i] \\ &= \sum_{i=1}^{n} \frac{1}{i}\left(\left(\sum_{j=1}^{i} j\right) - 1\right) \\ &= \sum_{i=1}^{n} \frac{1}{i}\left(\frac{i}{2}(i+1) - 1\right) \\ &= \sum_{i=1}^{n} \left(\frac{i+1}{2} - \frac{1}{i}\right) \\ &= \frac{1}{2}\sum_{i=1}^{n} (i+1) - \frac{1}{2}\sum_{i=1}^{n} \frac{1}{i} \\ &= \frac{1}{2}\left(\frac{n}{2}(2 + (n+1))\right) - \frac{1}{2}\sum_{i=1}^{n} \frac{1}{i} \\ &= \frac{n}{4}(n+3) - \frac{1}{2}\sum_{i=1}^{n} \frac{1}{i}\end{aligned}$$

From here, we note that the first term is quadratic, while the second term is $-\frac{1}{2}H(n)$, where $H(n)$ is the nth harmonic number. For any $n \geq 1$, $H(n) \geq 1$. Thus, we have:

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \frac{n}{4}(n+3) - \frac{1}{2}H(n) \leq \frac{n}{4}(n+3) - \frac{1}{2} = O(n^2).$$

Thus, the average-case runtime of InsertSort is asymptotically no better than its worst-case runtime.