ADS Tutorial 3

Instructor: Aris Filos-Ratsikas Tutor: Kat Molinet

October 13, 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) $3i/(1 + i)$

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 $O(n \log n)$.

Problem 3

A. Compute $\text{DFT}_4(0, 1, 2, 3)$ (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.

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.