CSC4120 Spring 2024 - Written Homework 2

Yohandi 120040025
Andrew Nathanael 120040007

January 31, 2024

Divide and conquer

Problem 1.

Suppose we change slightly the 2D peak finding algorithm for a $n \times n$ array A of integers. We consider the elements of the window (of the 3 rows and the 3 columns as before) and check if any of these is a peak. If yes, then we are done (we found a global peak). Else we consider the maximum element M among these and choose the order $n/2 \times n/2$ sub-array A' that contains an element next to M that is strictly greater than M (as before). Then we repeat the procedure for the array A'. Answer the questions below. Indicate the reason.

Questions

(a) The algorithm is correct, i.e., it always finds a true global peak (a peak of the original matrix A).

(b) The termination condition might report a peak which is not a true global peak.

(c) The algorithm might terminate without reporting a peak.

(d) The complexity of the algorithm is

(i) $\Theta(n \log n)$

(ii) $\Theta(n^2)$

(iii) $\Theta(n)$

(a) The statement is true. The current algorithm additionally checks more elements without losing the fundamental properties that make the original 2D peak finding algorithm work. As the original 2D peak finding algorithm is known to correctly solve the problem, the modification preserves the correctness.

(b) The statement is false. As explained previously, the correctness is preserved after the modification; hence, the termination condition will report a peak that is a true global peak.

(c) The statement is false. Given the algorithm's design and the proof that a peak always exists (either by moving upwards in value or finding a local maximum), it should always report a peak. The algorithm's recursive and divide-and-conquer nature ensures that it terminates when a peak is found, and it's constructed to ensure a peak is always found.

(d) Let $T(n)$ be the complexity of the algorithm that solves a $n \times n$ array. Then,

$$T(n) \leq T(\frac{n}{2}) + cn = T(\frac{n}{2}) + \Theta(n) \underset{\text{by Master's Theorem}}{\Rightarrow} T(n) = \Theta(n)$$

This implies that $T(n) = \mathcal{O}(n)$ and $T(n) = \Omega(n)$.

(i) The statement is false. As $T(n) = \mathcal{O}(n)$, $\exists c_0, n_0$ such that $T(n) \leq c_0 n, \forall n \geq n_0$; consequently, $\nexists c_1, n_1$ such that $T(n) \geq c_1 n \log n, \forall n \geq n_1$. Let b be the base of the log. The previous implication is made as if there exists such c_1 and n_1, then for $n = \max(1, b, n_0, n_1) + \lim_{\epsilon \to 0} \epsilon$, $T(n) > c_0 n$. This contradicts with the fact that $T(n) = \mathcal{O}(n)$.

(ii) The statement is false. As $T(n) = \mathcal{O}(n)$, $\exists c_0, n_0$ such that $T(n) \leq c_0 n, \forall n \geq n_0$; consequently, $\nexists c_1, n_1$ such that $T(n) \geq c_1 n^2, \forall n \geq n_1$. The previous implication is made as if there exists such c_1 and n_1, then for $n = \max(1, n_0, n_1) + \lim_{\epsilon \to 0} \epsilon$, $T(n) > c_0 n$. This contradicts with the fact that $T(n) = \mathcal{O}(n)$.

(iii) The statement is true as $T(n) = \Theta(n)$.

Problem 2.

Consider the simple algorithm for finding a peak in a 1D array A of size n, where we check each element $A[i]$ in the order of increasing index. Answer the questions below. Indicate the reason.

Questions

(a) If the elements of A are already sorted in decreasing order (i.e., $A[i] \geq A[i+1]$), then the complexity of the algorithm is:

(i) $\Theta(1)$
(ii) $\Theta(n)$
(iii) $O(n)$
(iv) $\Omega(1)$
(v) $\Omega(2)$
(vi) $\Omega(n)$

(b) Assume that the values of the elements are chosen with equal probability from the set of two numbers, say $\{0, 1\}$. Then the average number of steps our algorithm takes is

(i) $O(1)$
(ii) $O(n)$
(iii) $\Theta(n)$

(c) Assume that the values of the n elements are integers chosen with equal probability from $(-\infty, +\infty)$, and hence we can for simplicity assume that for any given number $A[i]$ the value of $A[i+1]$ is with probability $1/2$ larger. Then the average number of steps our algorithm takes is

(i) $O(1)$

(ii) $O(n)$

(iii) $\Theta(n)$

(a) The time complexity for the algorithm when elements of A is sorted in decreasing order:

(i) True. If the elements of A are sorted in decreasing order, the first element $A[0]$ is guaranteed to be a peak, as it is greater than or equal to $A[1]$. Therefore, the algorithm finds a peak in constant time, say k (defined for below parts).

(ii) False. $\nexists c_0, n_0$ such that $c_0 n \leq k, \forall n \geq n_0$. This is because when $n > \max(n_0, \frac{k}{c_0})$, $c_0 n > k$. By definition, the time complexity couldn't be $\Omega(n)$.

(iii) True. Because $\Theta(1)$ is a subset of $O(n)$, although it is not the tightest bound.

(iv) True. $\exists c_0 = k, n_0$ such that $c_0 \leq k, \forall n \geq n_0$. By definition, the time complexity is $\Omega(1)$.

(v) True. $\exists c_0 = \frac{k}{2}, n_0$ such that $2c_0 \leq k, \forall n \geq n_0$. By definition, the time complexity is $\Omega(2)$.

(vi) False. As proven in part (ii).

(b) Remark: as long as we find an element with value 1, it will be the peak that we will return. The problem follows a geometric series where $p = \frac{1}{2}$ (random number between the two numbers). The expected value is $\frac{1}{p} = 2$.

(i) True. $\exists c_0 = 2, n_0$ such that $2 \leq c_0, \forall n \geq n_0$. By definition, the time complexity is $\mathcal{O}(1)$.

(ii) True. $\exists c_0 = 2, n_0 = 1$ such that $2 \leq c_0 n, \forall n \geq n_0$. By definition, the time complexity is $\mathcal{O}(n)$.

(iii) False. $\nexists c_0, n_0$ such that $c_0 n \leq 2, \forall n \geq n_0$. This is because when $n > \max(n_0, \frac{2}{c_0})$, $c_0 n > 2$. By definition, the time complexity coudln't be $\Theta(n)$.

(c) We will have a $\frac{1}{2}$ probability to get $A[0]$ to be the peak (that is $A[0] \geq A[1]$). If that doesn't happen, that means $A[0] < A[1]$, we will also ahve a $\frac{1}{2}$ probability to get $A[1]$ to be the peak (as $A[1] > A[0]$, we only care about $A[1] \leq A[2]$). This means, the problem follows a geometric series where $p = \frac{1}{2}$. The expected value is $\frac{1}{p} = 2$.

(i) True. $\exists c_0 = 2, n_0$ such that $2 \leq c_0, \forall n \geq n_0$. By definition, the time complexity is $\mathcal{O}(1)$.

(ii) True. $\exists c_0 = 2, n_0 = 1$ such that $2 \leq c_0 n, \forall n \geq n_0$. By definition, the time complexity is $\mathcal{O}(n)$.

(iii) False. $\nexists c_0, n_0$ such that $c_0 n \leq 2, \forall n \geq n_0$. This is because when $n > \max(n_0, \frac{2}{c_0})$, $c_0 n > 2$. By definition, the time complexity coudln't be $\Theta(n)$.

Problem 3.

Consider the computational problem P for which we know two algorithms $A1$ and $A2$ that solve it. We know for $A1$ that its complexity is $\Theta(n \log n)$ and for $A2$ that it is $\Theta(n^2)$. Answer the questions below. Indicate the reason.

Questions

For the problem P, we know that its complexity is

(a) $O(n \log n)$

(b) $O(n^2)$

(c) $\Omega(n \log n)$

(d) $\Omega(n^2)$

(a) True. Algorithm $A1$ has a complexity of $\Theta(n \log n)$, in which it also establishes $n \log n$ as an upper bound for the problem's complexity, because it shows that the problem can be solved in at most $n \log n$ time. Hence, the complexity of problem P is $O(n \log n)$.

(b) True. $O(n^2)$ is a broader upper bound that encompasses $\Theta(n \log n)$. Since $n \log n$ grows slower than n^2, any algorithm with a complexity of $O(n \log n)$ is also $O(n^2)$. However, this upper bound is not tight because it allows for the possibility of the problem being solved in n^2 time or less, which includes the $n \log n$ time complexity of $A1$.

(c) False. $\Omega(n \log n)$ suggests a lower bound, meaning the problem requires at least $n \log n$ time to solve in the worst case. However, we cannot conclude this solely based on the given algorithms. The existence of a faster algorithm (with complexity less than $n \log n$) cannot be ruled out. Therefore, we cannot assert $\Omega(n \log n)$ as the complexity of problem P without further evidence.

(d) False. This would imply that the problem requires at least n^2 time to solve in the worst case, which contradicts the existence of $A1$ with a better (lower) complexity of $\Theta(n \log n)$. Since $\Theta(n \log n)$ is an achievable complexity for solving the problem, the problem's complexity cannot have a lower bound of n^2.

Problem 4.

Answer the questions below. Indicate the reason.

(a) The most common algorithm to solve linear programs is the Simplex method. It is shown that it is has exponential complexity. Does this imply that it should never be used to solve large linear programs?

(b) Simplex has proved in practice to work even for large programs. You guess this is because

(i) its best case complexity is constant? (T/F)

(ii) its average complexity is polynomial? (T/F)

(a) The method does indeed have exponential time complexity in the worst case. However, this does not imply that it should never be used for large linear programs. The key distinction here is between the worst-case complexity and the average-case complexity. Similar to data structures like hash tables or treaps, where worst-case scenarios are rarely encountered in practice, the Simplex method's exponential complexity is a theoretical limit that does not often materialize in real-world applications. In practical scenarios, the algorithm's behavior is much more efficient, making it a viable option even for large-scale problems.

(b) (i) False. The best-case complexity is not a significant factor. It does not typically represent the usual performance of the algorithm in real-world applications.

(ii) True. Even in large-scale applications, the method is practically success as its average complexity tends to be polynomial. The algorithm performs efficiently across a wide range of problems on average. The exponential worst-case complexity of the Simplex method is a theoretical concern that does not frequently manifest in typical use cases.

Problem 5.

You are given two sorted lists of size m and n. Give an $O(\log m + \log n)$ time algorithm for computing the k-th smallest element in the union of the two lists.

Let's define:

- L_1 as the list that has size of m and L_2 as the list that has size of n.
- m_l, m_r be the range divider of indexes in L_1 that we are considering, that is $L_1[m_l, m_r - 1]$, where $0 \leq m_l \leq m_r < m$.
- n_l, n_r be the range divider of indexes in L_2 that we are considering, that is $L_2[n_l, n_r - 1]$, where $0 \leq n_l \leq n_r < n$.
- m_m be the middle index of the considered range in L_1, that is $\frac{m_l + m_r}{2}$
- n_m be the middle index of the considered range in L_2, that is $\frac{n_l + n_r}{2}$.

The algorithm will be based on divide and conquer approach. Define smallest(k, m_l, m_r, n_l, n_r) as the k-th smallest number in the union of $L_1[m_l, m_r - 1]$ and $L_2[n_l, n_r - 1]$. We are interested in smallest$(k, 0, m, 0, n)$. The algorithm is as follows:

- If $m_r = m_l$, then we simply return $L_2[n_l + k - 1]$ (as we only consider L_2).
- If $n_r = n_l$, then we simply return $L_1[m_l + k - 1]$ (as we only consider L_1).

- If $(m_m - m_l) + (n_m - n_l) < k$, which means the total elements of both halved lists is less then k implying the k-th smallest element cannot be in the left halves of both lists, then
  - If $L_1[m_m] > L_2[n_m]$, then we return smallest$(k - (n_m - n_l) - 1, m_l, m_r, n_m, n_r)$ (because all elements in $L_2[n_l, n_m - 1]$ are guaranteed to be smaller than $L_1[m_m]$ and cannot include the k-th smallest element.)
  - Else, we return smallest$(k - (m_m - m_l) - 1, m_m, m_r, n_l, n_r)$ (because all elements in $L_1[m_l, m_m - 1]$ are guaranteed to be smaller than $L_2[n_m]$ and cannot include the k-th smallest element.)
- Else it implies the k-th smallest element is in the left halves of one of the lists.
  - If $L_1[m_m] > L_2[n_m]$, then we return smallest(k, m_l, m_m, n_l, n_r) (because the k-th smallest element must be within the first k elements of the merged list, and $L_1[m_m]$ and its right elements are too large to be among these.)
  - Else, we return smallest(k, m_l, m_r, n_l, n_m) (because the k-th smallest element must be in the left half of L_2 or somewhere in L_1.)

  In each transition, the range of $[m_l, m_r)$ or $[n_l, n_r)$ will be halved. In a worst case scenario, both range will be about a constant, making the algorithm runs in $c(\log_2(m_r - m_l) + \log_2(n_r - n_l)) = \mathcal{O}(\log m + \log n)$.

Problem 6.

Practice with the Fast Fourier Transform.

(a) What is the FFT of $(1, 0, 0, 0)$? What is the appropriate value of ω is this case? And of which sequence is $(1, 0, 0, 0)$ the FFT?

(b) Repeat for $(1, 0, 1, -1)$.

(a) In this case, we choose $\omega = e^{\frac{-2\pi i}{4}}$.

$$
\begin{aligned}
P(0) &= 1 \cdot \omega^{0\cdot 0} + 0 \cdot \omega^{1\cdot 0} + 0 \cdot \omega^{2\cdot 0} + 0 \cdot \omega^{3\cdot 0} = 1 \\
P(1) &= 1 \cdot \omega^{0\cdot 1} + 0 \cdot \omega^{1\cdot 1} + 0 \cdot \omega^{2\cdot 1} + 0 \cdot \omega^{3\cdot 1} = 1 \\
P(2) &= 1 \cdot \omega^{0\cdot 2} + 0 \cdot \omega^{1\cdot 2} + 0 \cdot \omega^{2\cdot 2} + 0 \cdot \omega^{3\cdot 2} = 1 \\
P(3) &= 1 \cdot \omega^{0\cdot 3} + 0 \cdot \omega^{1\cdot 3} + 0 \cdot \omega^{2\cdot 3} + 0 \cdot \omega^{3\cdot 3} = 1
\end{aligned}
$$

The FFT of $(1, 0, 0, 0)$ is $(1, 1, 1, 1)$.

Let the FFT of (a, b, c, d) is $(1, 0, 0, 0)$, then

$$\begin{aligned}
P(0) &= a \cdot \omega^{0\cdot 0} + b \cdot \omega^{1\cdot 0} + c \cdot \omega^{2\cdot 0} + d \cdot \omega^{3\cdot 0} = a + b + c + d = 1 \\
P(1) &= a \cdot \omega^{0\cdot 1} + b \cdot \omega^{1\cdot 1} + c \cdot \omega^{2\cdot 1} + d \cdot \omega^{3\cdot 1} = a - bi - c + di = 0 \\
P(2) &= a \cdot \omega^{0\cdot 2} + b \cdot \omega^{1\cdot 2} + c \cdot \omega^{2\cdot 2} + d \cdot \omega^{3\cdot 2} = a - b + c - d = 0 \\
P(3) &= a \cdot \omega^{0\cdot 3} + b \cdot \omega^{1\cdot 3} + c \cdot \omega^{2\cdot 3} + d \cdot \omega^{3\cdot 3} = a + bi - c - di = 0
\end{aligned}$$

We obtain $(a, b, c, d) = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$.

(b) In this case, we also choose $\omega = e^{\frac{-2\pi i}{4}}$.

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

The FFT of $(1, 0, 1, -1)$ is $(1, -i, 3, i)$.

Let the FFT of (a, b, c, d) to be $(1, 0, 1, -1)$, then

$$\begin{aligned}
P(0) &= a \cdot \omega^{0\cdot 0} + b \cdot \omega^{1\cdot 0} + c \cdot \omega^{2\cdot 0} + d \cdot \omega^{3\cdot 0} = a + b + c + d = 1 \\
P(1) &= a \cdot \omega^{0\cdot 1} + b \cdot \omega^{1\cdot 1} + c \cdot \omega^{2\cdot 1} + d \cdot \omega^{3\cdot 1} = a - bi - c + di = 0 \\
P(2) &= a \cdot \omega^{0\cdot 2} + b \cdot \omega^{1\cdot 2} + c \cdot \omega^{2\cdot 2} + d \cdot \omega^{3\cdot 2} = a - b + c - d = 1 \\
P(3) &= a \cdot \omega^{0\cdot 3} + b \cdot \omega^{1\cdot 3} + c \cdot \omega^{2\cdot 3} + d \cdot \omega^{3\cdot 3} = a + bi - c - di = -1
\end{aligned}$$

We obtain $(a, b, c, d) = (\frac{1}{4}, \frac{1}{4}i, \frac{3}{4}, -\frac{1}{4}i)$.

Problem 7.

Practice with polynomial multiplication by FFT.

(a) Suppose that you want to multiply the two polynomials $x + 1$ and $x^2 + 1$ using the FFT. Choose an appropriate power of two, find the FFT of the two sequences, multiply the results componentwise, and compute the inverse FFT to get the final result.

(b) Repeat for the pair of polynomials $1 + x + 2x^2$ and $2 + 3x$.

(a) Denote $A(k)$ and $B(k)$ as as the discrete Fourier coefficients of the polynomials $x + 1$ and $x^2 + 1$. The coefficients for the both original polynomials are $(1, 1, 0)$ and $(1, 0, 1)$, respectively. As 4 is the nearest power of two, we pad the coefficients so they become $(1, 1, 0, 0)$ and $(1, 0, 1, 0)$.

- For $(1, 1, 0, 0)$,

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

  The FFT of $(1, 1, 0, 0)$ is $(2, 1 - i, 0, 1 + i)$.

- For $(1, 0, 1, 0)$,

$$\begin{aligned} B(0) &= 1 \cdot \omega^{0 \cdot 0} + 0 \cdot \omega^{1 \cdot 0} + 1 \cdot \omega^{2 \cdot 0} + 0 \cdot \omega^{3 \cdot 0} = 2 \\ B(1) &= 1 \cdot \omega^{0 \cdot 1} + 0 \cdot \omega^{1 \cdot 1} + 1 \cdot \omega^{2 \cdot 1} + 0 \cdot \omega^{3 \cdot 1} = 0 \\ B(2) &= 1 \cdot \omega^{0 \cdot 2} + 0 \cdot \omega^{1 \cdot 2} + 1 \cdot \omega^{2 \cdot 2} + 0 \cdot \omega^{3 \cdot 2} = 2 \\ B(3) &= 1 \cdot \omega^{0 \cdot 3} + 0 \cdot \omega^{1 \cdot 3} + 1 \cdot \omega^{2 \cdot 3} + 0 \cdot \omega^{3 \cdot 3} = 0 \end{aligned}$$

  The FFT of $(1, 0, 1, 0)$ is $(2, 0, 2, 0)$.
  Then, the multiplied polynomials has $C(k)$ as the discrete Fourier coefficients, where:

$$\begin{aligned} C(0) &= A(0)B(0) = 4 \\ C(1) &= A(1)B(1) = 0 \\ C(2) &= A(2)B(2) = 0 \\ C(3) &= A(3)B(3) = 0 \end{aligned}$$

  Let the FFT of (a, b, c, d) to be $(4, 0, 0, 0)$, then

$$\begin{aligned} C(0) &= a \cdot \omega^{0 \cdot 0} + b \cdot \omega^{1 \cdot 0} + c \cdot \omega^{2 \cdot 0} + d \cdot \omega^{3 \cdot 0} = a + b + c + d = 4 \\ C(1) &= a \cdot \omega^{0 \cdot 1} + b \cdot \omega^{1 \cdot 1} + c \cdot \omega^{2 \cdot 1} + d \cdot \omega^{3 \cdot 1} = a - bi - c + di = 0 \\ C(2) &= a \cdot \omega^{0 \cdot 2} + b \cdot \omega^{1 \cdot 2} + c \cdot \omega^{2 \cdot 2} + d \cdot \omega^{3 \cdot 2} = a - b + c - d = 0 \\ C(3) &= a \cdot \omega^{0 \cdot 3} + b \cdot \omega^{1 \cdot 3} + c \cdot \omega^{2 \cdot 3} + d \cdot \omega^{3 \cdot 3} = a + bi - c - di = 0 \end{aligned}$$

  We obtain $(a, b, c, d) = (1, 1, 1, 1)$.
  This previous implies that the result of the multiplication of $x + 1$ and $x^2 + 1$ is $1 + x + x^2 + x^3$.

(b) Denote $A(k)$ and $B(k)$ as the discrete Fourier coefficients of the polynomials $1 + x + 2x^2$ and $2 + 3x$. The coefficients for the original polynomials are $(1, 1, 2)$ and $(2, 3)$, respectively. As 4 is the nearest power of two, we pad the coefficients to become $(1, 1, 2, 0)$ and $(2, 3, 0, 0)$.

- For $(1, 1, 2, 0)$,

$$\begin{aligned}
A(0) &= 1 \cdot \omega^{0 \cdot 0} + 1 \cdot \omega^{1 \cdot 0} + 2 \cdot \omega^{2 \cdot 0} + 0 \cdot \omega^{3 \cdot 0} = 4 \\
A(1) &= 1 \cdot \omega^{0 \cdot 1} + 1 \cdot \omega^{1 \cdot 1} + 2 \cdot \omega^{2 \cdot 1} + 0 \cdot \omega^{3 \cdot 1} = -1 - i \\
A(2) &= 1 \cdot \omega^{0 \cdot 2} + 1 \cdot \omega^{1 \cdot 2} + 2 \cdot \omega^{2 \cdot 2} + 0 \cdot \omega^{3 \cdot 2} = 2 \\
A(3) &= 1 \cdot \omega^{0 \cdot 3} + 1 \cdot \omega^{1 \cdot 3} + 2 \cdot \omega^{2 \cdot 3} + 0 \cdot \omega^{3 \cdot 3} = -1 + i
\end{aligned}$$

  The FFT of $(1, 1, 2, 0)$ is $(4, -1 - i, 2, -1 + i)$.

- For $(2, 3, 0, 0)$,

$$\begin{aligned}
B(0) &= 2 \cdot \omega^{0 \cdot 0} + 3 \cdot \omega^{1 \cdot 0} + 0 \cdot \omega^{2 \cdot 0} + 0 \cdot \omega^{3 \cdot 0} = 5 \\
B(1) &= 2 \cdot \omega^{0 \cdot 1} + 3 \cdot \omega^{1 \cdot 1} + 0 \cdot \omega^{2 \cdot 1} + 0 \cdot \omega^{3 \cdot 1} = 2 - 3i \\
B(2) &= 2 \cdot \omega^{0 \cdot 2} + 3 \cdot \omega^{1 \cdot 2} + 0 \cdot \omega^{2 \cdot 2} + 0 \cdot \omega^{3 \cdot 2} = -1 \\
B(3) &= 2 \cdot \omega^{0 \cdot 3} + 3 \cdot \omega^{1 \cdot 3} + 0 \cdot \omega^{2 \cdot 3} + 0 \cdot \omega^{3 \cdot 3} = 2 + 3i
\end{aligned}$$

  The FFT of $(2, 3, 0, 0)$ is $(5, 2 - 3i, -1, 2 + 3i)$.
  Then, the multiplied polynomials have $C(k)$ as the discrete Fourier coefficients, where:

$$\begin{aligned}
C(0) &= A(0)B(0) = 20 \\
C(1) &= A(1)B(1) = -5 + i \\
C(2) &= A(2)B(2) = -2 \\
C(3) &= A(3)B(3) = -5 - i
\end{aligned}$$

  Let the FFT of (a, b, c, d) to be $(20, -5 + i, -2, -5 - i)$, then

$$\begin{aligned}
C(0) &= a \cdot \omega^{0 \cdot 0} + b \cdot \omega^{1 \cdot 0} + c \cdot \omega^{2 \cdot 0} + d \cdot \omega^{3 \cdot 0} = a + b + c + d = 20 \\
C(1) &= a \cdot \omega^{0 \cdot 1} + b \cdot \omega^{1 \cdot 1} + c \cdot \omega^{2 \cdot 1} + d \cdot \omega^{3 \cdot 1} = a - bi - c + di = -5 + i \\
C(2) &= a \cdot \omega^{0 \cdot 2} + b \cdot \omega^{1 \cdot 2} + c \cdot \omega^{2 \cdot 2} + d \cdot \omega^{3 \cdot 2} = a - b + c - d = -2 \\
C(3) &= a \cdot \omega^{0 \cdot 3} + b \cdot \omega^{1 \cdot 3} + c \cdot \omega^{2 \cdot 3} + d \cdot \omega^{3 \cdot 3} = a + bi - c - di = -5 - i
\end{aligned}$$

  We obtain $(a, b, c, d) = (2, 5, 7, 6)$.
  This previous implies that the result of the multiplication of $2x^2 + x + 1$ and $3x + 2$ is $2 + 5x + 7x^2 + 6x^3$.