COMP3121 Assignment1

Fiona Lin z5131048

March 30, 2019

1. [20 marks] You are given two polynomials,

$$P_A(x) = A_0 + A_3x^3 + A_6x^6$$
and
$$P_B(x) = B_0 + B_3x^3 + B_6x^6 + B_9x^9$$

where all A_i's and B_j's are large numbers. Multiply these two polynomials using only 6 large number Multiplicaitons.

Solution Let $P_C(y) = P_A(y) \cdot P_B(y)$ and $y = x^3$, then we have
$P_A(y) = A_0 + A_3y + A_6y^2 \quad and \quad P_B(y) = B_0 + B_3y + B_6y^2 + B_9y^3$
Since the product polynomial $P_C(y) = P_A(y) \cdot P_B(y)$ is of degree 5, we need 6 value to uniquely determine $P_C(y)$. Let $y = -2, -1, 0, 1, 2, 3$; we have

$$P_A(-2) = A_0 + (-2)A_3 + (-2)^2A_6 = A_0 - 2A_3 + 4A_6$$
$$P_B(-2) = B_0 + (-2)B_3 + (-2)^2B_6 + (-2)^3B_9 = B_0 - 2B_3 + 4B_6 - 8B_9$$
$$P_A(-1) = A_0 + (-1)A_3 + (-1)^2A_6 = A_0 - 1A_3 + A_6$$
$$P_B(-1) = B_0 + (-1)B_3 + (-1)^2B_6 + (-1)^3B_9 = B_0 - 1B_3 + B_6 - B_9$$
$$P_A(0) = A_0 + (0)A_3 + (0)^2A_6 = A_0$$
$$P_B(0) = B_0 + (0)B_3 + (0)^2B_6 + (0)^3B_9 = B_0$$
$$P_A(1) = A_0 + (1)A_3 + (1)^2A_6 = A_0 + A_3 + A_6$$
$$P_B(1) = B_0 + (1)B_3 + (1)^2B_6 + (1)^3B_9 = B_0 + B_3 + B_6 + B_9$$
$$P_A(2) = A_0 + (2)A_3 + (2)^2A_6 = A_0 + 2A_3 + 4A_6$$
$$P_B(2) = B_0 + (2)B_3 + (2)^2B_6 + (2)^3B_9 = B_0 + 2B_3 + 4B_6 + 8B_9$$
$$P_A(3) = A_0 + (3)A_3 + (3)^2A_6 = A_0 + 3A_3 + 9A_6$$
$$P_B(3) = B_0 + (3)B_3 + (3)^2B_6 + (3)^3B_9 = B_0 + 3B_3 + 9B_6 + 27B_9$$

Thus, if we present the product $P_C(y) = P_A(y)P_B(y)$ in the coefficient form as
$P_C(y) = C_0 + C_1 y + C_2 y^2 + C_2 y^2 + C_3 y^3 + C_4 y^4 + C_5 y^5$
We get

$$\begin{aligned}
C_0 - 2C_1 + 4C_2 - 8C_3 + 16C_4 - 32C_5 + 64C_6 &= P_C(-2) = P_A(-2)P_B(-2) \\
C_0 - C_1 + C_2 - C_3 + C_4 - C_5 + C_6 &= P_C(-1) = P_A(-1)P_B(-1) \\
C_0 &= P_C(0) = P_A(0)P_B(0) \\
C_0 + C_1 + C_2 + C_3 + C_4 + C_5 + C_6 &= P_C(1) = P_A(1)P_B(1) \\
C_0 + 2C_1 + 4C_2 + 8C_3 + 16C_4 + 32C_5 + 64C_6 &= P_C(2) = P_A(2)P_B(2) \\
C_0 + 3C_1 + 9C_2 + 27C_3 + 81C_4 + 243C_5 + 729C_6 &= P_C(3) = P_A(3)P_B(3)
\end{aligned}$$

Solving this system of linear equations for $C_0, C_1, C_2, C_3, C_4, C_5$ we obtain

$$\begin{aligned}
C_0 &= P_C(0) \\
C_1 &= \frac{60P_C(3) - 15P_C(2) + 2P_C(1) - 20P_C(0) - 30P_C(-1) + 3P_C(-2)}{60} \\
C_2 &= -\frac{-16P_C(3) + P_C(2) + 30P_C(0) - 16P_C(-1) + P_C(-2)}{24} \\
C_3 &= -\frac{14P_C(3) - 7P_C(2) + P_C(1) - 10P_C(0) + P_C(-1) + P_C(-2)}{24} \\
C_4 &= \frac{-4P_C(3) + P_C(2) + 6P_C(0) - 4P_C(-1) + P_C(-2)}{24} \\
C_5 &= -\frac{-10P_C(3) + 5P_C(2) - P_C(1) + 10P_C(0) - 5P_C(-1) + P_C(-2)}{120}
\end{aligned}$$

Multiply these two polynomials using only these 6 large number multiplicaitons $C_0, C_1, C_2, C_3, C_4, C_5$.

2.

(a) **[5 marks]** Multiply two complex numbers $(a + ib)$ and $(c + id)$ (where a, b, c, d are all real numbers) using only 3 real number multiplications.

(b) **[5 marks]** Find $(a + ib)^2$ using only two multiplications of real numbers.

(c) **[10 marks]** Find the product $(a + ib)^2(c + id)^2$ using only five real number multiplications.

Solution

(a) Let $Z_0 = a + ib,\ Z_1 = c + id$, and we have $(a + b)(c + d) = ac + bd + bc + ad$, then

$$(a + ib)(c + id) = ac - bd + i(ad + bc) = ac - bd + i((a + b)(c + d) - ac - bd)$$

Therefore, multiply two complex numbers $(a + ib)$ and $(c + id)$ using only 3 real number multiplications $ac, bd, (a + b)(c + d)$.

(b) As $z_0^2 = (a+ib)^2 = a^2 - b^2 + 2iab = (a+b)(a-b) + 2iab$ Therefore, using only two multiplications of real numbers,$(a+b)(a-b), ab$.

(c) From previously, we have z_0z_1 using 3 multiplicaitons and z_0^2 using 2 multiplicaitons. So we can use evaluate $z_0^2z_1^2 = (a+ib)^2(c+id)^2 = (z_0z_1)^2$ with z_0z_1 using 3 multiplicaitons and then apply the result z_0^2 with 2 multiplicaitons. Thus, we can compute $(a+ib)^2(c+id)^2$ using only 5 multiplicaitons.

3.

(a) **[2 marks]** *Revision*: Describe how to multiply two n-degree polynomials together in $O(n \log n)$ time, using the Fast Fourier Transform (FFT). You do not need to explain how FFT works you may treat it as a black box.

(b) In this part we will use the Fast Fourier Transform (FFT) algorithm described in class to multiply multiple polynomials together (not just two). Suppose you have K polynomials $P1$, ... ,PK so that

$$degree(P1) + ... + degree(PK) = S$$

(i) **[6 marks]** Show that you can find the product of these K polynomials in $O(KS \log S)$ time. Hint: How many points do you need to uniquely determine an S-degree polynomial?

(ii) **[12 marks]** Show that you can find the product of these K polynomials in $O(S \log S \log K)$ time. Hint: consider using divide-and-conquer; a tree which you used in the previous assignment might be helpful here as well. Also, remember that if x, y, z are all positive, then $\log(x+y) < \log(x+y+z)$

Solution

(a) Multiply two n-degree polynomials, $P(x)$ and $Q(x)$, actually convert the coefficients of those polynomials into 2 vectors and perform convolution on these two verctors. Naive convolution take $O(n^2)$ time complexity.
However, the result polynomial will have degrees at most 2n, therefore it's sufficient to uniquely determine by the product of those 2 polynomials at $2n+1$ distinct points. So, utilising Discrete Fourier Transform(aka. DFT), we can evaluate the result polynomial for such a convolution at all complex roots of unity of order $2n$. The DFT can be optimised using divide-and-conquer algorithm called the Fast Fourier Transform (FFT) algorithm:
$\mathbf{convolution}(a, b) = \sqrt{n}(\mathbf{IFFT}(\mathbf{FFT}(a) \cdot \mathbf{FFT}(b)))$
each FFT takes $O(n \log n)$ time and the product of 2 FFT is elementwise product(also know as Hadamard product) and it's also $O(nlogn)$, hence the overall time complexity is $O(n \log n)$ time.

(b) (i) let A be the product of these K S-degree polynomials, then we have $A(k) = P_1(x) \cdot P_2(x) \cdot ... \cdot P_k(x)$ for all $1 \leq k \leq K$
When $K = 1$, A(1) is the product of S-degree polynomial and zero-degree polynomial. Then to uniquely determine A(1), it will need the complex roots of of unity of order S to evaluate that polynomial. So $A(1)$ takes $O(S \log S)$
When $K = 2$, from previous, we know $A(2)$ takes $O(S \log S)$
When $K = n$, $A(n)$ is the product of n S-degree polynomial. Then to uniquely determine A(n), it will need the complex roots of of unity of order nS to evaluate that polynomial. So $A(n)$ takes $O(nS \log S)$
So $K = n + 1$, we have,

$$A(n+1) = A(n) \cdot P_{n+1}(x) \Rightarrow nS \log S + S \log S = (n+1) \log S$$

Hence, the product of these K S-degree polynomials will take $O(KS \log S)$.
With the base case $K = 1$ takes $O(S \log S)$, we can compute $A(k) = P_1(x) \cdot P_2(x) \cdot ... \cdot P_k(x)$ for all $1 \leq k \leq K$ recursively. At each recurrence, the degree of the partial product $A(n)$ and of polynomial $P_{n+1}(x)$ 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 $S \log S$. Computing K such multiplications needs the total time complexity is $O(KS \log S)$

(ii) From last part, we know the product of these K S-degree polynomials will take $O(KS \log S)$. And it does wiht K times $O(S \log S)$, which in K linearly operations.
Moreover, the product of K S-degree polynomials follows the associative law: $(P_{i-1}(x) * P_i(x)) * P_{i+1}(x) = P_{i-1}(x) * (P_i(x) * P_{i+1}(x))$.
So we can approach this using divide-and-conquer to achieve $O(S \log S \log K)$, we can split k polynomials into pairs and half K times $O(S \log S)$ recursively. Therefore, we can use $\log K$ times $O(S \log S)$ instead of K times $O(S \log S)$. Hence, you can find the product of these K S-degree polynomials in $O(S \log S \log K)$ time.

4. [20 marks, each pair 4 marks] 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) **[5 marks]** 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) **[15 marks]** Hence or otherwise, give an algorithm that finds Fn in $O(\log n)$ time.

Solution **(a)** When $n = 1$, we have

$$\begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1$$

Let $n = k$, we have

$$\begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k$$

And when $n = k + 1$,

$$\begin{aligned} LHS &= \begin{pmatrix} F_{k+2} & F_{k+1} \\ F_{k+1} & F_k \end{pmatrix} = \begin{pmatrix} F_{k+1} + F_k & F_k + F_{k-1} \\ F_k + F_{k-1} & F_k \end{pmatrix} \\ &= \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k+1} = RHS \end{aligned}$$

Hence for all integers $n \geq 1$, $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$

(b) As previous, we know F_n can obtain by computing $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$, then let $G = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$.
Thus, $F_n = G^n$ using divide-and-conquer algorithm, G^2 take $O(1)$
If n is even, we have $G^n = G^{2^k} \Rightarrow k = \log n$. So it takes $O(\log n)$
If n is odd, we have $G^n = G^{2^{k+1}} \Rightarrow k = \log n - 1$. So it still takes $O(\log n)$ Hence, it takes overall $O(\log n)$ to find F_n using divide-and-conquer algorithm.

5. Your army consists of a line of N giants, each with a certain height. You must designate precisely $L \leq N$ of them to be leaders. Leaders must be spaced out across the line; specifically, every pair of leaders must have at least $K \geq 0$ giants standing in between them. Given N, L, K and the heights $H[1..N]$ of the giants in the order that they stand in the line as input, find the *maximum* height of the *shortest* leader among all valid choices of L leaders. We call this the optimisation version of the problem.

For instance, suppose $N = 10, L = 3, K = 2$ and $H = [1, 10, 4, 2, 3, 7, 12, 8, 7, 2]$. Then among the 10 giants, you must choose 3 leaders so that each pair of leaders has at least 2 giants standing in between them. The best choice of leaders has heights 10, 7 and 7, with the shortest leader having height 7. This is the best possible for this case.

(a) **[8 marks]** In the *decision* version of this problem, we are given an additional integer T as input. Our task is to decide if there exists some valid choice of leaders satisfying the constraints whose shortest leader has height no less than T .

(b) **[12 marks]** Hence, show that you can solve the optimisation version of this problem in $O(N \log N)$ time.

Solution

(a) In this task, we needs to works out if there are enough leaders candidate meeting the minimum required height T. Therefore, it needs to check if there are L leaders meeting the minimum required height T, and L leaders are seperating by at least every K griants in N giants.
Iterate all N giants, find the first eligible leader candidate and from then on, from the last eligible leader candidate skipping K giants to find the next eligible leader candidate. This takes $O(N)$ time complexity. After finding out all the eligible candidates, check if the total number of the eligible candidates is at least K. Return **true** if the total number is greater or equal to K and return **false** otherwise. This take $O(1)$ time so the overall time complexity is $O(N)$

(b) The optimisation version of the problem is to finding the largest value of T to the decision problem is **true** with the given N, L, K.
With the given N, L, K, our decision algorithm return **true** from the shortest height T_l to the tallest height T_h of L leaders. Hence, the decision problem will remaind true in T, where $T \in [T_l, T_h]$ ($[T_l, T_h]$ is in the range of all valid leader candidates' heights).
Therefore, assuming all N griants are valid candidates, merge sort the heights of N griants in $O(n \log n)$, we have $T \in [T_l,\ T_h]$. As we already know that the decision problem will remaind true in T, where $T \in [T_l,\ T_h]$. So from the largest T_l, use binary search to work out the maximum value of T where the decision problem returns **true**. The binary search will select the direction to go over $T \in [T_l,\ T_h]$ according to the result of the decision problem. Since there are $O(N \log N)$ iterations in the binary search of T the range of N griants' heights, each decision taking $O(N)$. Hence, the overall complexity of the algorithm is $O(N \log N)$.