Assignment 2

z5142340 Haiyu LYU

Question1:

Let $y = x^3$, we have that:

$P_A(y) = A_0 + A_3 y + A_6 y^2$

$P_B(y) = B_0 + B_3 y + B_6 y^2 + B_9 y^3$

Because the degree of $P_A(y)$ is 2 and the degree of $P_B(y)$ is 3. So, we can multiply two polynomials using only 6 multiplications.

Let c_i be the coefficient of the polynomial, then we have that:

$P_c(y) = P_A(y) \cdot P_B(y)$

$= c_0 + c_1 y^1 + c_2 y^2 + c_3 y^3 + c_4 y^4 + c_5 y^5$

Question2:

(a).

$(a + ib)(c + id) = ac - bd + i(cb + ad)$

Because $(a + b)(c + d) = ac + bd + bc + ad$, ac and bd make up the real part (2 multiplications). $(cb + ad) = (a + b)(c + d) - ac - bd$. So imaginary part needs 1 multiplication. Thus, we can use only 3 real number multiplications.

(b).

$(a + ib)^2 = a^2 - b^2 + i(2ab) = (a + b)(a - b) + i(2ab)$

Because $(a + b)(a - b)$ is the real part of $(a + ib)^2$ and it just uses 1 multiplication. For imaginary part, since $2ab = ab + ab$, we can only use 1 multiplication to find ab and then we can get $2ab$ by addition. Thus, we compute the product using 2 multiplications.

(c).$(a + ib)^2(c + id)^2 = [(a + ib)(c + id)]^2$

We can evaluate $(a + ib)(c + id)$ using 3 multiplications using part **(a)**. Then square the result using 2 multiplications using part **(b)**. Therefore, 5 real number multiplications are needed.

Question3:

(a).

Assume that we have 2 polynomials called P_A and P_B. The product of P_A and P_B has degree at $2\mathrm{n}$. In order to determine it, we need $2\mathrm{n}+1$ values at point. We can use FFT to evaluate P_A and P_B at $2\mathrm{n}+1$ points. Then we multiply the results of evaluation pointwise. So we got $2\mathrm{n}+1$ equations. Then we need to use inverse FFT to retrieve $P_A P_B$ in coefficient form. Therefore, run this algorithm in $O(\log n)$.

(b).

(i).

Let $C(i) = P_1(x) \cdot P_2(x) \cdots\cdots P_i(x)(1 \le i \le K)$

$C(i+1) = C(i) \cdot P_{i+1}(x)$

It looks like simple recursion (base case is $C(1) = P_1(x)$).

Because $degree(P_1) + \cdots\cdots + degree(P_k) = \mathrm{S}$, the degree of $C(i)$ and $P_{i+1}(x)$ is always less than S. For each multiplication, the time complexity may be $O(S \log S)$ using FFT. We have k multiplications, so the total time is $O(KS \log S)$.

(ii).

Using divide-and-conquer can reduce the time complexity.

At the first, we can make 2 polynomials as a pair, then we need calculate the product of the pair in $O(S \log S)$. For example, we can get the product

$C_{P_1P_2}, C_{P_3P_4} \cdots\cdots C_{P_{K-1}P_K}$.

For each $C_{P_iP_{i+1}}$, we can still set two adjacent C as a pair. Then we can also get the product of them.

Repeat the above algorithm until we finally get one result. This looks like a tree and the height of tree is $\log K$.

Therefore, we think $O(S \log S \log K)$ is needed for this algorithm.

Question4:

(a).

To proof $T(n) = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$, we use mathematical induction:

$T(n+1) = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_{n+1}+F_n & F_{n+1} \\ F_n+F_{n-1} & F_n \end{pmatrix}$

Because $F_{n+1} + F_n = F_{n+2}$ and $F_n + F_{n-1} = F_{n+1}$.

So $T(n+1) = \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}$

Thus, by mathematical induction, for all integer n>1,

$$T(n) = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

(b).

Let M is matrix of the Fibonacci numbers.

$$M^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

$$M^{\frac{n}{2}} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{\frac{n}{2}}$$

……

$$M^1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1$$

$$M^n = M^{\frac{n}{2}} \cdot M^{\frac{n}{2}} = \cdots = (M^1)^n$$

We can clearly see that M^n seems like a tree and the tree height is $\log_2 n$. Because we calculate the multiplication of two matrix in any level in $O(1)$. So the time complexity of M^n is $O(\log n)$.

Question5:

(a).

To find those choice of leaders, we only consider the case that the height of giant is less than or equal to T. Firstly we loop through the array H and find the leader that satisfies our condition (height <=T). Record this leader and jump to the next K. If the conditions are still met, then continue to record this leader. Repeat this calculation until you have traversed the entire array. We compare whether the number of leaders recorded is equal to L. If it equal to L, the leader who satisfies the condition exists. The complexity of this algorithm is $O(N)$.

(b).

We must find the ideal T, which is the maximum value of T. We can think of T as monotonic, because if there is a larger T that satisfies the condition, then a smaller T must also satisfy the condition. So we sort the possible T as a list. Then our strategy is to use the binary search to determine the maximum value of T. We can find a suitable T by binary search $O(\log N)$, and then use $O(N)$ to verify that T meets the condition. The total complexity is $O(N \log N)$