COMP 9101

Assignment 2

Student ID : z5174741

Student name : GUANQUN ZHOU

Q1

$$P_A(x) = A_0 + A_3x^3 + A_6x^6$$

And

$$P_B(x) = B_0 + B_3x^3 + B_6x^6 + B_9x^9$$

We can let $y = x^3$ and in this way we can get another two polynomials:

$$P_A(y) = A_0 + A_3y + A_6y^2$$

And

$$P_B(y) = B_0 + B_3y + B_6y^2 + B_9y^3$$

We let $P_c(y) = P_A(y)P_B(y)$

The degree of $P_A(y)$ is 2 and the degree of $P_B(y)$ is 3.

So the degree of $P_c(y)$ should be $2 + 3 = 5$ and we can use 6 coefficients to represent $P_c(y)$

In other word, $P_c(y) = C_0 + C_1y + C_2y^2 + C_3y^3 + C_4y^4 + C_5y^5 + C_6y^6$

In this way, we can multiply those two polynomials using only 6 large number multiplications.

We can use 6 small numbers, for example, -3,-2,-1,0,1,2 , to evaluate all coefficients C_i of $P_c(y)$.

If we just calculate the product of $P_A(y)P_B(y)$:

$$P_A(y)P_B(y) = A_0B_0 + (A_0B_3 + A_3B_0)y + (A_0B_6 + A_3B_3 + A_6B_0)y^2 + (A_0B_9 + A_3B_6 + A_6B_3)y^3 + (A_3B_9 + A_6B_6)y^4 + A_6B_9y^5$$

We can also get the values of $< C_0, C_1, C_2, C_3, C_4, C_5, C_6 >$ and multiply those two polynomials using only 6 large number multiplications.

Q2

(a) We can multiply $(a + ib)\ and\ (c + id)$ and we can get:

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

We can find that $(ad + bc) = (a + b)(c + d) - ac - bd$, which means if we already know the value of products of $ac\ and\ bd$, then we can get the value of $(ad + bc)$ with only one real number multiplications.

So the three real number multiplications we need to multiply $(a + ib)\ and\ (c + id)$ are $ac, (a + b)(c + d)\ and\ bd$.

(b) We can calculate $(a + ib)^2$ and we can get:

$$(a + ib)^2 = a^2 + 2abi - b^2$$

And we can find that $a^2 - b^2 = (a + b)(a - b)$

The product of $(a + ib)^2$ could be equal to $(a + b)(a - b) + 2abi$

So the two multiplications of real numbers we need to calculate $(a+ib)^2$ are $(a+b)(a-b)\ and\ ab$.

(c) We can let $\mathrm{R} = (a+ib)^2(c+id)^2$

We can find that $\mathrm{R} = (a+ib)^2(c+id)^2 = [(a+ib)(c+id)]^2$.

According to conclusion of (a), we can multiply $(a+ib)\ and\ (c+id)$ with only three real number multiplications. After calculation, we can get a new complex number, suppose we let the new complex number is $(e+if)$.

So now $\mathrm{R} = (e+if)^2$ and our job is to calculate the result of $(e+if)^2$.

According to conclusion of (b), we can calculate $(e+if)^2$ using only five real number multiplications.

So in total, we can find the product $(a+ib)^2(c+id)^2$ using only five real number multiplications.

Q3

(a) Assume two n-degree polynomials are:

$$P_A(x) = A_0 + A_1x + A_2x^2 + \cdots + A_nx^n$$

And

$$P_B(x) = B_0 + B_1x + B_2x^2 + \cdots + B_nx^n$$

We let $P_c(x) = P_A(x)P_B(x)$.

The degree of both $P_A(x)$ and $P_B(x)$ is n, so the degree of $P_c(x)$ is $2n$. Thus we will be able to use $2n+1$ distinct coefficients to uniquely determine $P_c(x)$.

First we calculate DFT for every polynomials using the roots of unity.

Since we need $2n+1$ distinct coefficients to uniquely determine $P_c(x)$, the DFT of $P_A(x)$ and $P_B(x)$ should be like :

$$\{P_A(1), P_A(\omega_{2n+1}^1), P_A(\omega_{2n+1}^2), P_A(\omega_{2n+1}^3), P_A(\omega_{2n+1}^4), \dots, P_A(\omega_{2n+1}^{2n})\}$$

And

$$\{P_B(1), P_B(\omega_{2n+1}^1), P_B(\omega_{2n+1}^2), P_B(\omega_{2n+1}^3), P_B(\omega_{2n+1}^4), \dots, P_B(\omega_{2n+1}^{2n})\}$$

We use FFT to evaluate these DFT, and this can be done in $O(nlogn)$.

The second step is to evaluate the multiplication of

$\{P_A(1), P_A(\omega_{2n+1}^1), P_A(\omega_{2n+1}^2), P_A(\omega_{2n+1}^3), P_A(\omega_{2n+1}^4), \dots, P_A(\omega_{2n+1}^{2n})\}$

and

$\{P_B(1), P_B(\omega_{2n+1}^1), P_B(\omega_{2n+1}^2), P_B(\omega_{2n+1}^3), P_B(\omega_{2n+1}^4), \dots, P_B(\omega_{2n+1}^{2n})\}$

The product is

$$\{P_A(1)P_B(1), P_A(\omega_{2n+1}^1)P_B(\omega_{2n+1}^1), P_A(\omega_{2n+1}^2), P_B(\omega_{2n+1}^2), \dots, P_A(\omega_{2n+1}^{2n})P_B(\omega_{2n+1}^{2n})\}$$

And this can be done in $O(n)$.

Third step is to evaluate $P_c(x)$ with IDFT, and this can be done in $O(nlogn)$.

Thus, the product of $P_A(x)$ and $P_B(x)$ can be compute in $O(nlogn)$ with the Fast Fourier Transform (FFT)

(b)

(i) We are given K polynomials $P_1, P_2, \dots, P_{k-1}, P_k$ and deg(P_1)+⋯+deg(P_k) = S

In order to find the product of these K polynomials, we can compute them in this way:

$$\big((P_1 * P_2)P_3\big)P_4 \dots)P_k)$$

We recursively multiply every P_i with the product of previous calculation and in every time we multiply P_i with the product of previous calculation, since degree of P_i and degree of the product of previous calculation are both less than S, so we can say that every multiplication can be done in $O(SlogS)$ using FFT.

As we need to compute K polynomials, which means we need to do $(K-1)$ multiplications to get the product of these K polynomials. $(K-1)$ multiplications can be done in $O((K-1)SlogS)$ and $O\big((K-1)SlogS\big) < O(KSlogS)$.

Thus, we can say that the product of these K polynomials can be found in $O(KSlogS)$

(ii) We are given K polynomials $P_1, P_2, \dots, P_{k-1}, P_k$ and deg(P_1)+⋯+deg(P_k) = S

We can put every polynomials and their relative products in a complete binary tree.

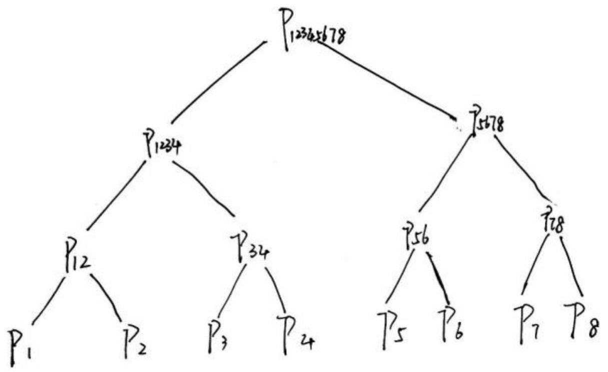
First step : we need to make sure all polynomials are assigned to one leaf. We need to compute the value of $\lfloor logk \rfloor$, we let $h = \lfloor logk \rfloor$. If $2^h = k$, then we can build a complete binary tree directly . If $2^h < k$, then we need to add two children to the left most $k - 2^h$ leaves. In this way, we can build a complete binary tree in any condition.

Second step: If $2^h < k$, we can treat the two lowest levels as one level, because in this way, the amount of the degrees of polynomials on each level of the binary tree is always equal to S. If $2^h = k$, the amount of the degrees of polynomials on each level of the binary tree is always equal to S.

We let $R_1\ and\ R_2$ to represent the degree of any two children polynomials of a node at some level. The degree of product of these two polynomials is $(R_1 + R_2)$. So we can compute the two polynomials in $O((R_1 + R_2)\log(R_1 + R_2))$.

Since $(R_1 + R_2) < S$, so $\log(R_1 + R_2) < \log(S)$, thus we can say these two children polynomials can be calculated in $O((R_1 + R_2)\log S)$. In this way, the amount tine of computing all pairs of polynomials on this level could be $O(S\log S)$ because the amount of the degrees of polynomials on each level of the binary tree is always equal to S.

Third step: The height of the tree would be $\lfloor logk \rfloor$ or $\lfloor logk \rfloor + 1$. According to second step, the calculation for every level can be done in $O(S\log S)$. And no matter height of the tree would be $\lfloor logk \rfloor$ or $\lfloor logk \rfloor + 1$, the whole calculation can be done in $O(S\log SlogK)$.



The graph above can show an example.
Suppose we have 8 polynomials and the amount of the degrees of polynomials is equal to S.
According to the method mentioned in (ii), we can compute every level in $O(S\log S)$ and the height of the tree is 3 which is $logK$. So this algorithm can help us to solve this problem in $O(S\log SlogK)$.

Q4

(a) Step1. we compute when $n = 1$:

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1$$

Step2. we assume the equation is true for n = k ($k \geq 1$),

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

Step3. We will prove n = k+1 can satisfy the equation:

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

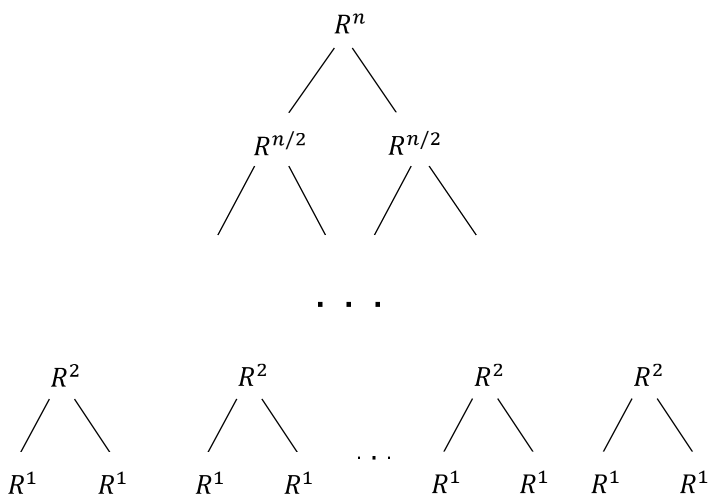
$$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k+1}$$

In this way, given formula can be proved for all integers $n \geq 1$.

(b) According to (a), $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$, then we find F_n via computing $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$

We can let $R = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$.

Then we can proceed with divide and conquer.



This graph shows how to compute R^n with divide and conquer.

If n is even, we just recursively calculate $R^{n/2}$ and square it. Every time of recursive calculation can be done in $O(1)$.

If n is odd, we just recursively calculate $R^{(n-1)/2}$,then square it and time it with a R. Every time of recursive calculation can also be done in $O(1)$.

We can see from the graph, the height of tree is $logn$, so we need to compute $logn$ times and every time calculation can be done in $O(1)$.

Thus, we can find F_n in $O(logn)$ time.

Q5

(a) First step: we need a variety C to record the number of elements which are no less than T. the initial value of C is zero.

Second step: We need to find the first element which is no less than T. we let the index of this element is R. Then we plus C with 1.

Third step: Then we need to check the element A whose index is $(R+K)$.

If the value of this element is no less than T, we plus C with 1 and update R with the index of this element A. Then we continue to skip K giants to check the new element A whose index is new $(R+K)$.

If the value of element A is less than T, then we need to check the next neighbour of element A. In other word, we will check the element whose index is $(R+1)$. If the value neighbour is less than T, then we continue to plus current index with 1 until we found next element whose value is no less than T. Then we plus C with 1 and update R with the index of this element whose value is no less than T. Then we continue to skip K giants to check the new element A whose index is new $(R+K)$.

We use method in third step to check over the entire list H. If $C \geq L$,return $True$, which means there exists some valid choice of leaders satisfying the constraints whose shortest leader has height no less than T. If $C < L$, return $False$, which means there is no any valid choice of leaders satisfying the constraints whose shortest leader has height no less than T.

In this way, the whole work can be done in $O(N)$.

(b) The optimisation version of this problem is to find the largest T which can return $True$ according to the decision version of this problem.

Since if one T can return $True$ in the decision version of this problem, which mean that all candidates whose value is less than T will all return $True$ in the decision version of this problem.

In other word, the result of the decision version of this problem is monotonously relative to T.

First step: we sort list H using Merge sort and this can be done in $O(NlogN)$.

Second step: we use binary search to find the optimal T.

We let initial value of S = 0(which is the index of the first element of list H) and initial value of L = the largest index of list H.

Then we recursively compute: $mid = \frac{(S+L)}{2}$, if for $H[mid]$ the decision version of this problem returns $True$, then $S = mid$. if for $H[mid]$ the decision version of this problem returns $False$, then $L = mid$.

In this way, we can find the optimal T.

Since binary search can be done in $O(logN)$ and for every $H[mid]$ the decision version of this problem need compute in $O(N)$.

Thus, the whole work can be done in $O(NlogN)$.