COMP9101

Assignment 2

Cong Cong

Z3414050

Question1:

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

And

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

Lets take $y = x^3$:

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

And

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

$$P_A(y) * P_B(y) = A_0B_0 + (A_0B_3 + A_3B_0)\, y + (A_0B_6 + A_3B_3 + A_6B_0)y^2 + (A_0B_9 + A_3B_6 + A_6B_3)y^3 + (A_3B_9 + A_6B_9)y^4 + A_6B_9y^5$$

We can use C_i to represent the constant terms:

$$P_A(y) * P_B(y) = C_0 + C_1y + C_2y^2 + C_3y^3 + C_4y^4 + C_5y^5$$

The degree of the product of $P_A(y)$ and $P_B(y)$ is $2 + 3 = 5$, which requires 6 large number multiplications. We can take the first 6 smallest integers to evaluate the equation.

Question2:

a) Lets take $n1 = (a + ib)$ and $n2 = (c + id)$

$$n1 * n2 = (a + ib) * (c + id) = ac + (ad + cb) * i - bd$$

And I know that $(a + b)(d + c) = ad + ac + db + db$

This means that if we know the product of ac and db, we are able to get the values of $(ad + cb)$, which is $(a + b)(d + c) - ac - db$

Thus, our three real number multiplications are:

1. ac
2. db
3. $(a + b)(d + c)$

b) Lets take $n1 = (a + ib)$

$$n1^2 = (a + ib)^2 = a^2 + 2abi - b^2$$

Here, I know that $a^2 - b^2 = (a + b)(a - b)$

Thus, we can find the result using two real number multiplications:

1. ab
2. $(a + b)(a - b)$

c) Lets take $r = (a + ib)^2(c + id)^2, n1 = (a + ib)^2\ and\ n2 = (c + id)^2$

1. From part b), I know that I can find the result of $(a + ib)^2$ using two multiplications, similarly, I can find the result of $(c + id)^2$ using two multiplications, then I need one final multiplication of n1 and n2 to get r, and in total, it only takes five multiplications.
2. Another way to think about this is $r = (a + ib)^2(c + id)^2 = [(a + ib)\,(c + id)]^2$, based on part a), $(a + ib)\,(c + id)$ can be calculate using three real number multiplications, suppose $r1 = (a + ib)\,(c + id) = x + yi$, then $r1^2 = (x + yi)^2$, moreover, from part b), $r1^2 = (x + yi)^2$ can be calculated using two real number multiplications, thus the total multiplications is 3+2 = 5.

Question3

a)

We have two n degree polynomials:

$$P_A(x) = A_0 + A_1x + \cdots + A_{n-1}x^{n-1}$$

And

$$P_B(x) = B_0 + B_1x + \cdots + B_{n-1}x^{n-1}$$

To find the product of two polynomials, first take the DFT of each every polynomials, and this step can be done in $O(nlogn)$ with the help of FFT, after this process, I end with:
$\{P_A(1), P_A(w_{2n-1}) \dots P_A(w_{2n-1}^{2n-2})\}$ and $\{P_B(1), P_B(w_{2n-1}) \dots P_B(w_{2n-1}^{2n-2})\}$

Here, the subscript of w is the number of terms of the product and It can be noticed that there is a mismatch between the number of terms of $P_A(x)$ and $\{P_A(1), P_A(w_{2n-1}) \dots P_A(w_{2n-1}^{2n-2})\}$, this can be solved by padding (n-1) zeros to $P_A(x)$.

Secondly, we do the multiplication between $\{P_A(1), P_A(w_{2n-1}) \dots P_A(w_{2n-1}^{2n-2})\}$ and $\{P_B(1), P_B(w_{2n-1}) \dots P_B(w_{2n-1}^{2n-2})\}$, this takes $O(n)$ and gives the product:

$$\{P_A(1)P_B(1), P_A(w_{2n-1})P_B(w_{2n-1}), \dots\ P_A(w_{2n-1}^{2n-2})P_B(w_{2n-1}^{2n-2})\}$$

Thirdly, in order to obtain the original coefficients $\sum_{i=0}^{j} A_i B_{j-1}$, we take the inverse FFT (IFFT), which takes $O(nlogn)$.

Thus, the product of $P_A(x)$ and $P_B(x)$ can be computed in time $O(nlogn)$ with the help of FFT.

b) Given K polynomials $P_1, P_2 \dots P_k$ and $degree(P_1) + degree(P_2) + \cdots + degree(P_k) = S$

i) Lets times each polynomials pairwise:

$$((P_1P_2) * P_3) * P_4 \dots$$

P_1P_2 can be obtained in

$$O((degree(P_1) + degree(P_2))\log(degree(P_1) + degree(P_2)$$

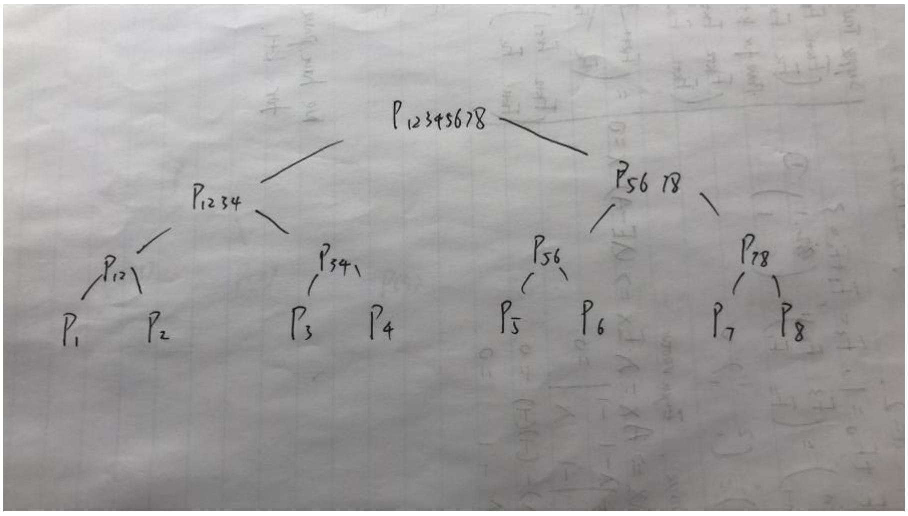
$(P_1P_2)P_3$ can be obtained in:

$O((degree(P_1) + degree(P_2) + degree(P_3))\log(degree(P_1) + degree(P_2) + degree(P_3)$

...

There will be K-1 such multiplications, and It is obvious that the time complexity of each multiplications is smaller than $O(SlogS)$, we can take $O(SlogS)$, and there are K-1 multiplications, this will result in $O((K-1)SlogS)$ and we also know that $O((K-1)SlogS) < O(KSlogS)$, thus, we can conclude it is possible to find the product of these K polynomials in $O(KSlogS)$.

ii) Assume that we have K = 8, we can imply divide and conquer, taking all polynomials as leaves of a binary tree and times two polynomials at a time:



Here, it is an example of K = 8, we have a binary tree of height = 3, and we can implement the same theory on K polynomials, same as in (i),on each level the multiplications is upper-bounded by than $O(SlogS)$, and the height of the tree is equals to $O(logK)$.

Thus, we do $O(SlogS)$ operations $O(logK)$ times and this results in $O(SlogSlogK)$.

Question4:

a) Show by induction:

Step 1: prove base case, we are given:

$$F_0 = 0, F_1 = 1 \; and \; F_n = F_{n-1} + F_{n-2} \; for \; all \; n \; \geq 2$$

Take n=2, $F_2 = F_1 + F_0 = 1$ and $F_3 = F_2 + F_1 = 2$, thus

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_3 & F_2 \\ F_2 & F_1 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$$

On the another side:

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

So, LHS = RHS and I have proven the base case is correct.

Step 2: Induction, suppose the given equation is true for k, which means:

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

Prove for k+1

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

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

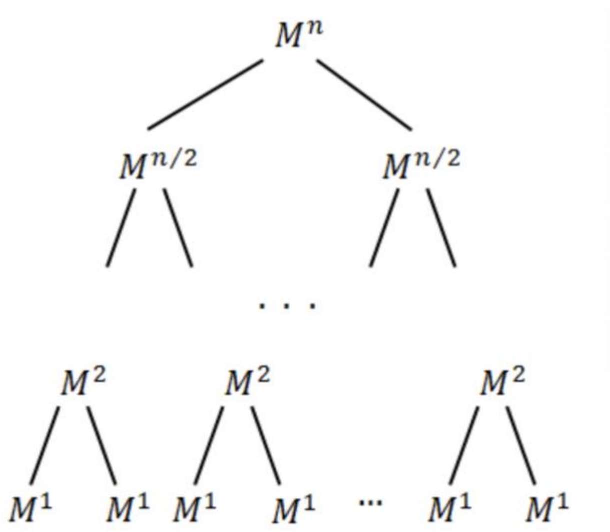
I have proven this formula is also true for k+1

b) To calculate F_n, we have:

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

Look at the right side of the equation, It can be seen that as long as I can figure out the result of the right hand of the equation, I can find the value of F_n,

To calculate $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ in $O(l\text{ogn})$ time, take, matrix $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ as M:



This graph shows that to obtain M^n, we need to calculate $M^{\frac{n}{2}}$ and times by itself, and to obtain $M^{\frac{n}{2}}$, we need to calculate $M^{\frac{n}{4}}$ and times by itself and so on.

This tree height is log_2n and on each level, the multiplications takes O(1), thus, in total, it ends up with a time complexity of $O(logn)$

Question5:

a) We are given $N, L, K, H\ with\ length\ of\ N, and\ T$,

To solve this problem in $O(N)$, It is reasonable to iterate over the given list H, and each time I check if the current element, which **with index i** ,is larger than or equal to T, if it is not the case, I continue checking the next element **(i+1)**. Otherwise, I jumps to index **(i+K)** and minus L by 1, then repeating this process until I got **L = 0** and I return true, or I reach the end of the list with **L >0** which ends up in the case of returning false.

b) With the help of a), the optimisation version of this problem can be solved in $O(NlogN)$. Firstly, sort list H with merge sort and this takes $O(NlogN)$, and then using the algorithm designed from a) and setting $T = H_{mid}$, where H_{mid} is the value in the middle of the sorted list H_sorted. If the algorithm returns **True**, this means that there might exist another value Larger than T which also satisfies the condition, so we take $[H_{mid} \dots H_{last}]$ and do the same check by taking $T = H'_{mid}$ of the new list $[H_{mid} \dots H_{last}]$. Whereas, if the algorithm returns **False**, this means values which is larger than H_{mid} will not satisfy the conditions. So we take $[H_{first} \dots H_{mid}]$ as our new list and do the same check by taking the middle value of it. This binary search approach takes a time complexity of $O(logn)$ and each check which used the method from a) takes $O(n)$, thus, in total, solving this problem used $O(nlogn)$ time.