Comp3121 Assignment2

Q1.

Let $y = x^3$, the $P_A(y) = A_0+A_3y+A_6y^2$, and $P_B(y) = B_0+B_3y+B_6y^2+B_9y^3$.
It is enough to be able to multiply any degree 3 polynomial by a degree 2 polynomial using 7 such multiplications: both P_A and P_B are really polynomials with those respective degrees in x^3.
The degree of $P_A+P_B = 3+2 = 6$. So, multiplications is 6+1 = 7. Since the result is a polynomial of degree 7, we can determining its values at 7 points and coefficients from these values by setting up a system of linear equations. Solving this is done by inverting a constant matrix , so this inversion can even be done by hand, offline and requires no computation. We then multiply the matrix by the vector formed by the pointwise multiplications, which again only multiplies these results by scalars, to give the final polynomial.

Q2.

a.

$$(a + i * b)(c + i * d) = a * c + a * i * d + i * b * c - d * b = a * c - d * b + (a + d) * i$$

So, there are three multiplication.

b.

$$(a+i*b)^2 = a^2 - b^2 + 2*a*b*I = (a+b)(a-b) + 2*a*b*i.$$

There are two multiplication.

c.

$$(a+i*b)^2(c+i*d)^2 = (a+i*b)(c+i*d)^2 = (a*c-d*b+(a+d)*i)^2 = (a*c-d*b)-(a+d))^2 + 2(a*c-d*b)*(a+d)*i$$

Thus, there are five multiplications.

Q3.

There are a convolution of two sequences. We can calculate the their product. P*Q(x) in O(nlog(n)) by using Fast Fourier Transform (FFT).

$P_A(x) = A_0 + A_1x + \ldots + A_nx^n$

$\Downarrow$ DFT O(n log n)

$\{P_A(1), P_A(\omega_{2n+1}), P_A(\omega_{2n+1}^2), \ldots, P_A(\omega_{2n+1}^{2n})\}$; ①

$P_B(x) = B_0 + B_1x + \ldots + B_nx^n$

$\Downarrow$ DFT O(n log n)

$\{P_B(1), P_B(\omega_{2n+1}), P_B(\omega_{2n+1}^2), \ldots, P_B(\omega_{2n+1}^{2n})\}$; ②

① and ②

$\Downarrow$ multiplication O(n)

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

$\Downarrow$ IDFT O(n log n)

$$P_C(x) = \sum_{j=0}^{2n}\left(\sum_{i=0}^{j} A_iB_{j-i}\right)x^j = \sum_{j=0}^{2n} c_j\,x^j = P_A(x)\cdot P_B(x)$$

The multiplication of two n-degree polynomials time complexity is O(nlogn)+O(n)+O(nlogn).

a.

I. We can suppose S+1 root of units. Then we have $Pi(x), \{i \in (1,2,3 \ldots k)\}$
$(\omega_{S+1}^0, P_i(\omega_{S+1}^0)), (\omega_{S+1}^1, P_i(\omega_{S+1}^1)), \ldots, (\omega_{S+1}^S, P_i(\omega_{S+1}^S))$
There are K polynomials $P_1, \ldots, P_K$. The degree is P(1)+P(2)+…P(k) = S. Each 2 Polynomials product is O(SlogS) by FFT. So K polynomials time complexity is O(KSlogS).

II. Step1. First, let the P1*P2,P3*P4….PK-1*PK. Then, the time complexity for this O((degree(p1)+degree(p2)log((degree(p1)+degree(p2))+…..)<SlogS. Since S is degree of P(1)+P(2)+…P(k).
Step2. Keeping product as a pair, we can image the product P as a tree. Each level of multiplication is less than SlogS. The height of tree is logK. Thus all the multiplication of time complexity is logK*SlogS.

Q4.

Use induction to prove $\begin{pmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ for (n>=2)

Step 1:

According to definition of Fibonacci sequence: F(n) = F(n-1) + F(n-2)

F(1) = 1, F(2) = 1, F(3) = 2 when n = 2

$$\begin{pmatrix} F(2+1) & F(2) \\ F(2) & F(2-1) \end{pmatrix} = \begin{pmatrix} F(2)+F(1) & F(2) \\ F(2) & F(1) \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2$$

Step 2:

Assume that when n=k (k>=2)

$$\begin{pmatrix} F(k+1) & F(k) \\ F(k) & F(k-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k}$$

Step 3:

When n=k+1 ((k+1)>=2)

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \bullet \begin{pmatrix} F(k+1) & F(k) \\ F(k) & F(k-1) \end{pmatrix} = \begin{pmatrix} F(k+2) & F(k+1) \\ F(k+1) & F(k) \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k+1}$$

(b)

As we have prove before,

$$\begin{pmatrix} F(n) & F(n-1) \\ F(n-1) & F(n-2) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}$$

Let $P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}$, thus we can obtain the F(n) by calculating matrix P, and F(n) is equal to the first element of the first line in P.

Now, the problem is simplified as how to calculate a matrix M^n, we can divide M^n into two parts each time, constructing a form of a tree.

$$M^{n}$$

$$M^{n/2} \qquad\qquad M^{n/2}$$

$$M^{n/4} \qquad M^{n/4} \qquad M^{n/4} \qquad M^{n/4}$$

$$\cdots \qquad \cdots \qquad \cdots \qquad \cdots$$

$$M^{2} \qquad M^{2} \qquad M^{2} \qquad M^{2}$$

$$M^{1}\ M^{1} \qquad M^{1}\ M^{1} \qquad M^{1}\ M^{1} \qquad M^{1}\ M^{1}$$

Height: log2n

If n is even, we need log2n time on multiplication.

If n is odd, we need calculate M^{n-1} and then multiply $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ once again, taking 1+ log2n time totally.

In summary, we could find F(n) in O(logn) time.

Q5.

(a).

Firstly, we can let the elements in H which is lower than T be 0:

```
For h in H:
    If h < T:
        h = 0
```

length of H = N, thus time complexity: O(N)

For example, the original sequence is H = [1,10,4,2,3,7,12,8,7,2], suppose T=5.

After being modified, H becomes H = [0, 10, 0, 0, 0, 7, 12, 8, 7, 0]

To decide if there exits some valid choice of leaders satisfying the constraints, we just estimate that whether the number of valid numbers is larger than L.

There are some pseudocode:

```
Leader(H):
   count = 0
   while i < N:
     if H[i] == 0:
         i += 1
     else: (H[i] >= T)
         count += 1
         i = i + k + 1 (K giants between leaders)
   if count >= L:
     return TRUE (exists required valid choice)
   else:
     return FALSE (does not exist)
```

Thus, the total time is O(N).

(b).

Sort the sequence H in non-increasing order of height using Merge-Sort in O(NlogN) time.

Choose the first some numbers that are not less than T.

Replace the number of sequence H with their original index, and the new sequence is named I.

Sort the index sequence in non-decreasing order of index.

For example:

T = 5

Original H:	[1,10,4,2,3,7,12,8,7,2]	
Sorted H:	[12,10,8,7,7,4,3,2,2,1]	O(NlogN)
Sorted H with numbers not less than T:	[12,10,8,7,7]	O(N)
Sorted H represented with index :	[7,2,8,6,9]	O(N)

Sort I in non-decreasing order of index [2,6,7,8,9] O(NlogN)

Next, use the same function as (a) has applied to determine the optimisation version, taking O(N) time.

Hence, we can solve the optimisation version of this problem in O(NlogN) time in total.