COMP3121 ASSIGNMNET 2

Jingjie Jin

Z5085901

Question 1:

(a).

Let the two n-degree polynomials be

$P_A(x) = A_0 + A_1x + A_2x^2 + \dots A_nx^n$, and

$P_B(x) = B_0 + B_1x + B_2x^2 + \dots B_nx^n$,respectively.

We will compute the multiplication of $P_A(x)$ and $P_B(x)$, which is represented as $P_C(x)$:

$P_C(x) = P_A(x) \cdot P_B(x) = C_0 + C_1x + C_2x^2 + \dots C_{2n}x^{2n}$

Firstly, we evaluate the complex roots of unity of order n+1 in both $P_A(x)$ and $P_B(x)$.

i.e. : we compute $P_A(W^l_{n+1})$ for all $0 <= l <= n$

The sequence of values ($P_A(1)$, $P_A(W^1_{n+1})$, ..., $P_A(W^n_{n+1})$) is called DFT (Discrete Fourier Transform) of the sequence $A = (A_0 , A_1 , A_2 , \dots, A_n)$

The DFT can be calculated by FFT which can be faster in O(nlogn) time .

Next, to compute $P_C(x)$ with degree 2n, we will evaluate them at the roots of unity of order 2n+1.

This produces the DFT of the sequences of A and B:

$A = (A_0 , A_1 , A_2 , \dots, A_n , 0, \dots..., 0)$

$B = (B_0 , B_1 , B_2 , \dots, B_n , 0, \dots..., 0)$

We will then multiply the corresponding values $P_A(W^l_{n+1})$ and $P_B(W^l_{n+1})$

Finally, we will use the inverse transformation for DFT (IDFT) to recover the coefficients of the product polynomials from its value at these roots of unity.

The general algorithm:

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

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

⇨ DFT O(nlogn) (using FFT)

$(P_A(1), P_A(W^1_{n+1}), \dots\dots, P_A(W^{2n}_{2n+1}))$

$(P_B(1), P_B(W^1_{n+1}), \dots\dots, P_B(W^{2n}_{2n+1}))$

⇨ Multiplication O(n)

$(P_A(1)\cdot P_B(1), P_A(W^1_{n+1})\cdot P_B(W^1_{n+1}), \dots\dots, P_A(W^{2n}_{2n+1})\cdot P_B(W^{2n}_{2n+1}))$

⇨ IDFT O(nlogn) (using FFT)

$$P_c(x) = \sum_{j=0}^{2n}(\sum_{i=0}^{j} A_i B_{j-i})x^j = \sum_{j=0}^{2n} C_j x^j = P_A(x)\cdot P_B(x)$$

Thus, the multiplication can be computed in O(nlogn) time.

(b).

(i).

Since degree(P1) + degree(P2) + ... + degree(Pk) = S

We can evaluate the result of multiple polynomials multiplication at roots of unity of order S+1.

Let the result of multiple multiplication be $P_r(x)$ with degree S+1, and then we can produce the DFT of the sequences of P1, P2, ..., Pk with the length of (S+1).

Thus, the stage of DFT will cost O(SlogS) time.

Next, we multiply the corresponding values $P_i(W^k_{n+1})$ and $P_j(W^k_{n+1})$ in the DFTs of K polynomials with the roots of unity of order S+1 (W^l_{S+1}) for l in [0, S+1].

In this stage, the time is O(KS).

Finally, we will use the inverse transformation for DFT (IDFT) to recover the coefficients of the product polynomials from its value at these roots of unity which will take O(SlogS) by using FFT.

Thus, the total time is O(KSlogS).

(ii).

We can reduce the time complexity in the stage of multiplication by using divide-and-conquer. This means that K polynomials can be multiplied pair by pair.

For example, we can multiply adjacent polynomials at first time to get K/2 outputs:

($P_1(1)\cdot P_2(1)$, $P_1(W^1_{S+1})\cdot P_2(W^1_{S+1})$, ..., $P_1(W^S_{S+1})\cdot P_2(W^S_{S+1})$)

($P_3(1)\cdot P_4(1)$, $P_3(W^1_{S+1})\cdot P_4(W^1_{S+1})$, ..., $P_3(W^S_{S+1})\cdot P_4(W^S_{S+1})$)

.....

($P_{k-3}(1)\cdot P_{k-2}(1)$, $P_{k-3}(W^1_{S+1})\cdot P_{k-2}(W^1_{S+1})$, ..., $P_{k-3}(W^S_{S+1})\cdot P_{k-2}(W^S_{S+1})$)

($P_{k-1}(1)\cdot P_k(1)$, $P_{k-1}(W^1_{S+1})\cdot P_k(W^1_{S+1})$, ..., $P_{k-1}(W^S_{S+1})\cdot P_k(W^S_{S+1})$)

And next time, obtain K/4 outputs,

($P_1(1)\cdot P_2(1)\cdot P_3(1)\cdot P_4(1)$, $P_1(W^1_{S+1})\cdot P_2(W^1_{S+1})\cdot P_3(W^1_{S+1})\cdot P_4(W^1_{S+1})$, ..., $P_1(W^S_{S+1})\cdot P_2(W^S_{S+1})$ $P_3(W^S_{S+1})\cdot P_4(W^S_{S+1})$)

.....

($P_{k-3}(1)\cdot P_{k-2}(1)\cdot P_{k-1}(1)\cdot P_k(1)$, $P_{k-3}(W^1_{S+1})\cdot P_{k-2}(W^1_{S+1})\cdot P_{k-1}(W^1_{S+1})\cdot P_k(W^1_{S+1})$, ..., $P_{k-3}(W^S_{S+1})\cdot P_{k-2}(W^S_{S+1})\cdot P_{k-1}(W^S_{S+1})\cdot P_k(W^S_{S+1})$)

until obtain only one result which includes K polynomials with products of the roots of unity of order S+1 (W^l_{S+1}) for l in [0, S+1].

($P_1(1)...P_k(1)$, $P_1(W^1_{S+1})...P_k(W^1_{S+1})$, ..., $P_1(W^S_{S+1})...P_k(W^S_{S+1})$)

Thus, the time of multiplication can reduce to O(logK).

Totally, the time complexity of finding the product of these K polynomials is in O(SlogSlogK) time.

Question 2:

This problem can be solved as a problem of combination in polynomial by squaring their sum.

As the given example, there are 3 coins (N=3) with the values 1, 4 and 5 (M=5). So we can calculate the result of $\mathbf{(x^1+x^4+x^5)^2}$.

$(x^1+x^4+x^5)^2$

$= (x^1+x^4+x^5)\ (x^1+x^4+x^5)$

$= x^2 + x^5 + x^6 + x^5 + x^8 + x^9 + x^6 + x^9 + x^{10}$

$= x^2 + 2x^5 + 2x^6 + x^8 + 2x^9 + x^{10}$

We choose the elements of the result whose coefficient is larger than 1. Coefficients larger than 1 represents the sums between two values, such as $2x^5$ (1,4 the sum is 5), $2x^6$(1,5 the sum is 6), $2x^9$(4,5 the sum is 9), there are 3 possible combinations in total.

Thus, we can deduct that the possible sums are relevant to the degree of the polynomial which means **the highest value(M)** in N coins. We just need to calculate the square of the polynomial and select the elements whose coefficient is larger than 1. The powers of the selected elements are the possible sums.

The time of squaring polynomial takes $O(M^2)$ time.

Obviously, we can use FFT to reduce the multiplication $O(M^2)$ to **O(MlogM).**

Question 3:

(a).

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)

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

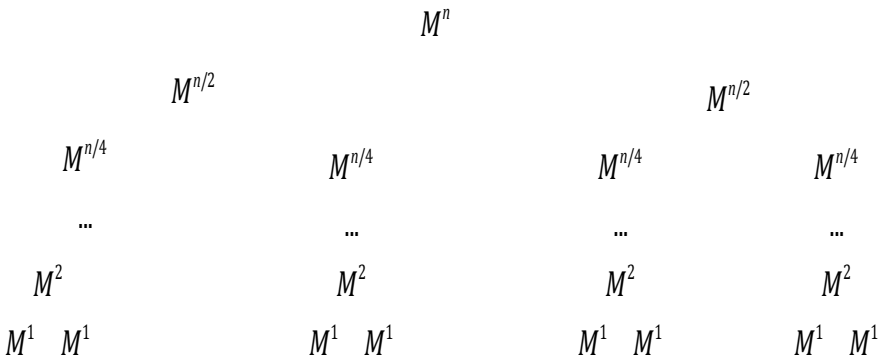
(b)

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 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.



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.

Question 4:

By comparing a[i] and b[i], we can create a new sequence c[1…N] including the larger offer of each item between a and b.

i.e. c = [a[1],a[2],a[3],b[4],b[5],a[6],b[7],a[8],a[9],b[10]] (just for example)

Then, sort the c in an non-increasing order of amount of items using Merge-Sort in O(NlongN) time.

aCount = count(a[i], c) for i in N:

the number of Alice offers larger than Bob

bCount = count(b[i], c) for i in N:

the number of Bob offers larger than Alice

As we know, aCount + bCount = N

Case 1: N=A+B

i. If aCount = A, then bCount = B:
   We sell the N items as its larger offer in c:
   a[i] to Alice, b[i] to Bob.
   The maximum total amount = sum of sequence c
ii. If aCount > A, then bCount < B:
   We sell the first A items of a[i] in sorted c to Alice, remaining aCount – A items sold to Bob with bCount items.
   Hence, we can make the maximum total amount.
iii. If aCount < A, then bCount > B:
   We sell the first B items of b[i] in sorted c to Bob, remaining bCount – B items sold to Alice with aCount items.

Case 2: N<A+B

i. If aCount < A and bCount < B:
   We sell the N items as its larger offer in c:
   a[i] to Alice, b[i] to Bob.
   The maximum total amount = sum of sequence c

ii. If aCount > A, then bCount < B,
   Because aCount + bCount = N < A + B
   => aCount – A < B – bCount
   Due to aCount – A > 0, there must be B-bCount > 0
   => bCount < B
   We sell the first A items of a[i] in sorted c to Alice, remaining aCount – A items sold to Bob with bCount items.
   (aCount – A + bCount = N – A < B, so the number of items Bob buying will not be more than B)

iii. If aCount < A, then bCount > B,
   Similar reason as Case 2(ii).
   We sell the first B items of b[i] in sorted c to Bob, remaining bCount – B items sold to Alice with aCount items.

Thus, determining the maximum total amount of money we can earn will take O(NlogN) time.

Question 5:

(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.

Function:

```
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)
```

(Time: O(N))

```
if count >= L:
    return TRUE (exists required valid choice)
else:
    return FALSE (does not exist)
```

(Time: O(1))

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.

END