COMP3821 Assignment2 (z5119666)

1.(a)

Suppose that we have 2 polynomials A = $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ and B = $b_0 + b_1x + b_2x^2 + \ldots + b_nx^n$. We can represent A and B by vector form such that $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n + 0x^{n+1} + \ldots + 0x^{2n}$, $B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_nx^n + 0x^{n+1} + \ldots + 0x^{2n}$. So we get a sequence of the coefficients of the polynomials a = $\langle a_0, a_1, a_2, \ldots, a_n, 0, 0, \ldots, 0\rangle$ (number of 0 = n) and b = $\langle b_0, b_1, b_2, \ldots, b_n, 0, 0, \ldots, 0\rangle$ (number of 0 = n). And use FFT to find the DFT of the two sequences A(x) and B(x) (point-value representation). Remember to pad them to the nearest power of 2 first. Then multiply the two DFTs to get C(x) = A(x)*B(x) (point-value representation) and use IFFT to find the coefficients of the polynomial C(x).

(b).(i)

For K polynomials, their coefficient form should be: for every i ($1 \le i \le K$), $Pi(x) = i_0 + i_1x + \ldots + i_sx^s$. So use FFT to find the DFT of each $P_i(x)$. Then multiple the K DFTs and use IFFT to find the coefficients of the final polynomial. Total cost O(KSlogS).

(ii)

Multiply two polynomials, suppose the degree are 'a' and 'b', then O((a+b)log(a+b)). So the first multiplications are: P_1*P_2, P_3*P_4, P_5*P_6, …, $P_{k-1}*P_k$. The complexity is $O((\deg(P_1) + \deg(P_2))\log(\deg(P_1) + \deg(P_2)) + (\deg(P_3) + \deg(P_4))\log(\deg(P_3) + \deg(P_4)) + \ldots) < O((\deg(P_1) + \deg(P_2) + \deg(P_3) + \ldots)\log S) = O(S\log S)$. And the total degree will not change. Do the recursive adjacent multiplications, and the complexity always O(SlogS). And logK recursions in total.

2.

If the value of coins are $v_1, v_2, \ldots, v_n$, then we can represent these values as polynomial $x^{v1} + x^{v2} + \ldots + x^{vn}$. We can multiply the polynomial by itself to get all possible sum of each pair in O(MlogM) by FFT. Since 'without replacement', we need to minus some values like $x^{2(vi)}$. Thus, $(x^{v1} + x^{v2} + \ldots + x^{vn})*(x^{v1} + x^{v2} + \ldots + x^{vn}) – (x^{2(v1)} + x^{2(v2)} + \ldots + x^{2(vn)})$

3. (a)

Assuming

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

Then

$$A_1 = \begin{pmatrix} F_2 & F_1 \\ F_1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1$$

By induction

$$A_{n+1} = A_n A_1$$
$$= \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} F_n + F_{n+1} & F_n + F_{n-1} \\ F_{n+1} & F_n \end{pmatrix}$$
$$= \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix}$$

(b).

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

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

……

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \times \ldots \times \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Consider it like a tree. And the height is logn

$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ calculation time: $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ is the same size in any power. Thus, we can perform the multiplication of 2 matrices in any power in O(1). Further, we shoud perform logn of such multiplications. Therefore, $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ complexity is O(logn)

4.

Calculate c[i] = a[i] – b[i] to help us know whether we should sell to Bob or Alice. And sort the absolute value of c[1..N] in descending order. Since we need to prioritise the items where there is the biggest price difference in order to maximise the profits. Time complexity is O(NlogN).

Finally, we need to sell the items base on the order of c until the quota of either A or B is met. Then the rest of items sold to the other. Meanwhile, store the sum of money we earned. Time complexity is O(N).

Total complexity is O(NlogN)

5.(a)

Algorithm

```
if    N < L + K(L-1)
then
        does not exist
        return false
for each    i in 1 … N
        if    all leaders are selected
        then
                return true
        elseif    H[i] >= T
        then
                H[i] is leader
                i = i + K
        endif
endfor
if    the number of leader < L
then
        does not exist
        return false
return true
```

Check if there is enough number of giants, which is at least L + K(L-1). And then go through the height H[1..N]. Find the first giant that higher than T and select it as leader, and then move K spaces down and repeat. And finally, after go through the array, check if we find L leaders. If yes, then exist; otherwise, no. The time complexity is O(N).

(b).

```
Algorithm
Merge Sort H in ascending order
let T = H[N/2]
if T is possible for part(a)
then
        let T = H[3N/4]
        check by using part(a) principle
else
        let T = H[N/4]
        check by using part(a) principle
endif
do the binary search recursively until we found the optimal solution.
```

Use part(a) algorithm cost O(N) time, binary search recursively cost O(logN).

Therefore, total time complexity is O(NlogN).