Assignment 2 COMP9101
zID: z5119751
Name: SHIQING ZHANG
Q1.
(a) $P_A(x) = A_0 + A_1x + \dots + A_n x^n$
$\Downarrow$ DFT O(n log n)
$\{P_A(1), P_A(\omega_{2n+1}), P_A(\omega_{2n+1}^2), \dots, P_A(\omega_{2n+1}^{2n})\}$; ①

$P_B(x) = B_0 + B_1x + \dots + B_n x^n$
$\Downarrow$ DFT O(n log n)
$\{P_B(1), P_B(\omega_{2n+1}), P_B(\omega_{2n+1}^2), \dots, P_B(\omega_{2n+1}^{2n})\}$; ②

① and ②
$\Downarrow$ multiplication O(n)
$\{P_A(1)P_B(1), P_A(\omega_{2n+1})P_B(\omega_{2n+1}), \dots, P_A(\omega_{2n+1}^{2n})P_B(\omega_{2n+1}^{2n})\}$
$\Downarrow$ IDFT O(n log n)

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

In conclusion, convolution can be computed in time O(n log n).

(b) (i) Method (1)
There are K polynomials $P_1, \dots, P_K$,degree(P1) + · · · + degree(PK) = S
Suppose each Pi(i=1,2,…,k) has S items. For each Pi(i=1,2,…,k), if there is no such item, this item coefficient will be set as 0.like $0x^j$ ($0 \leqslant j \leqslant S$).
First, let $P_1 * P_2 = Q_1$, this step's time complexity is O(SlogS), because each one has s items.

Then let $Q_1 * P_3 = Q_2$, time complexity is also O(SlogS).
…
$Q_{k-1} * P_K$ time complexity is also O(SlogS), same as before.

In conclusion, the total time complexity is O(KSlogS).

Method (2)
There are K polynomials $P_1, \dots, P_K$,degree(P1) + · · · + degree(P_K) = S
Suppose each Pi(i=1,2,…,k) has S items. For each Pi(i=1,2,…,k), if there is no such item, this item coefficient will be set as 0.like $0x^j$ ($0 \leqslant j \leqslant S$).
Each P_i (i = 1, … k), using DFT in O(SlogS) to represent by point-value representation, so K polynomials time complexity is O(KSlogS). Then multiplication by point and then IDFT. The total complexity is O(KSlogS).

(ii)
The complexity is O(SlogSlogK).
Consider doing the first two-by-two multiplications, P1 by P2, P3 by P4, P5 by P6, and the complexity is O((d1+d2)log(d1+d2)+(d3+d4)log(d3 +d4) + ...) < O((d1+d2+d3+...)logS) = O(SlogS).
Suppose that last step is the situation that Multiplied by two polynomials of degree a and b are O((a+b)log(a+b)). degree is constant and the complexity of the next two-by-one multiplication is still O(SlogS), and there is a total of logK required.
Thus, the complexity is O(SlogSlogK).

Q2.
There are N coins in total. These values are between 1 and M ($M \geq N$).
Suppose that is two polynomials, which are $A_0 + A_1 x + \dots + A_M x^M$ and $B_0 + B_1 x + \dots + B_M x^M$.
Because the value of coins is only 1 and M, not including 0, A_0 is always 0.
Step 1: Time complexity is O(M)
for each $A_i x^i$ ($1 \leq i \leq M$),$B_i x^i$ ($1 \leq i \leq M$):

If a coin value i does not exist, this A_i is 0.

If i exists in the range of coins value, A_i is equal to 1. B_i is equal to times of coin value i appearing in all coins.

Step 2: Time complexity is O(M log M)

calculate the result of $A_0 + A_1 x + ... + A_M x^M$ multiply with $B_0 + B_1 x + ... + B_M x^M$, the result is written as $C_0 + C_1 x + ... + C_{2M} x^{2M}$

Step 3: Time complexity is O(M)

if $C_i > 1$ ($1 \le i \le M$), value i is one possible sum of two coins among all coins.

In the end, we can find all possible sum in this way.

Thus, the total time complexity is O(M log M).

Q3.

(a)

Method (1)

First, we have known that $\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$

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

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

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} F_0 \\ F_{-1} \end{bmatrix}$$

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

F(1}=1,F(0)=0,F(-1)=1

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

Method (2)

n=1,

$B^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$

Assume when n = k

$B^k = \begin{bmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{bmatrix}$

So let n = k+1

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

So n≥1, it is always correct.

(b)

$$A^n = \{ \begin{matrix} A \cdot A^{2^{\frac{n-1}{2}}}, \text{if } n \text{ is odd} \\ \left(A^2\right)^{\frac{n}{2}}, \quad \text{if } n \text{ is even} \end{matrix}$$

If n is odd, F(n - 1) could be calculated by dividing to two, repeat do that while n = 1.

Calculate F((n-1) / 2) , F((n-1) / 4), … ,F(1) times itself, then multiply

If n is even, same as odd, calculate F(n / 2), F(n / 4), …, F(1) times itself can acquire the result, time complexity is O(log n)
Thus, total time complexity is O(log n)
Pseudocode :

```
Function F(n):
    input n, the nth Fibonacci numbers
    output the nth Fibonacci numbers
    initial base_case = ((1, 1), (1, 0))
    if n = 1:
        return base_case
   if n is even:
       return F(n / 2) ^ 2
   if n is odd:
       return F(n-1) * base_case
```

Q4.
There are N items and Alice wants A items and Bob wants B items.
A+B≧N, each items must be bought by one of them but not both.
We have payment list Alice a[1…N], Bob list b[1…N] for every items.

Persuade code:

```
Function find_max_money(N, a, b, A, B):
      Initial  c= list(),positive_number=0, zero_number=0,  max_amount=0
      For i in range N:
            c.append([i, a[i]-b[i]])
      Mergesort c according to c[i][1] in descending  order  in O(NlogN)
#if c[i][1]>0, which means Alice pays more than Bob for item c[i][0].
#if c[i][1]<0, which means Bob pays more than Alice for item c[i][0].
      For i in range N:
            If c[i][1]>0:
              positive_number++
            If c[i][1]==0:
              zero_number++
       If  A<= (positive_number+zero_number):
            #A items all are sold to Alice
            # c[i][1]=0 means both of them pay the same price for item c[i][0], as for max_amount, choose either
of them is same
            for i in range A:
                  max_amount+=a[c[i][0]]
            for i in range(A,N):
                  max_amount+=b[c[i][0]]
       If  A > (positive_number+zero_number):
          If  B > = (N-positive_number-zero_number):
             for i in range (positive_number+zero_number):
                  max_amount+=a[c[i][0]]
             for i in range(positive_number+zero_number,N):
                   max_amount+=b[c[i][0]]
          If  B < (N-positive_number-zero_number):
              for i in range(N-B,N):
                   max_amount+=b[c[i][0]]
              for i in range (N-B):
                  max_amount+=a[c[i][0]]
       return max_amount
```

The merge sort time complexity is O(N log N), the loop time complexity are O(N).all the loops are parallel.
Thus, the total time complexity is O(N log N)

Q5.

(a)

persuade code :

```
Function decisionVersion(N, K, L, H[1…N], T):
   input N    # N giants in a line
          L    #the number of leaders
          K   #every pair of leaders at least K giants between them
          H[1…N]    #the heights of giants list
          T    #shortest leader's height is no less than T

   output True, if exists;
   otherwise,  False
initial i = 0
while i <= N:
# satisfied L are all found, break the loop
     if L == 0:
        return True
     if H[i] < T:
        i + +
# if H[i] is a candidate leader, then jump the K gap
     If H[i] >= T
       L - -
       i = i + K + 1
       if L == 0:
          return True
      else:
          return False
```

The total time complexity is O(n), because of only a loop.

(b)

persuade code :

```
Function optimisation_Version(N, L, K, H[1…N]):
   input N    # N giants in a line
          L    #the number of leaders
          K   #every pair of leaders at least K giants between them
          H[1…N]    #the heights of giants list
          T    #shortest leader's height is no less than T
   output maxInShortest, the maximum height of the shortest leader
among all valid choices of L leaders
      if maxInShortest does not exist, then -1
copy_list is a copy of H
sorted_copy_list = sorted(copy_list)
initial candidate_list = []
initial left = 0, right = N
mid = (left + right) / 2
# by using binary search
while left < right:
    if decisionVersion(N, L, K, H[1…N], mid):
       candidate_list.append(mid)
       left = mid + 1
    else:           # decisionVersion() return false
       right = mid - 1
       mid = (left + right) / 2
if not candidate_list:
   return -1
```

```
else:
    return max(candidate_list)
```

The sort is O(N log N), the binary search is O(logN) and the function decisionVersion time complexity is O(N).
Thus, the total time complexity is O(N log N)