Comp3121 assignment 2 Question 4
Chen En (z5335039)

Question 4

[20 marks] Alice and Bob are meeting up at the park. Alice arrives at time $t \in \mathbb{Z}^+$ with probability a_t (where $\sum_t a_t = 1$). Bob arrives at time $t \in \mathbb{Z}^+$ with probability b_t (where $\sum_t b_t = 1$). Let n be the number of distinct t's.

4.1 **[4 marks]** Provide an expression for the probability that Alice arrives k minutes before Bob (where k can be negative).

4.2 **[10 marks]** Design an $O(n \log n)$ algorithm that computes the probability that Alice arrives before Bob.

> How might FFT fit into this problem?

4.3 **[6 marks]** Design an $O(n)$ algorithm to compute the probability that Alice and Bob arrive an even number of minutes apart.

> Note: this is tricky!

4.1
when k ≥ 0;
Alice before k min than Bob:

$$when\ k\ >=\ 0,\ \sum_{t=1}^{n-k} a_t b_{t+k}$$

when k < 0;
Bob before k min than Alice:

$$when\ k\ <\ 0,\quad \sum_{t=1}^{n+k} a_{t-k} b_t$$

4.2
Given by the question, our goal is to compute the probability that Alice arrives k minutes before Bob. Apply from 4.1, we know that the probability can be calculated by

$$\sum_{t=1}^{n-k} a_t b_t + k$$

which we the equation can be expanded to

$$C(x) = a_1 * b_2 + a_1 * b_3 + a_1 * b_4 + \cdots + a_1 * b_n + \ldots + a_2 * b_3 + \ldots + a_{n-1} * b_n.$$

Let $C(x) = A(x) * B(x)$, which A(x) and B(x) are two polynomials of degree n.
In order to make the all possible combination of Alice coming k minutes before Bob in the term of same degree in C(x), we let:

$$A(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + \ldots + a_{n-1} * x^2 + a_n x$$
$$B(x) = b_1 x + b_2 x^2 + b_3 x^3 + \ldots + b_{n-1} * x^{n-1} + b_n x^n$$

To compute C(x) we take $A(x) * B(x)$. (C[i] means the coefficient of x^i)

A(x) * B(x) =

case1:
$a_n b_1 x^2 +$ => C[2] x^2
$a_n b_2 x^3 + a_{n-1} b_1 x^3 +$ => C[3] x^3
$a_n b_3 x^4 + a_{n-1} b_2 x^4 + a_{n-2} b_1 x^4 +$ => C[4] x^4
…..
$a_n b_{n-t} x^{(n+1-t)} + \cdots + a_{n-1} b_{n-1-t} x^{(n+1-t)}$ => C[4] x^4

case 2:
$a_1 b_1 x^{n+1} + \cdots + a_n b_n x^{n+1}$ => C[n+1] x^{n+1}

case3 :
$a_1 b_2 x^{n+2} + \cdots +$
...
$a_1 b_{1+t} x^{n+1+t} + a_2 b_{2+t} x^{(n+1+t)} + \ldots.$ => C[n+1+t] $x^{(n+1+t)}$
...
$a_1 b_n x^{2n}$

The indices value relationships between a and b can present different cases of the possibility of arriving.
Case:

1. When Bob arrives earlier than Alice, the index of a is greater than the index of b.
2. Bob and Alice arrive at the same time which happens when a and b have the same index.
3. When Alice arrives earlier than Bob, the index of a is less than the index of b.

From the formula above, we can conclude that if Alice arrives t second earlier, the probability will be the coefficient of $x^{(n+1+t)}$.
We use the divide-and-conquer strategy to calculate the multiple of two polynomials mentioned in the lecture slide of FFT to calculate the coefficients of A(x) * B(x).

We notice that if we want to find the probability of Alice arriving before Bob, just simply find the coefficient from Case 3, which is $x^{(n+1+t)}$in C(x) where $1 \leq t < n$.

Time complexity

Since we use same approach from the lecture slide (5. The Fast Fourier Transform p15 - p20), we are dealing with two problem size of n/2. Our algorithm satisfies the recurrence which $T(n) = 2T(n/2) + cn$. Apply the master theorem, the time complexity is On(logn).

4.3

Since we only need to find the coefficient of the terms of even degree, we can divide this problem into two subproblems:

1. terms of even degree in A(x) multiply terms of even degree in B(x)
2. terms of odd degree in A(x) multiply terms of odd degree in B(x)

That is, we calculate the coefficients of A_even(x) * B_even(x) and A_odd(x) * B_odd(x), where

$$A_{even}(x) = a_{n-1}x^2 + a_{n-3}x^4 + a_{n-5}x^6 + a_{n-7}x^8 + \cdots$$
$$A_{odd}(x) = a_n x^1 + a_{n-2}x^3 + a_{n-4}x^5 + a_{n-6}x^7 + \cdots$$

$$B_{even}(x) = b_2x^2 + b_4x^4 + b_6x^6 + b_8x^8 + \cdots$$
$$B_{odd}(x) = b_1x^1 + b_3x^3 + b_5x^5 + b_7x^7 + \cdots$$

For subproblem (1), we can apply the FFT algorithm to this problem, without the odd degree terms (i.e. A[1] mentioned in the slide). As for subproblem (2), there will not be the even degree terms (i.e. A[0] mentioned in the slide). Since they both lack either odd or even degree terms, there is no need to combine odd and even degree terms while running the algorithm. That is, we only approximately need a time of (n/2 + n/4 + …) to calculate each subproblem, which results in time complexity of O(n). The answer is the sum of coefficients of subproblems (1) and (2).

Time complexity

As mentioned above, there is no need to combine odd and even degree terms while running the algorithm. Therefore, when calculating the time complexity, we can drop the O(n) term, which results in:

T(n) = T(n/2) + T(n/2), which is O(n).