I decided to apply Business initially, without a deep understanding of the subject. However after Having studied Economics, I felt like this subject requires a lot of memorization and the theories being taught are far from the reality and often rarely offer little insight into how the real business are conducted. Later I found out that, it is the pitfall for the subject as a the whole, so I gradually lost interests.

While in the summer of 2018, I attended a summer school regarding to Computer Science at University of Cambridge, which was my first time experience programming and I was strongly attracted by the creative logic and the innovative spirit among the computer science courses. I was so impressed about the different methods to print out fibonacci numbers. The first one that came up in my mind is using recursion. then If n<=1, return n , else return fib(n-1)+fib(n-2). However, I realized that recursive calls take up too much memory space and there are too much repetitive operations. It’s time complicity can be represented by a tree diagram, there are n layers, so the maximum number of leaves is 2 to the power of n. Therefore, the total time complexity is 2 to the power of n, which is very slow, because the rate of growth of exponential function is faster than the polynomial. Then, I found an iteration method to handle it, by storing and updating the previous two Fibonacci numbers so that we save the time to recalculate the previous Fibonacci numbers. The time complexity of this method is O(n) which is much faster than the former one. However, I found an O(log n ) method online by using matrix. Firstly, we use the first number of the matrix A(n) to represent the (n+1)th Fibonacci number and then we create a 2x2 matrix called A1 which has (1,1) on the left-hand side and (1,0) on the right-hand side. After that, we use f(n)\*f(m)=f(n+m) to get the consecutive Fibonacci numbers. The reason why the time complexity is log n is if we use binary division we will get log n operations for the A(n) and each operation uses O(1). Therefore, the total time complexity is O( log n). Through Fibonacci numbers algorithms, I learned that the repetitive operations lead to the large increase in the time complexity . With the development of computer storage ,we can optimize algorithms by sacrificing the memory space to store the data that we need to be used more than once, and also this is the idea of dynamic programming and we use it in the iteration method to optimize the algorithms. While, for the matrix method, there is no repetitive data to be calculated, so it is effective. In addition, I found out that there are many different algorithms with different time complexities for the same problem. Therefore, computer scientists keep updating the original algorithms and aimed at using less computer resources to solve the problem.

After I coming back from the summer school, I took A-level computer science. However, I thought the A-level content wasn’t enough and I wanted to learn more extracurricular knowledge, so I self study by reading the ‘introduction to algorithms ‘ . At the beginning, I was quite confused about the divide and conquer topic because the way it solve the problem is totally different from human’ logic. Lets take sorting as an example, human are prefer to use insertion sort while sorting ,which means in iteration i, swap a[i] with each larger entry to its left until the end. However , the time complexity of the insertion sort is O( n squared), because there is a nested for loop and the inside and outside for loop both take O(n) ,so the total time complexity is O( n squared). It is quite slow compared to other sorting algorithms. After I searched information online and discussed with my teachers , I successfully figured out what is divide and conquer . It divides up problem into several subproblems and solve each subproblem recursively.Finally we combine solutions to subproblems into overall solution. Moreover, I learned some faster sorting algorithms involves the use of divide and conquer ,which is Merge sort and Quick sort .For merge sort , we recursively divides the list into two haves ( left and right ) until there is only one element in each array and then merge two halves to make sorted whole. Since it uses divide and conquer , so it’s time complexity( T(n) ) can be represented as 2T(n/2)+n. Therefore we can use binary tree to represent the time complexity . For each layer of the tree , the total time complexity is n and there are log n layers .Therefore the time complexity for merge sort is O( n log n ).When we consider the worst case , the time complexity is still O(n log n) . For quick sort , we choose a pivot ( a[j] ) and do partition so that no larger entry to the left of the pivot and no smaller entry to the right of the pivot, then we sort each piece recursively . It’s time complexity is still n log n, but for the worst case which is a list of number in descending order ,it needs O(n square) time and then I came up with another question . It seems like merge sort has less time complexity than the quick sort in the worst case , but why people are prefer to use quick sort? After I discussed with my teacher , I realized that the quick sort is an in-place sorting algorithms which means no extra memory needed ,while merge sort requires extra space to store the left and right sub-arrays. Apart from that , the worst case of the quick sort can be avoided by choosing the median among the first, middle and last elements as pivot .

In addition , I attended the ASDAN business simulation in 2017 , I recognized the importance of creativity and I learned how to express my ideas logically and explicitly .Apart from that , in the year of 2017, I got gold certificate in UKMT senior mathematical challenge , and Distinction In Galois contest.

When my finish my degree , I probably choose to take postgraduate courses of computer science.

Also, I would like to study machine learning and write some algorithms to apply it in the reality .