To me, algorithms are intertwined in everyday life. For example, a single click on Google triggers an incredibly complex reaction in the digital world, and a single touch on a mobile phone screen translates into content – all made possible by algorithms. Even outside the world of computing, algorithms still cast their magic, for example, when searching for a word in a dictionary, we tend to use binary search albeit subconsciously.

Farmers serve a fundamental function in a kingdom. To me, the modern computer system is a giant kingdom, where sorting algorithms are its farmers. Till now, I’ve learnt a great range of sorting algorithms, from the clumsy BubbleSort to its big brother QuickSort, and from MergeSort to HeapSort, and BogoSort, which seems ridiculous but can theoretically work efficiently on a quantum computer. For all comparison sorts, I learnt that it’s impossible to be faster than O(nlog n) on average, and we can derive such a theorem easily through a decision tree with a depth of n. Although there are linear time sorting algorithms, they are limited to some specific circumstances, for instance, if Counting Sort is used to sort an array with no repeated number, the time complexity is O(n^2) instead of O(n).

Once I was working on a difficult sorting algorithm test and ran out of ideas. With curiosity, I inputted sorted(a), the in-build function of python, to test how fast it is, and the result far exceeded my expectation – this function beat 99% others used! Later, I learnt sorted(a) uses a complex algorithm called Timsort, a hybrid of insertion sort and MergeSort. It has the best time complexity of O(n), an average and the worst time complexity of O(nlogn), and the worst space complexity of O(n).

The idea of using different sorting algorithms to treat arrays of different lengths inspires me. I started to code insertion sort, QuickSort and MergeSort using python. When the length of lists is extremely small (<64), QuickSort works as fast as insertion sort, and both outperformed MergeSort. As the length increases, QuickSort works the fastest. However, when the length is greater than a million, MergeSort was the most efficient. By combining different algorithm, I coded a function which shortened the sorting time significantly. I then adapted the functions to optimise performance. While I improved the efficiency for insertion sort and MergeSort, my adaptation of QuickSort, led to an unexpected result. I first shuffled the array randomly, then selected the middle number of the first three elements as a pivot. When comparing, I added a third part for elements which are equal to the pivot. Surprisingly, the adaptation reduced the efficiency, as they did not serve randomly generated fractions. Also, I realised that all the functions I coded are much slower than sorted(), as the latter is written in C to solve issues at low-level, including uses of cache memory and parallelism.

I used to think of computer merely as a tool to execute algorithms, but as I learnt more, I developed a holistic perspective. We cannot use algorithms directly to solve real-life questions, and we are incapable of optimising functions without knowing the feature of the data we are going to deal with, the mechanism of a computer system, and even the performance of hardwires. Instead of only learning about algorithms, I began to read and research more about computer systems such as the CSAPP.

Computer science is a subject of precision, creativity and logic. The programs computers run are among the most complex products ever created, and I am eager to learn how to design and use them effectively at a deep level, and learn about different software and hardware and their applications. It is my desire to pursue a Computer Science degree course that focuses on creating links between theory and practice at a world-class university, I am looking forward to the excitement and challenge the course will present at a higher level.