Este semestre tuve la oportunidad de realizar diversas actividades que me retaron a desarrollar habilidades y competencias que yo utilizaré en un futuro. Las actividades eran entretenidas debido a que había un nivel de dificultad que yo debía ser capaz de afrontar con los conocimientos que me había presentado el profesor durante sus clases. Empezamos entonces con diferentes métodos de ordenamiento y de búsqueda. Dichos métodos iban aumentando en complejidad pero también iban aumentando en rapidez y eficiencia. Al final, por más que se me hizo más fácil programar “quick sort”, “merge sort” terminó siendo el método de ordenamiento que más rápido lograba ordenar los datos. Este método lo empezamos a utilizar cuando trabajamos con “linked lists”, las cuales eran lo que nosotros llamábamos “super vector” o en mi caso “lista Frankenstein”. Normalmente, tendríamos que estar recorriendo apuntadores de manera secuencial para este ordenamiento. Claro que a mi se me ocurrió pasar los apuntadores de ciertas posiciones como parámetros para que el código fuese capaz de accesarlos de manera más precisa. Esto hizo que el código pasara de tardar casi un minuto a menos de 4 segundos. Es impactante como unos pequeños cambios logran hacer que algo sea más eficiente. Esto fue muy satisfactorio debido a que el propósito de la clase es que eventualmente nosotros estaremos en un lugar en donde estaremos buscando como hacer el código de manera más eficiente. Eventualmente llegamos a los árboles de búsqueda binaria, los cuales me parecían estructuras con mucho potencial, sobre todo el tema de max heap. Para esta actividad se nos dijo que podíamos usar cualquier tipo de estructura de datos que hubiésemos visto durante el semestre, siendo vector la más sencilla de utilizar. Sin embargo, sentía yo que se podía lograr que con un linked listed se obtuviese un max heap más rápido al max heap con vector. Intente crear una linkedlist en la cual los nodos tuviesen un hijo derecho y un hijo izquierdo para que así no hubiera necesidad de recorrer secuencialmente y accesar datos de manera más inmediata. Al final tuve muchos problemas y me estaba quedando sin tiempo, por lo que tuve que optar por usar vectores, pero aun siento que sería más rápido el ordenamiento de la manera que yo mencione. Luego llegó la teoría de grafos, la cual fue bastante interesante y fue el tema que más me llamó la atención. Siento que hubiese estado mejor dedicarle más tiempo a esta teoría y dejarla más al final para así poder ver cómo incorporar todo lo visto en el semestre. El algoritmo de Dijkstra fue muy interesante y al principio me estaba complicando demasiado. La solución del profe terminó siendo mucho más simple que la mía y mucho más eficiente, y siento que no habría mucha más forma de hacer más eficiente el código. Pero me gustaría hacer énfasis en que me hubiera gustado poder haber hecho más uso de Grafos. Al final terminamos la clase con el tema de Tablas Hash, las cuales igual llamaban mucho mi atención debido a su forma única de almacenamiento. Siento que podríamos haber hecho más eficiente nuestra tabla hash si hacíamos uso de un struct para el almacenamiento de los datos, pero la manera en la que lo realizamos siento que también fue bastante eficiente. Al final del semestre terminé con un nivel de satisfacción muy elevado ya que fue muy gratificante voltear a ver atrás y observar todo el camino que llevamos en la clase. Me hubiera gustado que esta clase la tuviésemos más de 4 horas a la semana; le saque mucho provecho.