Alejandro Lizárraga Vizcarra A01284610

Reflexión Evidencia 2

Para esta evidencia se tuvo que diseñar un programa que era capaz de leer la información de un archivo para ordenarla. Después con esta información ordenada se buscaría mediante un identificador encontrar las cantidades de naves que entran por diferentes mares por cada mes registrado en el archivo de datos. Para esta implementación utilizaríamos las listas doblemente ligadas.

Para este proyecto se utilizaron templates y clases para crear los diferentes componentes que se necesitarían para una lista doblemente ligada. Se creo un Nodo como la entidad individual dentro de una lista ligada, un Log para administrar la información que se leería del archivo y un DoubleLinkedList para administrar los apuntadores de los nodos.

Para el programa utilizamos una búsqueda secuencial para encontrar el identificador, esto es debido al tipo de estructura de datos que estamos utilizando. En la evidencia anterior habíamos utilizado una búsqueda binaria, esto es porque utilizábamos un vector. Al utilizar una doubly-linkedlist para llegar al valor de cierto índice se debe de recorrer secuencialmente por los apuntadores de cabeza o cola por la naturaleza de la estructura. Aunque en el código parece que solo llamamos al índice y obtenemos el dato a diferencia de una has table donde se tiene acceso a los datos “en medio”, en una linkedlist es necesario recorrer los elementos con apuntadores. Debido a esto si utilizamos un método de búsqueda binaria utilizando índices del linkedlist, por cada iteración de los pasos de la búsqueda binaria hay una búsqueda secuencial. Esto hace que el procesamiento de ambos métodos se multiplique por lo que la búsqueda binaria ahora sería de **O(N log (N))**, esto es considerablemente más lento cuando se tratan cantidades grandes de datos que con una búsqueda secuencial de **O(N)**. Y es por esta razón que se utilizó una búsqueda secuencial. Probado el código también se determinó que el sistema en el que se corrió encontró los datos más rápido con una búsqueda lineal que secuencial.

También se utilizó el método de Quicksort. Este normalmente tendría una complejidad de **O(N log (N))** cual es bastante bueno y en peor caso la complejidad es de **O(N^2)**. Pero también aquí tenemos acceso a datos mediante el uso de índices en la linkedlist, por lo que se multiplicaría una complejidad secuencial y esto lleva a que tenga complejidad de **O(N^2 log (N))** cual es bastante bueno y en peor caso la complejidad es de **O(N^3)**, en este caso no teníamos mucha flexibilidad para el ordenamiento según la estructura de datos y por eso se utilizó el quick sort que era bastante veloz para ordenar los componentes.

Como ventajas de la lista doblemente ligada tenemos que esta es modular por lo que se le pueden agregar o restar valores de modo que cambia su tamaño. Aparte el acceso directo con apuntadores abre posibilidades a una mejor administración de la memoria y los datos que se manejan. Como desventaja tenemos que los datos en medio no se pueden acceso sin antes pasar por otros valores ya sea mediante la cola o cabeza, esto lleva a que por detrás siempre haya un proceso secuencial a diferencia de un vector o un hash table donde se tiene acceso al dato directo por un indicador, a consecuencia de esto utilizar un procedimiento como un binary

search es más lento que un sequential search. Sin embargo, al ser doblemente ligada tenemos mejor eficiencia que con una lista ligada en un solo sentido, sobre todo cuando se tratan de los datos que se encuentran después de la mitad de la lista, que en promedio representaría el 50% de los casos que se utiliza.