Francisco Rocha Juárez
Reflexión :

En esta situación problema se nos daba en un archivo de texto más de 16 líneas, donde cada una contenía un evento, este evento contaba con la fecha y hora. El problema es que estos eventos estaban en completo desorden, es por esto que teníamos que ordenarlos cronológicamente para su consulta, pero esto no es suficiente, también se le debe facilitar al usuario una manera en que consulte los datos de manera ordenada de una fecha de inicio a una fecha determinada.

Es por esto que recurrimos a los algoritmos de ordenamiento, para ordenar esta gran cantidad de datos, en esta ocasión nosotros implementamos el algoritmo de ordenamiento llamado: Ordenamiento por intercambio, el cual es el más sencillo de implementar, pero no el más eficiente. Este algoritmo está conformado por el siguiente código:

```
void Sort(){
    for (int i = 0; i < registros.size(); i++)
    {
        for (int j = i; j < registros.size(); j++)
        {
            if (meses[i] > meses[j])
            {
                Switch(i, j);
            }
            else if (meses[i] == meses[j])
            {
                if (dias[i] > dias[j])
                {
                    Switch(i, j);
                }
                else if (dias[i] == dias[j])
                {
                    if (horas[i] > horas[j])
                    {
                        Switch(i, j);
                    }
                    else if (horas[i] == horas[j])
                    {
                        if (minutos[i] > minutos[j])
                        {
                            Switch(i, j);
                        }
                        else if (minutos[i] == minutos[j])
                        {
                            if (segundos[i] > segundos[j])
                            {
                                Switch(i, j);
                            }
                        }
                    }
```

```
                    }
                }

            }

        }


}
```

Este algoritmo tiene una complejidad de O(n^2), no es tan rápido como uno de O(n) o O(logn), sin embargo funciona bien para nuestro propósito en esta actividad.

Al momento de hacer las búsquedas ocupamos un algoritmo de búsqueda directa secuencial, el cual va recorriendo el arreglo y haciendo la búsqueda de uno en uno, la complejidad de este algoritmo es de O(n), no es tan eficiente como una búsqueda binaria de complejidad logarítmica, sin embargo nos sirve bien para esta actividad. El algoritmo está descrito en el código a continuación:

```
for (int i = 0; i < registros.size(); i++){
        if (meses[i] >= mesi && meses[i] <= mesf){
            if (meses[i] == mesi && dias[i] >= diai)
            {
                cout << registros[i] << endl;
            }
            else if (meses[i] == mesf && dias[i] <= diaf)
            {
                cout << registros[i] << endl;
            }
            else if (meses[i] != mesf && meses[i] != mesi)
            {
                cout << registros[i] << endl;
            }

        }
    }
```

En este tipo de situaciones problema, los algoritmos de ordenamiento y búsqueda son muy importantes y es aún más importante poder decidir cual de todos los algoritmos debemos usar, ya que en algoritmos de búsqueda contamos con algunos como lo son:

-Ordenamiento por intercambio O(n^2)
-Bubble Sort O(n^2)
-Insertion Sort O(n^2) en el peor de los casos
-Merge Sort O(nlogn)
-Quick Sort O(nlogn) en el mejor de los casos y O(n^2) en el peor de los casos

Todos nos sirven para ordenar arreglos o vectores, sin embargo, unos son más rápidos que otros en general o algunos hacen menos iteraciones dependiendo de la manera en que el

arreglo venga previamente ordenando, dependiendo el conjunto de datos que tengamos, es el tipo de algoritmo que tendríamos que elegir.

Y para hacer la búsqueda de datos algunos ejemplos de búsqueda serían la búsqueda directa y la búsqueda binaria, con complejidad O(n) y O(logn) respectivamente. La búsqueda binaria es una mejor opción ya que con cada iteración va descartando la mitad de los resultados, obteniendo el resultado de una manera más rápida, mientras que el algoritmo de búsqueda directa secuencial hace más iteraciones, pasando los datos del arreglo uno por uno hasta encontrar el deseado.