# Reflexión Individual
# Actividad 1.3

**Santiago Álvarez Valdivia**

**A01640172**

Programación de Estructuras de Datos y Algoritmos Fundamentales

**Tecnológico de Monterrey**

Campus Guadalajara

Escuela de Ingeniería y Ciencias

Profesor: Dr. Eduardo Arturo Rodríguez Tello

**Martes, 21 de septiembre de 2021**

# Índice



## Índice de figuras



## Índice de tablas

# Introducción

Las computadoras son indispensables para el manejo de información en el mundo moderno. Actualmente no hay entidad importante que no las utilice para guardar la información de sus clientes, compras, ventas y un largo etcétera, usualmente en forma de listas o arreglos. Por su parte, la búsqueda de información es parte fundamental de dicho manejo de información, ya que esta es una de las aplicaciones más directas del almacenamiento de información.

Por supuesto, al trabajar con cantidades importantes de elementos (que llamaremos registros) en una lista, que son buscados muchísimas veces al día, la eficiencia de la búsqueda de información se vuelve clave. Cabe mencionar que si bien es posible buscar datos en arreglos sin ordenar, la mayor eficiencia se da en arreglos ordenados de acuerdo a algún criterio.

Así llegamos a otra problemática, que es encontrar el modo más eficiente posible para ordenar la información, para que esta pueda ser buscada eficientemente. A diferencia de los algoritmos de búsqueda, que prácticamente se pueden agrupar en sólo dos categorías (secuencial y binaria, siendo esta última la mejor), existe una gran variedad en algoritmos de ordenamiento, con ejemplos como el ordenamiento por intercambio, por inserción, de burbuja, por fusión, etc.

¿Y como sabemos cual es el algoritmo más eficiente? Para eso, hay múltiples modos de medición. Uno de ellos es la complejidad en función del número de elementos involucrados. Esta se refiere al número de procesos ejecutados de acuerdo a (en este caso) el número de elementos del arreglo. Para escribirlo se utiliza la notación:

$$O(f(n))$$

En donde la función $f(n)$ indica el orden de crecimiento de la complejidad conforme aumenta el número de elementos. Mientras menor sea el orden de crecimiento, mejor.

Otro modo de medir la complejidad es el espacio en memoria que utiliza un algoritmo, que se expresa mediante la misma notación.

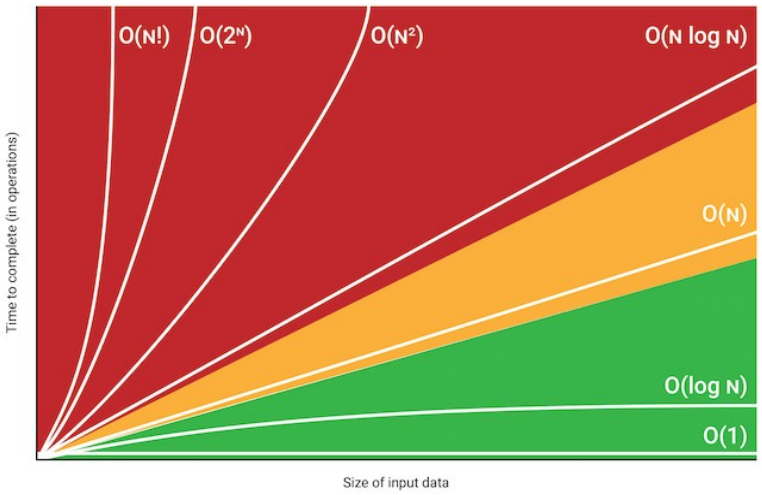


***Figura 1:*** *Orden de crecimiento de distintas funciones f(n)*

# Algoritmos de Ordenamiento

Un algoritmo de ordenamiento es aquel que pone elementos de una lista o arreglo en una secuencia lógica, ya sea de menor a mayor, mayor a menor, orden alfabético, entre otros.

# Elección de Algoritmo

El algoritmo de ordenamiento utilizado para la solución de la problemática fue el llamado “Quick Sort” (o ordenamiento rápido en español).

| Ordenamiento Rápido (Quick Sort) | |
|---|---|
| Complejidad en función de n | |
| Mejor | $O(n \log n)$ |
| Promedio | $O(n \log n)$ |
| Peor | $O(n^2)$ |
| Estable | Sí |
| Espacio en Memoria | $O(\log n)$ |

***Tabla 1:*** *Características del "Ordenamiento Rápido"*

Como podemos ver, este algoritmo es eficiente en términos de rapidez de ejecución, y no realiza una gran carga en la memoria, lo que lo hace factible para el uso en este programa.

El quick sort funciona al seleccionar un elemento aleatorio del arreglo llamado pivote (que en mi implementación es siempre el último), y dividir el arreglo en 2 partes, una con los elementos menores que el pivote, y otra con los elementos mayores que el pivote. El proceso se realiza recursivamente hasta que el arreglo queda ordenado.

# Algoritmos de Búsqueda

Un algoritmo de búsqueda es aquel cuya tarea es encontrar un determinado elemento dentro de un arreglo. La variedad en estos algoritmos es muy reducida en comparación con los algoritmos de Ordenamiento. Así tenemos dos categorías generales: los secuenciales (especialmente para arreglos no ordenados, aunque no limitados a ellos) y los binarios (para arreglos ordenados), que tienen la mayor eficiencia.

# Elección de Algoritmo

Como en este programa estamos utilizando arreglos ordenados, un algoritmo de búsqueda binario es la mejor opción. A su vez, se decidió que su implementación fuera recursiva, para poner en práctica aspectos de recursividad.

| Búsqueda Binaria | |
|---|---|
| Complejidad en función de n | |
| Mejor | $O(1)$ |
| Promedio | $O(\log_2 n)$ |
| Peor | $O(\log n)$ |
| Espacio en Memoria | $O(1)$ |

***Tabla 2:*** *Características de la Búsqueda Binaria*

El modo en que la búsqueda binaria funciona es primero comparar el elemento en la posición del medio con el valor a buscar, si no encuentra allí el valor, al menos sabe en que mitad del arreglo se encuentra (hay que recordar que el arreglo está ordenado), y puede descartar la otra. El proceso se repite para la otra mitad, y así sucesivamente.

# Manejo de Fechas y Horas

La función del presente programa es tomar una serie de reportes de incidencia, leerlos y procesarlos computacionalmente, para después ordenarlos y hacer búsquedas.

Tanto el ordenamiento como la búsqueda se dan en función de la fecha y hora de cada registro. Para una mayor facilidad en el manejo de estas, se utilizan las llamadas *timestamps*, que son fechas expresadas de manera numérica.

Los reportes originales de incidencia están en el siguiente formato:

```
Sep 23 12:58:18 80.169.79.65:1150 Failed password for illegal user root
```

Es decir, Mes, Día, Hora, Dirección IP y Razón. Los primeros 3 son convertidos a la *timestamp* correspondiente. El año siempre es considerado como 2021.


***Figura 2:*** *Ejemplo de Conversión de Fecha a Timestamp*

Después, las *timestamp* son comparadas como números comunes a la hora de hacer búsquedas y ordenamientos.

# Husos Horarios y Daylight Saving Time (DST)

La conversión de fechas entre formato “humano” y timestamp no es tan sencilla. Mientras que los timestamps son absolutos para cualquier parte del mundo, el formato humano depende de la zona a la que se hace referencia, por ejemplo, la hora en Hong Kong no es la misma que la de Nueva York, aunque se consulten al mismo tiempo.

Por tanto, para hacer la conversión de formato humano a timestamp, se debe utilizar un huso horario en específico. El presente programa hace estas conversiones por medio de `mktime(struct tm)`, la cual toma en cuenta el horario local del dispositivo, es decir, que la conversión se hace de acuerdo a la zona horaria con la que esté registrada la computadora en la que se ejecuta el programa. La conversión inversa, que se realiza al momento de imprimir, utiliza `localtime (time_t)`, la cual es el inverso de `mktime`.

Por otra parte está el Daylight Saving Time (DST) o horario de verano, en español. Este consiste en añadir o quitar una hora al horario local en una cierta temporada, con el propósito de aprovechar mejor la luz solar. Las funciones `mktime` y `localtime` lo implementan automáticamente, ya que se especificó a la bandera `tm_isdst = -1`

# Referencias


[1] *C Epoch Converter Routines*. (s. f.). EpochConverter. Recuperado 16 de septiembre de 2021, de https://www.epochconverter.com/programming/c

[2] CPlusPlus. (s. f.-a). getline (string). En *C++ Reference*. Recuperado 15 de septiembre de 2021, de https://www.cplusplus.com/reference/string/string/getline/

[3] CPlusPlus. (s. f.-b). strftime. En *C++ Reference*. Recuperado 16 de septiembre de 2021, de https://www.cplusplus.com/reference/ctime/strftime/

[4] CPlusPlus. (s. f.-c). vector::size. En *C++ Reference*. Recuperado 15 de septiembre de 2021, de https://www.cplusplus.com/reference/vector/vector/size/

[5] GeeksforGeeks. (2021, 8 julio). *Exception Handling in C++*. https://www.geeksforgeeks.org/exception-handling-c/

[6] Mandliya, A. (2021, 13 abril). *How to Split String by space in C++*. Java2Blog. https://java2blog.com/split-string-space-cpp/

[7] Miessler, D. (2019, 14 septiembre). *Big-O Notation Explained*. Daniel Miessler. https://danielmiessler.com/study/big-o-notation/

[8] *Modern C++ best practices for exceptions and error handling*. (2020, 24 agosto). Microsoft Docs. https://docs.microsoft.com/en-us/cpp/cpp/errors-and-exception-handling-modern-cpp?view=msvc-160

[9] Reinstate, M. (2016, 16 marzo). *Interpretation of tm_isdst field in struct tm*. Stack Overflow. https://stackoverflow.com/questions/36031757/interpretation-of-tm-isdst-field-in-struct-tm