# Análisis Primer Punto

Para solucionar el primer punto inicialmente se utiliza el algoritmo de ordenamiento MergeSort para obtener un arreglo ordenado de las fechas de finalización de cada rodal.

Posteriormente con cada elemento del arreglo (que contiene cada rodal) verifico si las fechas no se solapan con los elementos siguientes del arreglo (fecha final comparada con la fecha inicial del siguiente rodal), y también se utiliza una función llamada reverse, con lo cual se verifica que las fechas del rodal seleccionado no se solapen con cada uno de los elementos anteriores (fecha de inicio con la fecha final del elemento anterior) y se genera una lista con todos los elementos compatibles.

Con cada una de las listas de rodales compatibles se obtiene el beneficio total que se obtendrá al talarlos y se almacena en una variable “temporal”.

Cada que se realiza este procedimiento se compara si el nuevo valor del beneficio es mayor al que se encuentra almacenado en la variable “temporal”, en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría, de lo contrario se queda con la escogida anteriormente.

# Pasos utilizados:

* Ordenamiento con MergeSort. 🡪 **O(n.lgn)**
* For(recorrer arreglo de elementos) 🡪Complejidad total **O(n2)**
  + funcionalidades.MaximoNumeroRodales 🡪 **O(n)**
  + funcionalidades.MaximoNumeroRodalesReverse 🡪 **O(n)**
* Comparación para saber si es el mayor elemento.
* Comparación() 🡪 **O(n)**

De esta forma determino que la complejidad del algoritmo es de **O(n2)**

# La complejidad de esta solución

Es determinado por la complejidad del algoritmo de ordenamiento MergeSort la cual es de **O(n.lgn).** Sumado a la complejidad de recorrer los elementos previamente organizados para su selección que obtenemos una complejidad de **O(n)** en cada selección pero al realizarlo en un ciclo **for** se convierte en **O(n2)** elemento de esta forma determina que nuestra solución tiene una complejidad total de **O(n2).**

Uso de aplicativo

Es un proyecto de Netbeans para su ejecución solo es necesario construir el ejecutable e iniciar la ejecución. **”Triángulo de reproducción”**



Al ejecutarse despliega una pantalla para la selección del archivo que se va a tomar como entrada, la estructura del archivo de entrada cumple con las especificaciones del documento del proyecto.

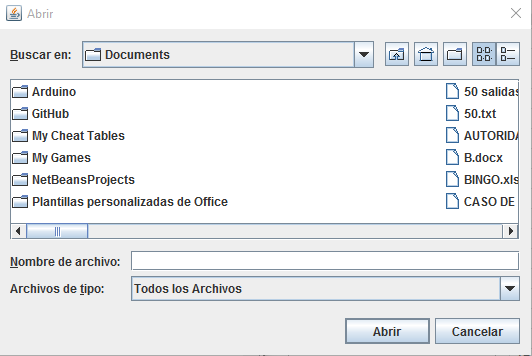


Ilustración 1

Posteriormente pide que se le indique un directorio para el almacenamiento de los resultados

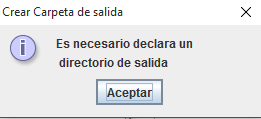


Ilustración 2

Se selecciona la ruta donde se va a realizar el almacenamiento de los resultados.

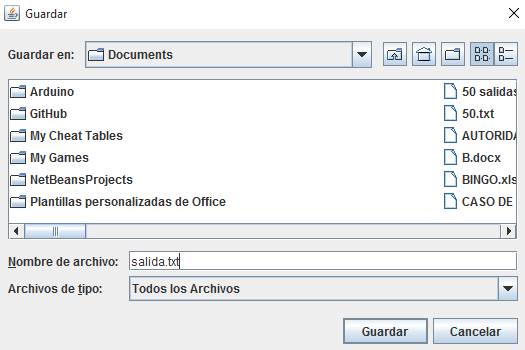


Ilustración 3

Al finalizar todos los procedimientos Retorna un mensaje de finalización.

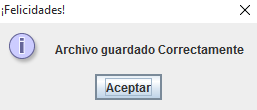


Ilustración 4

# Análisis de complejidad.

Se realizaron pruebas de rendimientos con múltiples entradas que fueron obtenidas con la construcción de un algoritmo de generación de entradas de forma aleatoria, Este aplicativo está disponible con el nombre **“Generador de prueba de entrada “**como proyecto de Netbeans los archivos de prueba se encuentra en la carpeta **"PruebasInput".**

# Tabla de rendimiento

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre de Archivo** | **Numero de registro de entradas** | **Tiempo Milisegundos** | **Numero de Iteraciones** |
| 50 registros | 50 | 532 | 3423 |
| 150 registros | 150 | 672 | 25123 |
| 500 registros | 500 | 5485 | 258573 |
| 1600 registros | 1600 | 14891 | 2587273 |
| 2999 registros | 2999 | 33579 | ……. |
| 5200 registros | 5200 | 33157 | …….. |

Ilustración 1

No. De Registros

No. De Iteraciones

Ilustración 2

Entorno de pruebas

Procesador: Amd E1-2500 1.40Ghz.

Memoria: 4GB Ddr3.

Disco duro: 500 Gb 5400 rpm.

Sistema Operativo: Windows 10 Pro 64bit.

Conclusiones

Como podemos observar aunque la complejidad en cualquiera de las pruebas realizadas es n2 el tiempo pude variar debido a que el procesador ejecuta diferentes tareas y le puede dar prioridad a otra lo que hará que se retarde el programa y un numero menor de registros nos de mayor tiempo de ejecución que el esperado.