Documentación Proyecto FADA

Edison Felipe Mamian Ceron – 1224279
Mauricio Castillo Mina – 1226715
Nathalia Bedoya Hurtado - 1226305

Fundamentos de Analisis y Diseños de Algoritmos

Universidad del Valle

Análisis Primer Punto

Para solucionar el problema 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 almacenada en el vector de tipo Rodal "MayorNumerodeArbolesTemporales".

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal "MayorNumerodeArbolesTemporales" se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles y esta se compara con la variable "MayorNumerodeArbolesTemporal" (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que "tmpNumerodeArboles" sea mayor a "MayorNumerodeArbolesTemporal", en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal "MayornumerodeArboles", de lo contrario se queda con la ya almacenada anteriormente.

Por ultimo al haber obtenido la mejor solución se pasa a guardar en un archivo plano la lista que este en el vector de tipo Rodal "MayornumerodeArboles" (teniendo en cuenta que el objeto contiene el numero del rodal, numero de arboles, de empleados y las fechas).

Pasos utilizados:

- Ordenamiento con MergeSort.(Por fecha de finalización del rodal) → **O(n.lgn)**
- For(recorrer arreglo de elementos) →Complejidad total **$O(n^2)$**
    - 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 **$O(n^2)$**

La complejidad de esta solución

Primeramente es determinado por la complejidad del algoritmo de ordenamiento MergeSort la cual es de **O(n.lgn).** En el segundo paso se le suma 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** (desde la posición en la que esta hacia delante, y en sentido contrario) se convierte en **$O(n^2)$.** De esta forma determina que nuestra solución tiene una complejidad total de **$O(n^2)$.**

PSEUDOCODIGO

Algoritmo Dinamico

```
ArrayList<Rodales> MergeOut = ordenamiento.mergeSort();


    for (int i = 0; to MergeOut.size()) {
       funcionalidades.Solucion.clear();
       funcionalidades.MaximoNumeroRodales(i);
       funcionalidades.MaximoNumeroRodalesReverse(i);
       funcionalidades.Probar();
    }
```

Funciones establecidas para la llegar a la solución optima

```
  ArrayList<Rodales> MaximoNumeroRodales(int inicio) {
    Punto ←inicio;

    inicial_Rodal ← entrada.get(punto);
    Solucion.add(inicial_Rodal);
    //Almaceno el valor de numero de arboles par saber quien es mayor
    MayorNumeroArbolesTemporal.add(inicial_Rodal);
    tmpNumeroArboles += inicial_Rodal.numero_arboles;


    //Este algoritmo resuelve el problema del máximo numero de rodales permitido
    for (int i = punto to entrada.size()) {
       Date date1 ← Solucion.get(Solucion.size() - 1).fecha_fin;
       Date date2 ← entrada.get(i).fecha_inicio;

       complejidad +=1;

       if (!date2.before(date1)) {
          Solucion.add(entrada.get(i));
          tmpNumeroArboles += entrada.get(i).numero_arboles;
          MayorNumeroArbolesTemporal.add(entrada.get(i));
       }
    }
    return Solucion;
  }


  /*Complejidad O(n)
```

```
  Esta función separa en un punto del arreglo que identifique con la variable de inicio y
compare si no se solapa con todos los elementos anteriores
  a el*/
  ArrayList<Rodales> MaximoNumeroRodalesReverse(int inicio) {
    int punto ← inicio - 1;


    ArrayList<Rodales> salida;
    //entrada.
    //Este algoritmo resuelve el problema del máximo numero de rodales permitido
    int contador ← 1;
    for (int i = punto to 0) {
      Date date1 ← entrada.get(entrada.size() - contador).fecha_inicio;
      Date date2 ← entrada.get(i).fecha_fin;


      if (date2.before(date1)) {
        salida.add(entrada.get(i));
        tmpNumeroArboles += entrada.get(i).numero_arboles;
        if (!MayorNumeroArbolesTemporal.contains(entrada.get(i))) {
          MayorNumeroArbolesTemporal.add(entrada.get(i));
        }
        contador++;
          }
    }
    return salida;
  }


//Complejidad O(1)
  //Realiza las comparación para saber si esta operación obtuvo la mayor ganancia en
arboles
  //Comparo el valor que ya existe y el nuevo si el nuevo es mayor almaceno el nuevo. de
lo contrario lo ignoro
 Probar() {
    if (tmpNumeroArboles > maximoNumeroArboles) {
      maximoNumeroArboles ← tmpNumeroArboles;

      MayorNumeroArboles ←  MayorNumeroArbolesTemporal;
      maximoNumeroArboles);
    }
    tmpNumeroArboles ← 0;
    MayorNumeroArbolesTemporal.clear();
  }
```

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.

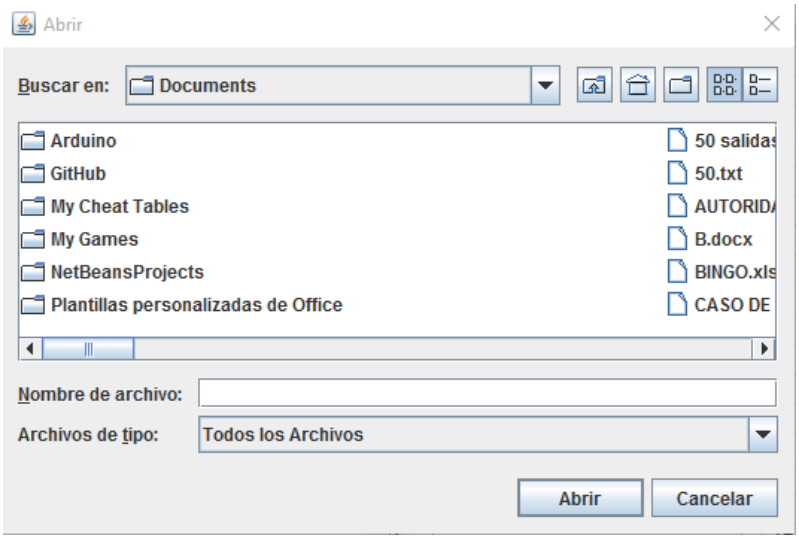

Ilustración 1

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

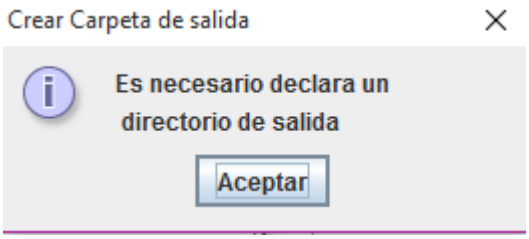

Ilustración 2

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

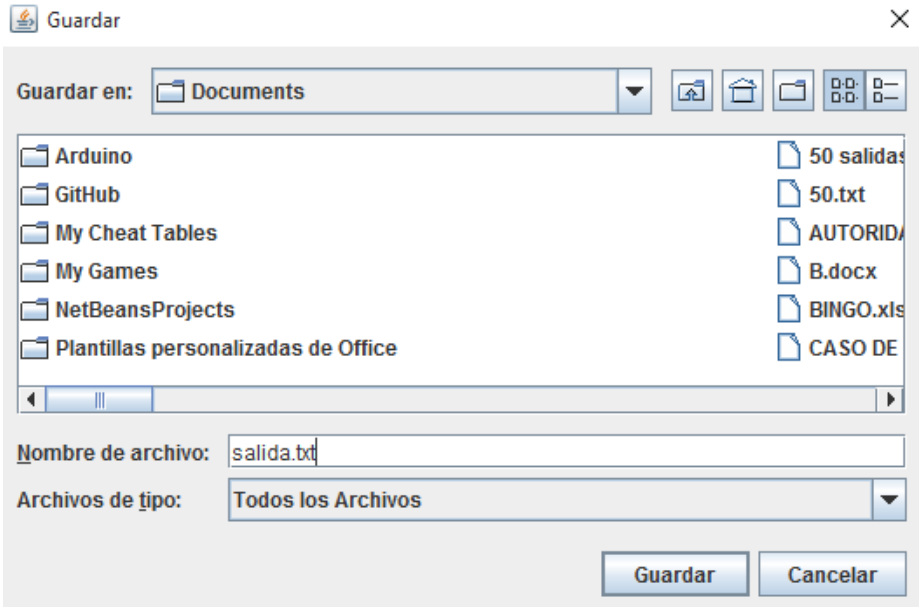

Ilustración 3

Al finalizar todos los procedimientos Retorna un mensaje de finalización.

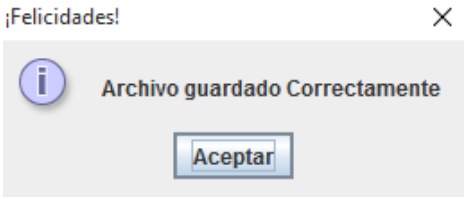

Ilustración 4

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	2	3423
150 registros	150	7	25123
500 registros	500	22	258573
1600 registros	1600	98	2587273
2999 registros	2999	208	…….
5200 registros	5200	342	……..

Ilustración 1

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.

Análisis Segundo Punto

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

Posteriormente gracias a este ordenamiento se obtiene el primer elementó en finalizar y se hace una búsqueda hasta encontrar un segundo elemento que permitiera su tala de árboles sin solaparse con el rodal anterior. Al ya haber realizado el ordenamiento por fecha de finalización se garantiza que el segundo elemento encontrado tiene una fecha de finalización previa al resto de elementos que se encuentra en el arreglo después de el.

Los rodales seleccionados se van almacenando en el vector de tipo Rodal "MayorNumerodeArboles"

Esto se realiza hasta terminar con todos los rodales disponibles dando como resultado la tala del mayor número de rodales sin solapar sus fechas. Por lo que se imprime en el archivo los rodales seleccionados.

Pasos utilizados:

- Ordenamiento con MergeSort. → **O(n.lgn)**
- Recorrido del arreglo encontrando el siguiente → **O(n)**

De esta forma determino que la complejidad del algoritmo es de **O(n.lgn)**

La complejidad de esta solución

Es determinado por la complejidad del algoritmo de ordenamiento MergeSort la cual es de **O(nlgn).** Sumado a la complejidad de recorrer los elementos previamente organizados para su selección que obtenemos una complejidad de **O(n)** de esta forma determina que nuestra solución tiene una complejidad total de **O(nlgn).**

Uso de aplicativo

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.

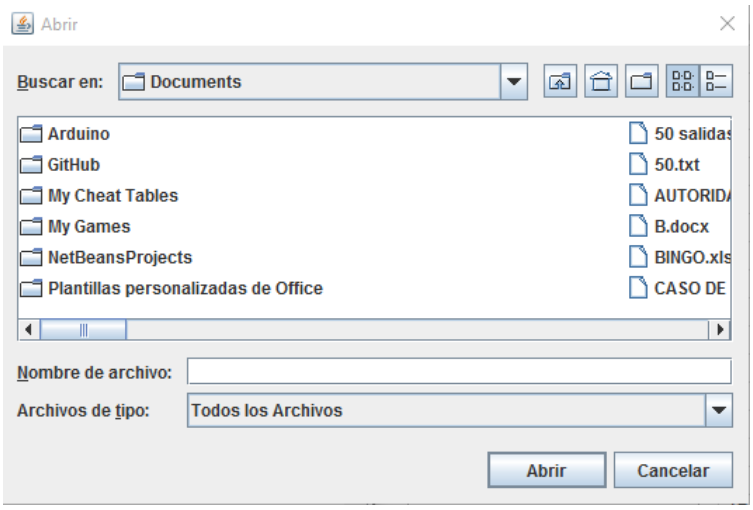

Ilustración 5

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

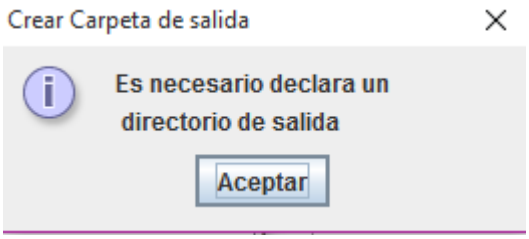

Ilustración 6

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

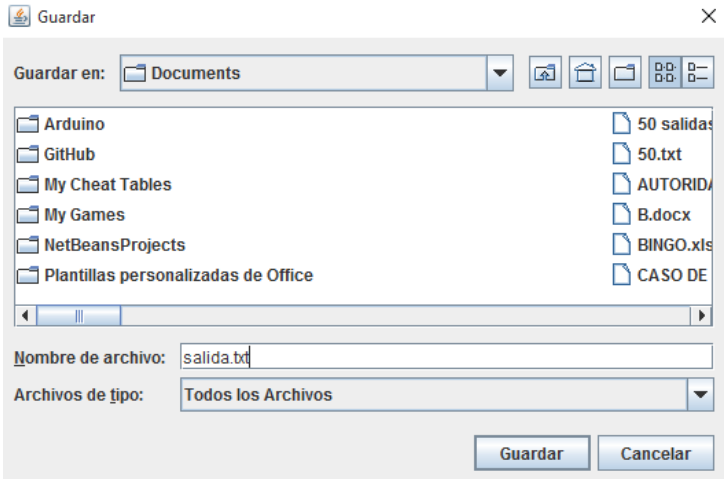

Ilustración 7

Al finalizar todos los procedimientos Retorna un mensaje de finalización.

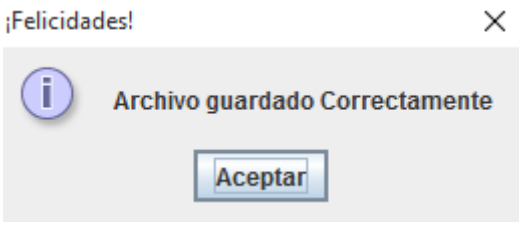

Ilustración 8

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	Número de registro de entradas	Tiempo Milisegundos
50 registros	50	0.01
150 registros	150	2
500 registros	500	3
1600 registros	1600	5
2999 registros	2999	9
5200 registros	5200	11

Ilustración 3

Ilustración 4

Entorno de pruebas

Procesador: Amd E1-2500 1.40Ghz.

Memoria: 4GB Ddr3.

Disco duro: 500 Gb 5400 rpm.

Sistema Operativo: Windows 10 Pro 64bit.

Análisis Tercer Punto

Para solucionar el problema 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 almacenada en el vector de tipo Rodal "MayorNumerodeArbolesTemporales".

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal "MayorNumerodeArbolesTemporales" se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles y esta se compara con la variable "MayorNumerodeArbolesTemporal" (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que no supere el valor de "CostoTotal" se verifica si "tmpNumerodeArboles" es mayor a "MayorNumerodeArbolesTemporal", en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal "MayornumerodeArboles", de lo contrario se queda con la ya almacenada anteriormente.

Dado que se puede dar el caso que las mejores soluciones están en los extremos y hasta el punto anterior estas no son tomadas dentro de las opciones(esto debido a que el total de beneficios ya esta próximo a alcanzar el umbral o ya llego a el) es necesario crear una copia del vector original que contiene los rodales y utilizar de nuevo el MergeSort para realizar un ordenamiento esta vez no por la fecha de finalización, sino por el numero de arboles que contiene cada rodal(orden descendente), con lo anterior utilizamos de nuevo los algoritmos MaximoNumeroRodales y MaximoNumeroRodalesReverse y 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 con la función llamada reverse 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 almacenada en el vector de tipo Rodal "MayorNumerodeArbolesTemporales".

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal "MayorNumerodeArbolesTemporales" se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles y esta se compara con la variable "MayorNumerodeArbolesTemporal" (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que no supere el valor de "CostoTotal" se verifica si "tmpNumerodeArboles" es mayor a "MayorNumerodeArbolesTemporal", en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal "MayornumerodeArboles", de lo contrario se queda con la ya almacenada anteriormente.

Por ultimo al haber obtenido la mejor solución se pasa a guardar en un archivo plano la lista que este en el vector de tipo Rodal “MayornumerodeArboles” (teniendo en cuenta que el objeto contiene el numero del rodal, numero de arboles, de empleados, fechas y costo).

Pasos utilizados:

- Ordenamiento con MergeSort.(Por fecha de finalización del rodal) → **O(n.lgn)**
- For(recorrer arreglo de elementos) →Complejidad total **O(n^2)**
    - funcionalidades.MaximoNumeroRodales → **O(n)**
    - funcionalidades.MaximoNumeroRodalesReverse → **O(n)**
- Comparación para saber si no el supera el valor de “CostoTotal”.
- Comparación para saber si es el mayor elemento.
- Comparación() → **O(n)**
- Ordenamiento con MergeSort.(Por numero de arboles que tiene el rodal) → **O(n.lgn)**
- For(recorrer arreglo de elementos) →Complejidad total **O(n^2)**
    - funcionalidades.MaximoNumeroRodales → **O(n)**
    - funcionalidades.MaximoNumeroRodalesReverse → **O(n)**
- Comparación para saber si no el supera el valor de “CostoTotal”.
- Comparación para saber si es el mayor elemento.
- Comparación() → **O(n)**

De esta forma determino que la complejidad del algoritmo es de **O(n^2)**

La complejidad de esta solución

Primeramente es determinado por la complejidad del algoritmo de ordenamiento MergeSort la cual es de **O(n.lgn).** En el segundo paso se le suma 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** (desde la posición en la que esta hacia delante, y en sentido contrario) se convierte en **O(n^2).** De esta forma determina que nuestra solución tiene una complejidad total de **O(n^2).**

PSEUDOCODIGO

Algoritmo Dinamico

```
ArrayList<Rodales> MergeOut = ordenamiento.mergeSort();
```

```
ArrayList<Rodales> MergeOut = ordenamieto.mergeSort(c);
funcionalidades.MaximoNumeroRodales(MergeOut);
```

Funciones establecidas para la llegar a la solución optima

```
ArrayList<Rodales> MaximoNumeroRodales(int inicio) {
    Punto ←inicio;

    inicial_Rodal ← entrada.get(punto);
    Solucion.add(inicial_Rodal);
    //Almaceno el valor de numero de arboles par saber quien es mayor
    MayorNumeroArbolesTemporal.add(inicial_Rodal);
    tmpNumeroArboles += inicial_Rodal.numero_arboles;


    //Este algoritmo resuelvo el problema del máximo numero de rodales permitido
    for (int i = punto to entrada.size()) {
        Date date1 ← Solucion.get(Solucion.size() - 1).fecha_fin;
        Date date2 ← entrada.get(i).fecha_inicio;

        complejidad +=1;

        if (!date2.before(date1)) {
            Solucion.add(entrada.get(i));
            tmpNumeroArboles += entrada.get(i).numero_arboles;
            MayorNumeroArbolesTemporal.add(entrada.get(i));
        }
    }
    return Solucion;
}
```

/*Complejidad O(n)
Esta función separa en un punto del arreglo que identifique con la variable de inicio y compare si no se solapa con todos los elementos anteriores

a el*/

//Complejidad O(1)
//Realiza las comparación para saber si esta operación obtuvo la mayor ganancia en arboles
//Comparo el valor que ya existe y el nuevo si el nuevo es mayor almaceno el nuevo. de lo contrario lo ignoro

```
Probar() {
    if (tmpNumeroArboles > maximoNumeroArboles) {
        maximoNumeroArboles ← tmpNumeroArboles;

        MayorNumeroArboles ←  MayorNumeroArbolesTemporal;
        maximoNumeroArboles);
    }
    tmpNumeroArboles ← 0;
    MayorNumeroArbolesTemporal.clear();
}
```

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.

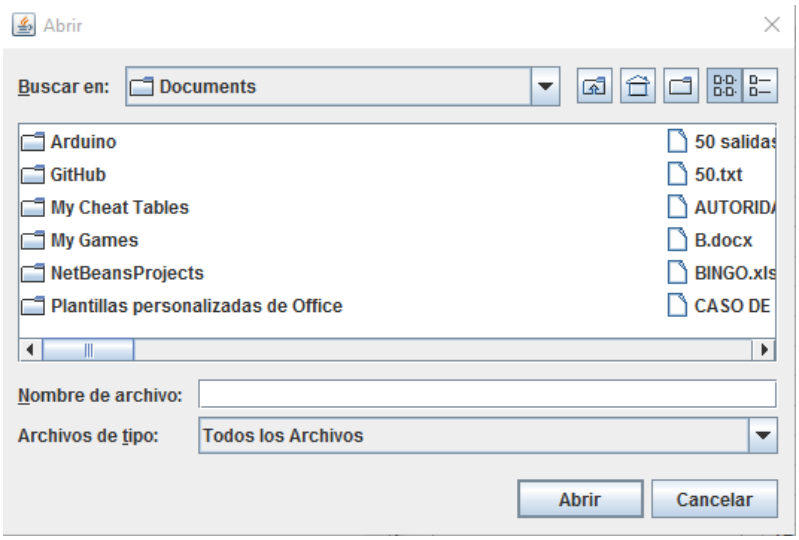

Ilustración 9

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

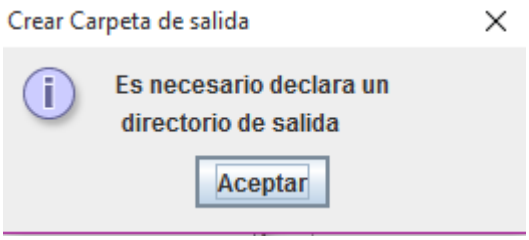

Ilustración 10

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

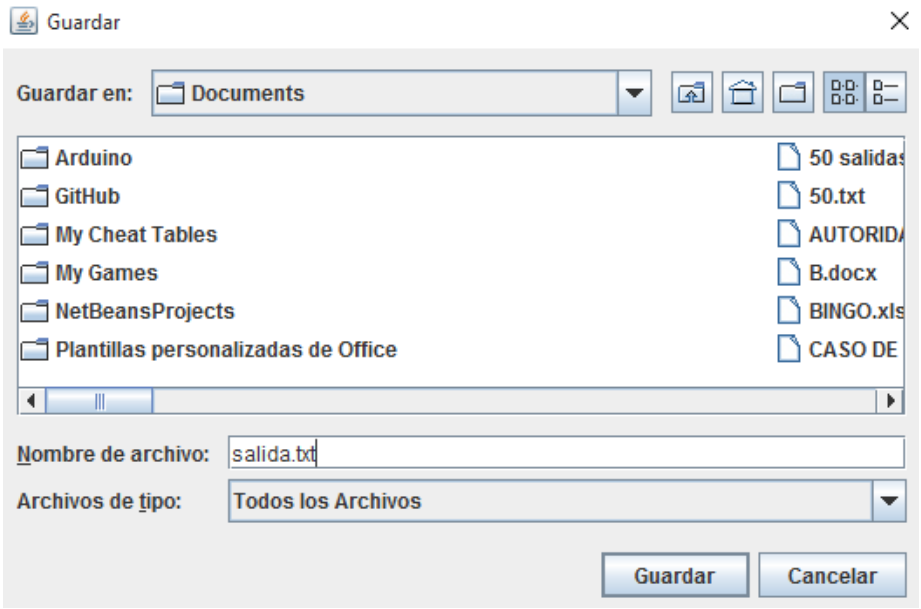

Ilustración 11

Al finalizar todos los procedimientos Retorna un mensaje de finalización.

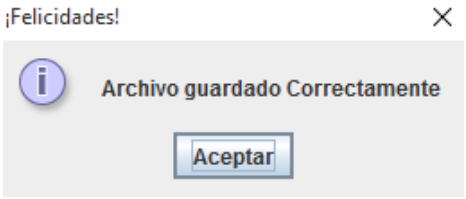

Ilustración 12

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	5	3423
150 registros	150	9	25123
500 registros	500	26	258573
1600 registros	1600	128	2587273
2999 registros	2999	328	…….
5200 registros	5200	564	……..

Ilustración 5

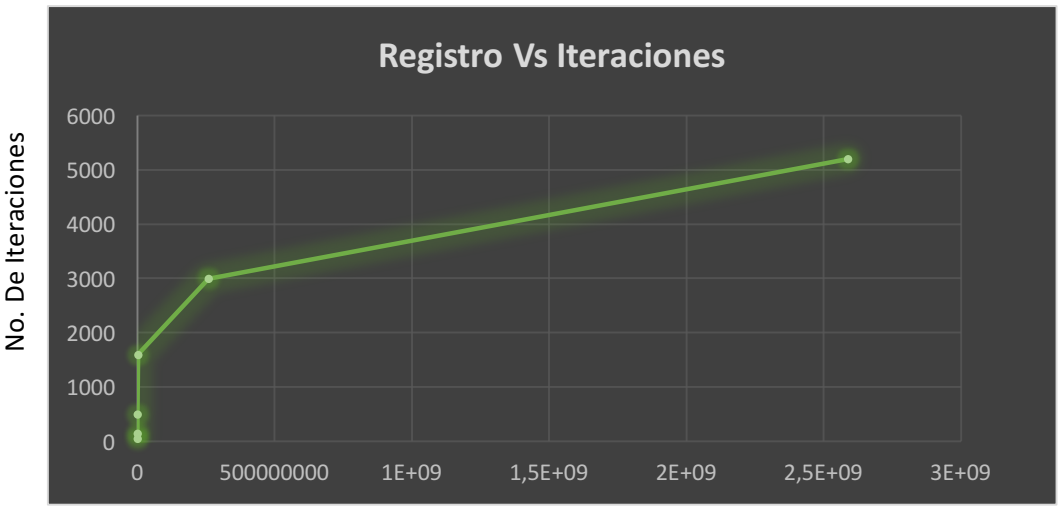

Ilustración 6

Entorno de pruebas

Procesador: Intel Core I5 4ta Generación.

Memoria: 4GB Ddr3.

Disco duro: 128GB SSD.

Sistema Operativo: OS X El Capitan.

Análisis Cuarto Punto

Para solucionar el cuarto 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) verificó 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 almacenada en el vector de tipo Rodal "MayorNumerodeArbolesTemporales".

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal "MayorNumerodeArbolesTemporales" se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles, se valida que no supere el umbral de arboles que se pueden talar, si esta restricción se cumple entonces se compara con la variable "MayorNumerodeArbolesTemporal" (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que "tmpNumerodeArboles" sea mayor a "MayorNumerodeArbolesTemporal", en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal "MayornumerodeArboles", de lo contrario se queda con la ya almacenada anteriormente.

Dado que se puede dar el caso que las mejores soluciones están en los extremos y hasta el punto anterior estas no son tomadas dentro de las opciones(esto debido a que el total de beneficios ya esta próximo a alcanzar el umbral o ya llego a el) es necesario crear una copia del vector original que contiene los rodales y utilizar de nuevo el merge sort para realizar un ordenamiento esta vez no por la fecha de finalización, sino por el numero de arboles que contiene cada rodal(orden descendente), con lo anterior utilizamos de nuevo los algoritmos MaximoNumeroRodales y MaximoNumeroRodalesReverse y 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 con la función llamada reverse 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 almacenada en el vector de tipo Rodal "MayorNumerodeArbolesTemporales".

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal "MayorNumerodeArbolesTemporales" se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles, se valida que no supere el umbral de arboles que se pueden talar, si esta restricción se cumple entonces se compara con la variable "MayorNumerodeArbolesTemporal" (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que "tmpNumerodeArboles" sea mayor a "MayorNumerodeArbolesTemporal", en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal "MayornumerodeArboles", de lo contrario se queda con la ya almacenada anteriormente.

Por último al haber obtenido la mejor solución se pasa a guardar en un archivo plano la lista que esté en el vector de tipo Rodal "MayornumerodeArboles" (teniendo en cuenta que el objeto contiene el numero del rodal, número de arboles, de empleados y las fechas).

Los pasos que utilice para resolverlos

- Ordenamiento con MergeSort. → **O(nlgn)**
- For(recorrer arreglo de elementos) →Complejidad total **$O(n^2)$**
    - funcionalidades.MaximoNumeroRodales → **O(n)**
    - funcionalidades.MaximoNumeroRodalesReverse → **O(n)**
- comparación para saber si es el mayor elemento.
- Comparación() → **O(n)**

De esta forma determinó que la complejidad del algoritmo es de **$O(n^2)$**

La complejidad de esta solución

Es determinado por la complejidad del algoritmo de ordenamiento MergeSort la cual es de **O(nlgn).** Sumado a la complejidad de recorrer los elementos previamente organizados para su selección que obtenemos una complejidad de **O(n)** encada selección pero al realizarlo en un ciclo **for** se convierte en **$O(n^2)$** elemento de esta forma determina que nuestra solución tiene una complejidad total de **$O(n^2)$.**

PSEUDOCODIGO

```
//Ordeno el arreglo con el algoritmo mergeSort Complejidad  O(nlogn)

ArrayList<Rodal> MergeOut  ordenamiento.mergeSort(c, 0);

ArrayList<Rodal> MergeOut_arboles  ordenamiento.mergeSort(c, 1);

for (int i = 0 to MergeOut.size()) {
        /*Casos Base*/
        funcionalidades.entrada  MergeOut;
        funcionalidades.Solucion.clear();
        funcionalidades.MaximoNumeroRodales(i);
        funcionalidades.MaximoNumeroRodalesReverse(i);
        /*Posibilidades alternas*/
        funcionalidades.entrada  MergeOut_arboles;
        funcionalidades.Solucion.clear();
        funcionalidades.MaximoNumeroRodales(i);
        funcionalidades.MaximoNumeroRodalesReverse(i);
        /*Probar el mayor beneficio*/
        funcionalidades.Probar();
  }
```

Funciones establecidas para la llegar a la solución óptima

```
  ArrayList<Rodales> MaximoNumeroRodales(int inicio) {
     Punto ← inicio;
       inicial_Rodal ←  entrada.get(punto);
     Solucion.add(inicial_Rodal);
     //Almaceno el valor de numero de arboles par saber quien es
```

```
mayor
    MayorNumeroArbolesTemporal.add(inicial_Rodal);
    tmpNumeroArboles += inicial_Rodal.numero_arboles;
    //Este algoritmo resuelve el problema del máximo número de rodales
permitido
    for (int i = punto to entrada.size()) {
      Date date1 ← Solucion.get(Solucion.size() - 1).fecha_fin;
      Date date2 ←  entrada.get(i).fecha_inicio;

      complejidad +=1;



      if (!date2.before(date1)) {
        Solucion.add(entrada.get(i));
        tmpNumeroArboles += entrada.get(i).numero_arboles;
        MayorNumeroArbolesTemporal.add(entrada.get(i));
      }
    }
    return Solucion;
  }

  /*Complejidad O(n)
  Esta función separa en un punto del arreglo que identifique con la variable de inicio y compare si
no se solapa con todos los elementos anteriores a él*/

  ArrayList<Rodales> MaximoNumeroRodalesReverse(int inicio) {
    int punto ← inicio - 1;
    ArrayList<Rodales> salida;
    //entrada.
    //Este algoritmo resuelve el problema del máximo número de rodales permitido
    int contador ← 1;
    for (int i = punto to 0) {
```

```
        Date date1 ←  entrada.get(entrada.size() - contador).fecha_inicio;
        Date date2 ←  entrada.get(i).fecha_fin;

        if (date2.before(date1)) {
           salida.add(entrada.get(i));
           tmpNumeroArboles += entrada.get(i).numero_arboles;
           if (!MayorNumeroArbolesTemporal.contains(entrada.get(i))) {
              MayorNumeroArbolesTemporal.add(entrada.get(i));
           }
           contador++;
              }
     }
     return salida;
  }
//Complejidad O(1)
//Realiza las comparación para saber si esta operacion obtuvo la mayor ganancia en arboles
//Comparó el valor que ya existe y el nuevo si el nuevo es mayor almaceno el nuevo. de lo
contrario lo ignoro
  Probar() {
     if (tmpNumeroArboles > maximoNumeroArboles) {
        maximoNumeroArboles ←  tmpNumeroArboles;
        MayorNumeroArboles ←   MayorNumeroArbolesTemporal;
        maximoNumeroArboles);
     }
     MayorNumeroArbolesTemporal.clear();
  }

  Agregar(Rodal entrada) {
     tmpcosto += entrada.numero_arboles;
     if (tmpcosto <= umbral) {
        mayorNumeroArbolesTemporal.add(entrada);
```

```
    } else {
        tmpcosto -= entrada.numero_arboles;
    }
}
```

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.

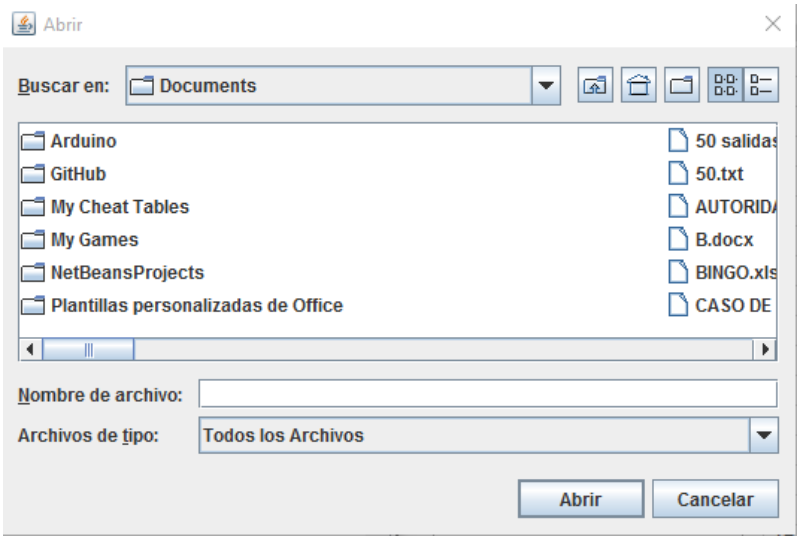

Ilustración 1

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

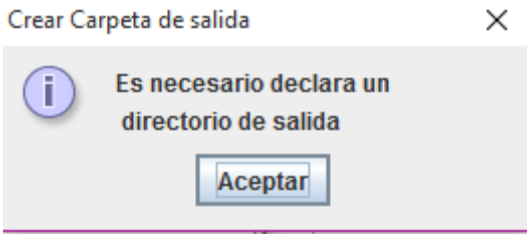

Ilustración 2

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

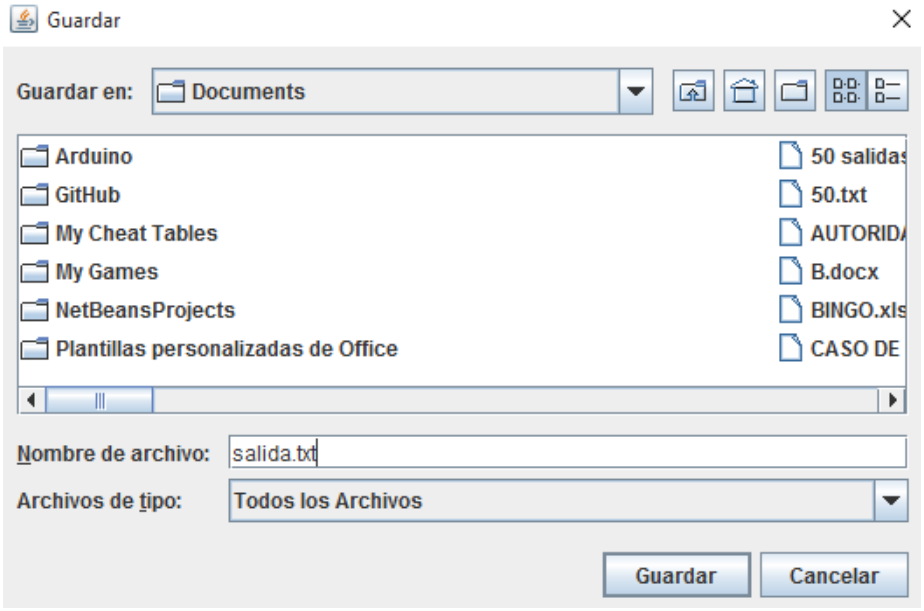

Ilustración 3

Al finalizar todos los procedimientos Retorna un mensaje de finalización.

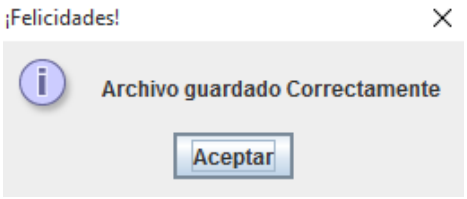

Ilustración 4

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 registro de entradas2	Numero de Iteraciones
50 registros	50	5	50	3423
150 registros	150	12	150	25123
500 registros	500	33	500	258573
1600 registros	1600	146	1600	2587273
2999 registros	2999	223	2999	9045056,25
5200 registros	5200	476	5200	27128472

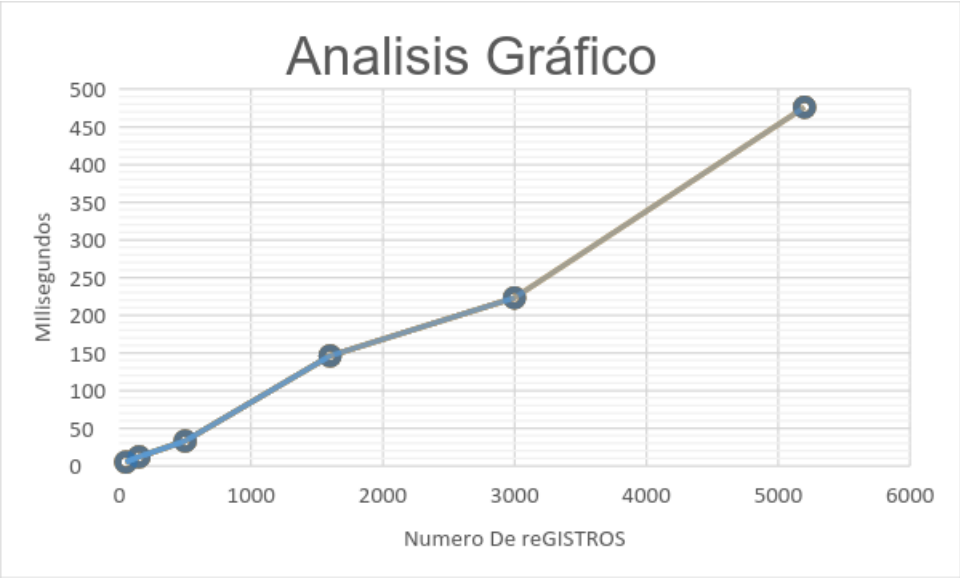

Ilustración 7

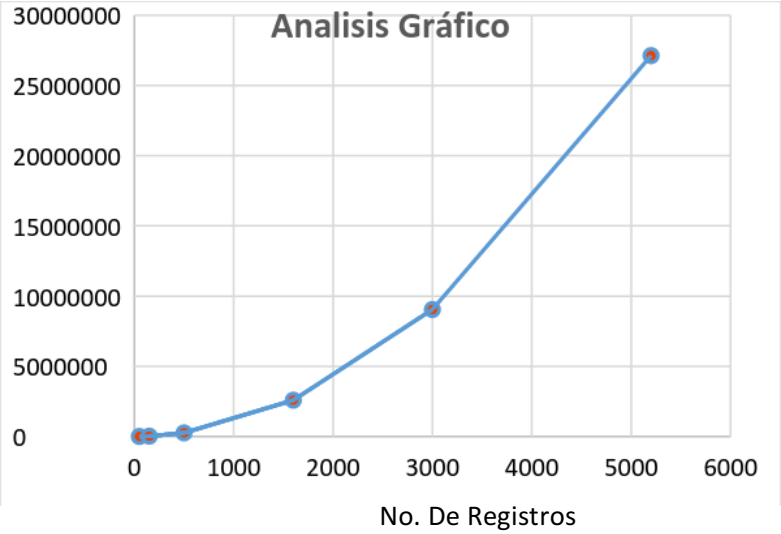

Ilustración 8

Entorno de pruebas

Procesador: Intel Core i7-3610 2,30Ghz.

Memoria: 7,7GB Ddr3.

Disco duro: 103,1 Gb 5400 rpm.

Sistema Operativo: Linux Mint 17,2 64bit.

Análisis Quinto Punto

Para solucionar el quinto 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 almacenada en el vector de tipo Rodal "MayorNumerodeArbolesTemporales".

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal "MayorNumerodeArbolesTemporales" se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles, además seo obtiene el total de personas que serán empleadas y se almacena en la variable tmpNumeroEmpleados, luego se valida que no supere el umbral de arboles que se pueden talar y la cantidad de empleados este por encima del umbral mínimo, si se cumplen estas restricciones entonces se compara con la variable "MayorNumerodeArbolesTemporal" (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que "tmpNumerodeArboles" sea mayor a "MayorNumerodeArbolesTemporal", en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal "MayornumerodeArboles", de lo contrario se queda con la ya almacenada anteriormente.

Dado que se puede dar el caso que las mejores soluciones están en los extremos y hasta el punto anterior estas no son tomadas dentro de las opciones(esto debido a que el total de beneficios ya esta próximo a alcanzar el umbral o ya llego a el) es necesario crear una copia del vector original que contiene los rodales y utilizar de nuevo el merge sort para realizar un ordenamiento esta vez no por la fecha de finalización, sino por el numero de arboles que contiene cada rodal(orden descendente), con lo anterior utilizamos de nuevo los algoritmos MaximoNumeroRodales y MaximoNumeroRodalesReverse y 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 con la función llamada reverse 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 almacenada en el vector de tipo Rodal "MayorNumerodeArbolesTemporales".

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal "MayorNumerodeArbolesTemporales" se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles, además seo obtiene el total de personas que serán empleadas y se almacena en la variable tmpNumeroEmpleados, luego se valida que no supere el umbral de arboles que se pueden talar y la cantidad de empleados este por encima del umbral mínimo, si se cumplen estas restricciones entonces se compara con la variable "MayorNumerodeArbolesTemporal" (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que "tmpNumerodeArboles" sea mayor a "MayorNumerodeArbolesTemporal", en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal "MayornumerodeArboles", de lo contrario se queda con la ya almacenada anteriormente.

Por ultimo al haber obtenido la mejor solución se pasa a guardar en un archivo plano la lista que este en el vector de tipo Rodal "MayornumerodeArboles" (teniendo en cuenta que el objeto contiene el numero del rodal, numero de arboles, de empleados y las fechas).

Los pasos que utilice para resolverlos

- Ordenamiento con MergeSort. → O(n.lgn)

- For(recorrer arreglo de elementos) → Complejidad total O(n2)

o funcionalidades.MaximoNumeroRodales → O(n)

o 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(nlgn).** Sumado a la complejidad de recorrer los elementos previamente organizados para su selección que obtenemos una complejidad de **O(n)** encada selección pero al realizarlo en un ciclo **for** se convierte en **O(n^2)** elemento de esta forma determina que nuestra solución tiene una complejidad total de **O(n^2).**

PSEUDOCODIGO

```
//Ordeno el arreglo con el algoritmo mergeSort Complejidad  O(nlogn)
ArrayList<Rodal> MergeOut ←  ordenamieto.mergeSort(c, 0);
ArrayList<Rodal> MergeOut_arboles ←  ordenamieto.mergeSort(c, 1);

for (int i = 0 to MergeOut.size()) {
        /*Casos Base*/
        funcionalidades.entrada ←  MergeOut;
        funcionalidades.Solucion.clear();
        funcionalidades.MaximoNumeroRodales(i);
        funcionalidades.MaximoNumeroRodalesReverse(i);

          /*Posibilidades alternas*/
        funcionalidades.entrada ←  MergeOut_arboles;
        funcionalidades.Solucion.clear();
        funcionalidades.MaximoNumeroRodales(i);
```

```
            funcionalidades.MaximoNumeroRodalesReverse(i);

            /*Probar el mayor beneficio*/
            funcionalidades.Probar();
    }
```

Funciones establecidas para la llegar a la solución óptima

```
  ArrayList<Rodales> MaximoNumeroRodales(int inicio) {

      Punto ← inicio;
      inicial_Rodal ←  entrada.get(punto);
      Solucion.add(inicial_Rodal);

      //Almaceno el valor de numero de arboles par saber quien es mayor

      MayorNumeroArbolesTemporal.add(inicial_Rodal);
      tmpNumeroArboles += inicial_Rodal.numero_arboles;

      //Este algoritmo resuelve el problema del máximo numero de rodales
permitido

      for (int i = punto to entrada.size()) {

         Date date1 ←  Solucion.get(Solucion.size() - 1).fecha_fin;

         Date date2 ←  entrada.get(i).fecha_inicio;

         complejidad +=1;

         if (!date2.before(date1)) {
            Solucion.add(entrada.get(i));
            tmpNumeroArboles += entrada.get(i).numero_arboles;
            MayorNumeroArbolesTemporal.add(entrada.get(i));
```

```
        }
      }
      return Solucion;
    }

    /*Complejidad O(n)

    Esta función separa en un punto del arreglo que identifique con la
    variable de inicio y compare si no se solapa con todos los elementos
    anteriores a el*/

    ArrayList<Rodales> MaximoNumeroRodalesReverse(int inicio) {

      int punto ← inicio - 1;
      ArrayList<Rodales> salida;
      //entrada.
      //Este algoritmo resuelve el problema del máximo numero de rodales
permitido

      int contador ← 1;

      for (int i = punto to 0) {
        Date date1 ← entrada.get(entrada.size() - contador).fecha_inicio;
        Date date2 ← entrada.get(i).fecha_fin;

        if (date2.before(date1)) {
          salida.add(entrada.get(i));
          tmpNumeroArboles += entrada.get(i).numero_arboles;
          if (!MayorNumeroArbolesTemporal.contains(entrada.get(i))) {
            MayorNumeroArbolesTemporal.add(entrada.get(i));
          }
          contador++;
```

```
            }
        }
        return salida;
    }

//Complejidad O(1)
//Realiza las comparación para saber si esta operación obtuvo la mayor
ganancia en arboles
//Comparo el valor que ya existe y el nuevo si el nuevo es mayor
almaceno el nuevo. de lo contrario lo ignoro

    Probar() {
        if (tmpNumeroArboles > maximoNumeroArboles) {
            maximoNumeroArboles ← tmpNumeroArboles;

            MayorNumeroArboles ← MayorNumeroArbolesTemporal;
            maximoNumeroArboles);
        }
tmpNumeroArboles ← 0;
tmpcosto ← 0;
        MayorNumeroArbolesTemporal.clear();
    }

    Agregar(Rodal entrada) {
        tmpcosto += entrada.numero_arboles;
        if (tmpcosto <= umbral) {
            mayorNumeroArbolesTemporal.add(entrada);
        } else {
            tmpcosto -= entrada.numero_arboles;
        }
}
```

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.

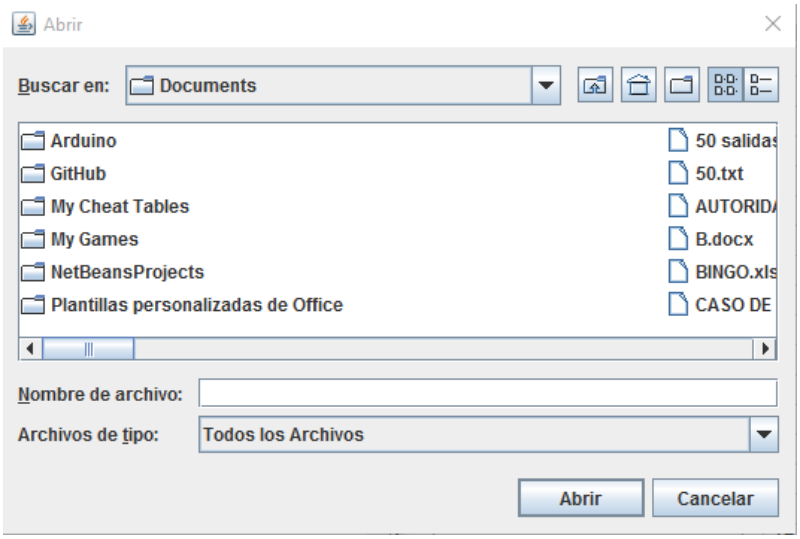

Ilustración 1

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

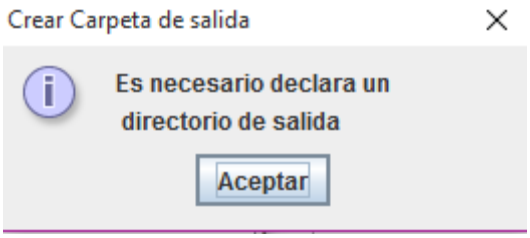

Ilustración 2

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

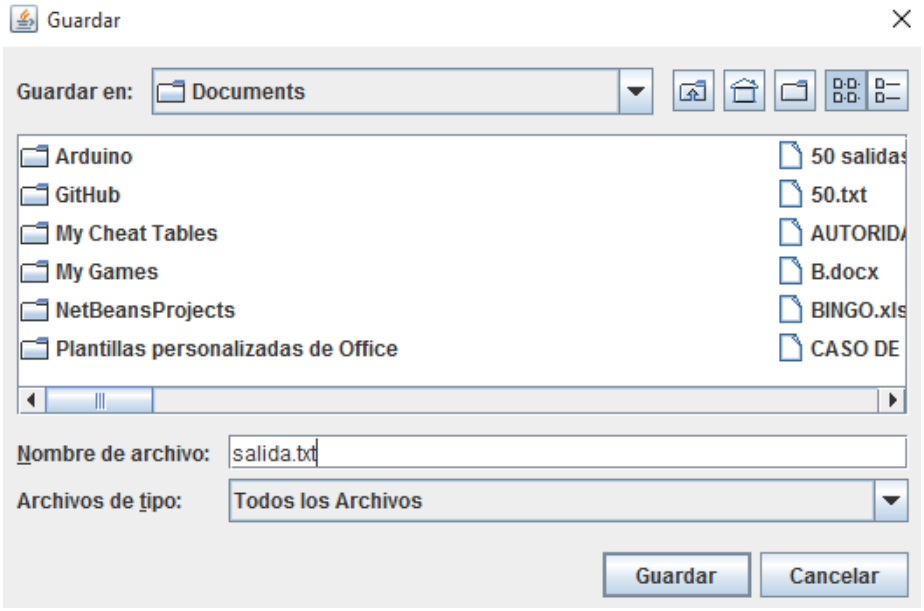

Ilustración 3

Al finalizar todos los procedimientos Retorna un mensaje de finalización.

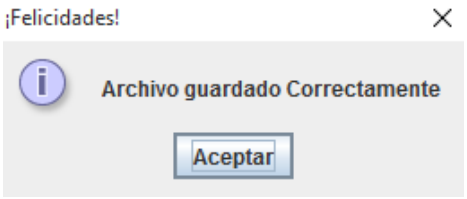

Ilustración 4

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 registro de entradas2	Numero de Iteraciones
50 registros	50	3	50	3423
150 registros	150	9	150	25123
500 registros	500	27	500	258573
1600 registros	1600	94	1600	2587273
2999 registros	2999	214	2999	9045056,25
5200 registros	5200	716	5200	27128472

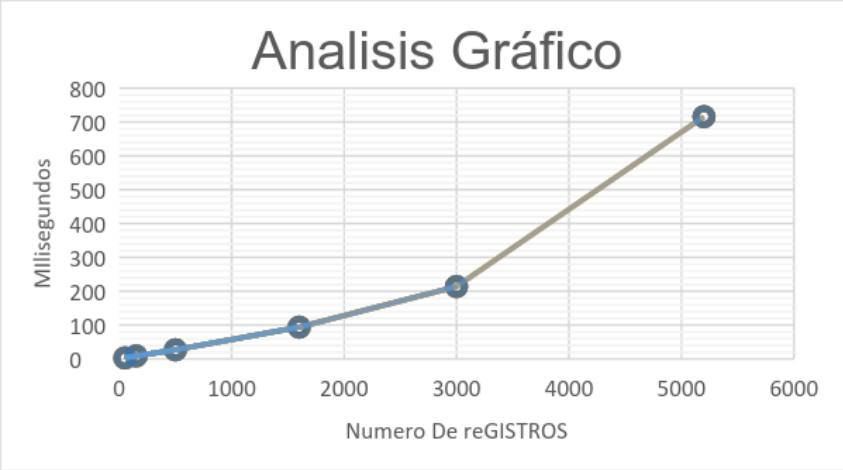

Ilustración 9

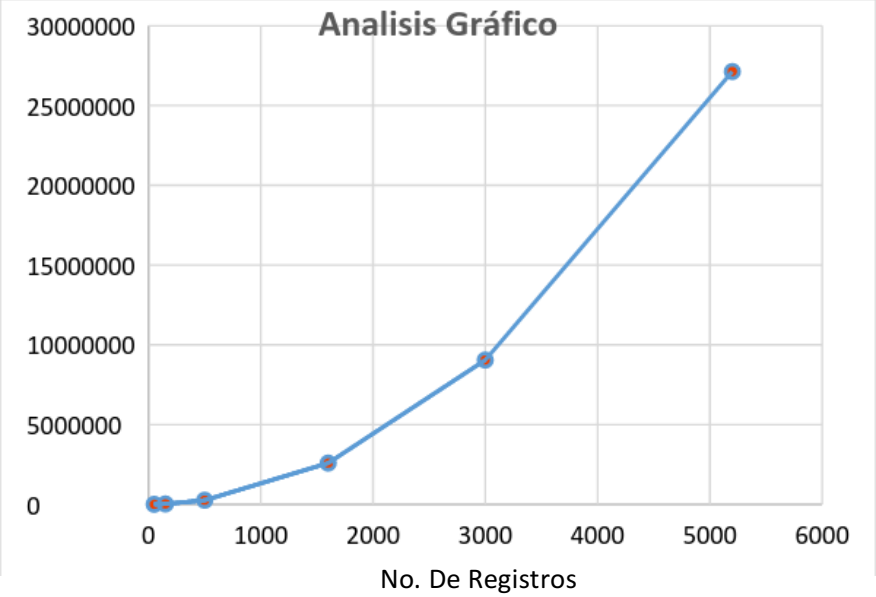

Ilustración 10

Entorno de pruebas

Procesador: Intel Core i7-3610 2,30Ghz.

Memoria: 7,7GB Ddr3.

Disco duro: 103,1 Gb 5400 rpm.

Sistema Operativo: Linux Mint 17,2 64bit.

Conclusiones

- Como podemos observar aunque la complejidad en cualquiera de las pruebas realizadas(Excepto el punto 2) es n^2 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.
- Cuando desarrollamos algoritmos óptimos, podemos observar que se usan menor cantidad de recursos de hardware, para ejecutar tareas con entradas de tamaño significativo lo que permitiría ser usado en una mayor cantidad de equipos.
- El desarrollo de este proyecto nos permitió adquirir habilidades que nos permitirán encontrar la mejor solución computacionalmente posible ante una serie de restricciones.