Introducción al Problema

El problema a resolver será ordenar un archivo de jugadores de la NBA según sus medias de puntuaciones en cada temporada que han jugado y teniendo en cuenta también el porcentaje de aciertos que tuvieron (FG%), ya que no queremos jugadores que tengan muchos puntos pero tengan una baja tasa de aciertos. Añadir que este archivo deberá estar ordenado, al menos, por nombre de jugador. Por tanto, un jugador deberá tener todas sus apariciones seguidas, de lo contrario se considerará que es un nuevo jugador.

Solución al Problema

El problema será solucionado de forma recursiva usando un algoritmo de ordenación basado en MergeSort ya que es menos tedioso, aunque también se podría haber basado en QuickSort. En resumidas cuentas, nuestro programa cargará el archivo en un ArrayList de la clase Players (proporcionada por el repositorio) a la que hemos aplicado una serie de cambios:

-Se ha implementado un método add para añadir a cada jugador valores como un nuevo equipo, posición y puntos.
-Hemos implementado el método toString para ajustar la forma en que se convertirá en cadena de caracteres.

Nuestro programa hará uso del algoritmo de ordenación implementado para conseguir extraer del archivo quiénes son los mejores 10 jugadores de todos los tiempos. Este algoritmo aplicará Divide y Vencerás, dividiendo así el problema en partes más pequeñas haciéndolo así más sencillo para ordenar, tal y como funcionaría MergeSort:
- Si la posición de inicio del ArrayList es la misma que la final entonces ya está ordenado.
- Si no, el ArrayList a ordenar se divide en dos mitades de tamaño similar. Cada mitad se ordena de forma recursiva aplicando el método MergeSort. A continuación las dos mitades ya ordenadas se mezclan formando una secuencia ordenada. Añadiendo una mejora para este caso, que sería la finalización del algoritmo al encontrar los 10 mejores resultados, que en este caso serían los 10 mejores jugadores.

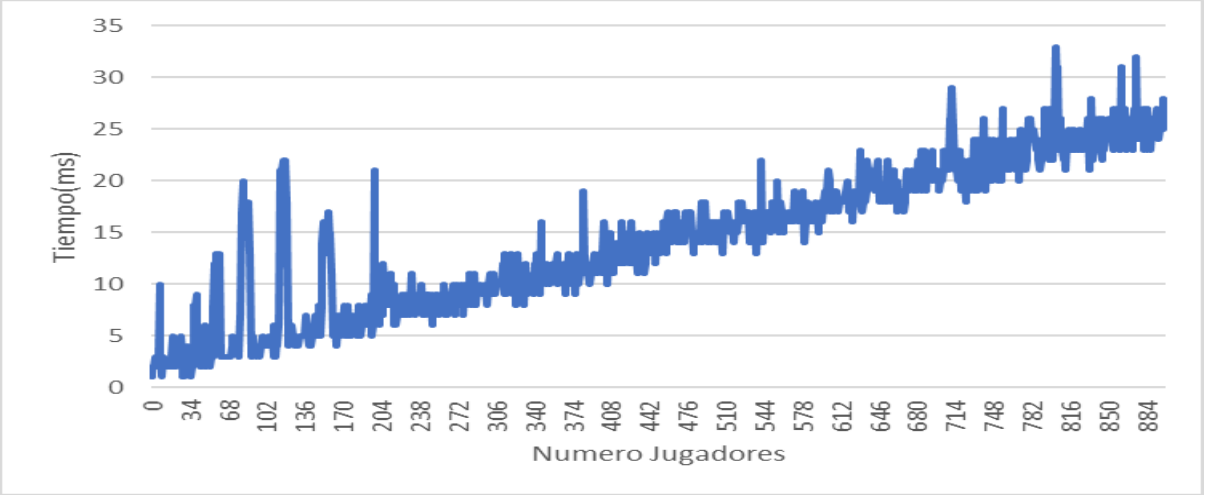
```java
private static ArrayList<Player> mejoresJugadores(int principio, int fin) {
    ArrayList<Player> mejoresJugadores = new ArrayList<Player>(numJugadores);
    if (principio == fin) {
        mejoresJugadores.add(nba.get(principio));
    } else {
        int mitad = (principio + fin) / 2;
        ArrayList<Player> jugadores1 = mejoresJugadores(principio, mitad);
        ArrayList<Player> jugadores2 = mejoresJugadores(mitad + 1, fin);

        int i = 0;
        int j = 0;
        while (mejoresJugadores.size() < numJugadores && i <= jugadores1.size() - 1 && j <= jugadores2.size() - 1) {
            if (jugadores1.get(i).getScore() > jugadores2.get(j).getScore()) {
                mejoresJugadores.add(jugadores1.get(i));
                i++;
            } else {
                mejoresJugadores.add(jugadores2.get(j));
                j++;
            }
        }
        while (mejoresJugadores.size() < numJugadores && i <= jugadores1.size() - 1) {
            mejoresJugadores.add(jugadores1.get(i));
            i++;
        }
        while (mejoresJugadores.size() < numJugadores && j <= jugadores2.size() - 1) {
            mejoresJugadores.add(jugadores2.get(j));
            j++;
        }
    }
    return mejoresJugadores;
}
```

Algorítmicamente el caso base de nuestro algoritmo tiene O(1), lo que sería el coste no recursivo. Mientras que el coste recursivo sería de O(n), ya que nuestro algoritmo se ejecutaría n (para este problema en concreto 10) veces recursivamente. Por tanto:

-caso base:
t(n)=O(1)
n^(k1)=n^0=1
k^1=0

-algoritmo:
t(n)= at(n/b)+g(n)=2t(n/2)+O(n)
n^(k2)=n^1=n
k=max(k1,k2)=1
como a=b^k, entonces el tiempo de ejecución de nuestro algoritmo será de (n^k)*logn=nlogn, mejorando así el tiempo $\Theta(n)$ del MergeSort.



Esta sería la gráfica del tiempo que tardaría este algoritmo respecto al número de jugadores a ordenar. Es interesante comentar que junto al programa también se desarrolló un generador de archivos de jugadores como juego de pruebas para analizar mejor el algoritmo. Por tanto con un potente procesador podríamos conseguir una gráfica más similar a la ideal de nlogn.

Realizado por: Alejandro Miguel Ruiz Baños