Estructura de Datos y Algoritmos II

Memoria 1
Divide y Vencerás

Alejandro Dámaso Pastor

Francisco Escamilla Vizcaino

ÍNDICE

INTRODUCCIÓN

En la siguiente práctica se pretende que obtengamos los 10 mejores jugadores de NBA de todos los tiempos a partir de un conjunto de datos que se nos entrega para la realización de la práctica.

Para ello lo que haremos será usar el método divide y vencerás y a continuación explicaremos como lo hemos hecho y cómo funciona nuestro código.

ANÁLISIS DEL CÓDIGO

En esta sección analizaremos las funcionalidades del código, explicando cada uno de los métodos de forma independiente.

MÉTODO SORT

Primeramente tenemos el método sort que consiste básicamente en que comprueba que el array que le pasamos con la lista de jugadores tenga datos y no esté vacío para después coger y crear un nuevo array de jugadores, para después ordenarlo y con .clear limpiarlo y añadir al final el array bueno con .addAll .

```
public static void sort(ArrayList<Player> playerList) {
     if (playerList.isEmpty()) {
               System.out.println("Array vacio");
     }

     Player[] playerArray = new Player[playerList.size()];
     playerList.toArray(playerArray);

     sort(playerArray, 0, playerArray.length - 1);

     playerList.clear();
     playerList.addAll(Arrays.asList(playerArray));
}
```

MÉTODO SORT SOBRECARGADO

En el siguiente método lo que hacemos principalmente es comparar los valores “Left” y ”Right” para que siempre que el índice de el lado izquierdo sea menor Que el derecho creamos la variable de “middle” que es el valor central de nuestro array final a partir de la operación (left +(right - left) / 2) y esos serán nuestros índices.
después con el método sort ordenaremos los arrais de los jugadores de la izquierda al centro y el array del centro a la derecha, para después usar el método merge y unir los métodos ordenándolos también.

```
public static void sort(Player[] player, int left, int right)
{
     if (left < right) {
          int middle = left + (right - left) / 2;

          // ordena la primera y segunda mitad.
          sort(player, left, middle);
          sort(player, middle + 1, right);

          // une las dos en una sola ordenandolas.
          merge(player, left, middle, right);
     }
}
```

MÉTODO MERGE

En este método lo que hacemos es definir los índices de los arrays izquierdo y derecho para después de tenerlos definimos con el “While” los ordenamos diciendo básicamente que si el valor que hay dentro de la celda con el índice i es menor a la celda con el índice que esté comparando del otro array meta este y viceversa, para que luego vuelva a hacer lo mismo en un bucle hasta introducir todos los valores de los arrays ordenados de menor a mayor.

```
private static void merge(Player[] player, int left, int middle, int right) {
        int nL = middle - left + 1;
        int nR = right - middle;

        Player[] leftArray = new Player[nL];
        Player[] rightArray = new Player[nR];

        // Copiar player[] a cada lado de los arrays.
        for (int i = 0; i < nL; ++i) {
                leftArray[i] = new Player(player[left + i]);
        }
        for (int i = 0; i < nR; ++i) {
                rightArray[i] = new Player(player[middle + 1 + i]);
        }

        int index = left;
        int leftIndex = 0;
        int rightIndex = 0;


        while (leftIndex < nL && rightIndex < nR) {
                if (leftArray[leftIndex].getScore() >  rightArray[rightIndex].getScore()) {
                       player[index] = new Player(leftArray[leftIndex]);
                       leftIndex++;
                } else {
                        player[index] = new Player(rightArray[rightIndex]);
                        rightIndex++;
                        }
                index++;
        }

        while (leftIndex < nL) {
                player[index].SetData(leftArray[leftIndex]);
                leftIndex++;
                index++;
        }

        while (rightIndex < nR) {
                player[index].SetData(rightArray[rightIndex]);
                rightIndex++;
                index++;
        }
}
```

TEOREMA MAESTRO

El teorema maestro nos permite resolver recurrencias de la siguiente forma donde a>0 y b>1.

$$T(n) = aT(n/b) + f(n)$$

Vamos a definir algunas de estas variables basándose en el algoritmo que hemos implementado:

- **n** - El tamaño del problema, en este caso **n** es la longitud de la lista.
- **a** - El número de subproblemas en cada paso recursivo. En el caso de nuestra implementación sería **a = 2** ya que "dividimos" el array en dos secciones.
- **b** - El número por el cual reducimos los subproblemas. Para este caso concreto sea **b = 2** porque pasamos la mitad del array a cada subproblema.
- **f(n)** - El trabajo que se realiza fuera de los pasos recursivos, en este caso unir de forma ordenada ambos arrays.

Se nos quedaría en esta fórmula:

$$T(n) = 2T(n/2) + O(n)$$

Cuyas variables son las siguientes:

a = 2
b = 2
f(n) = O(n)

Realizamos la siguiente comparación:

$$n^{log_b(a)} <=> O(n)$$

$$n^1 = O(n)$$

Esto nos indica que nuestro algoritmo encaja con el segundo caso del Teorema Maestro por el cual:

$$T(n) = O(n * log\, n)$$

Tiempos de Ejecución

tamaño	tiempo
3000	3
10000	2
100000	14
100000	44
200000	57
400000	87
500000	119
600000	641
700000	761
800000	842
900000	1042
1000000	1134