PRÁCTICA 1
EDA II

ALEJANDRO RODRÍGUEZ, ÁNGEL LÓPEZ Y JOSÉ LUIS PÉREZ

- EN ESTA PRÁCTICA SE NOS PLANTEABA OBTENER LA LISTA DE LOS DIEZ MEJORES JUGADORES DE LA HISTORIA DE LA NBA EN BASE A LA MEDIA DE PUNTOS EN SU CARRERA, DÁNDONOS UNA LISTA CON DATOS Y PIDIÉNDONOS UNA SOLUCIÓN QUE USE UN ESQUEMA DIVIDE Y VENCERÁS.

- HEMOS IMPLEMENTADO LOS DOS ALGORITMOS DE ORDENACIÓN QUE CONOCEMOS QUE USAN UN ESQUEMA DIVIDE Y VENCERÁS. ESTOS SON EL QUICKSORT Y EL MERGESORT.
- ESTOS MÉTODOS LOS HEMOS APLICADO A LA LISTA DE JUGADORES FINAL Y HEMOS CALCULADO CUÁNTO TARDA CADA UNO EN FINALIZAR.

QUICKSORT

- IMPLEMENTACIÓN:

```
if (listaJugadores.size() <= 1)
        return listaJugadores;

Player pivote = listaJugadores.get(0); // Tomamos primer elemento como pivote

ArrayList<Player> left = new ArrayList<Player>(); /* Lista izquierda */
ArrayList<Player> right = new ArrayList<Player>(); /* Lista derecha */
ArrayList<Player> sol = new ArrayList<Player>(); /* Array solucion */

for (int i = 1; i < listaJugadores.size(); i++) { /* Bucle que debe recorrer todo el ArrayList */

        // System.out.println("Valor de i " + i); /* Usado para entender cuanto iteraba
        // el metodo*/

        if (listaJugadores.get(i).getScore() <= pivote.getScore()) { // Si el Player es menor o igual que el pivote
                // lo meto a la izquierda
                left.add(listaJugadores.get(i));
        }

        else { // Si el Player es menor que el pivote lo meto a la derecha y viceversa
                right.add(listaJugadores.get(i));

        }
}
sol.addAll(quickSort(left));
sol.add(pivote);
sol.addAll(quickSort(right));

return sol;
```

QUICKSORT

- LA COMPLEJIDAD DE ESTE ALGORITMO ES DE $O(n \operatorname{LOG} n)$ EN EL MEJOR CASO Y DE $O(n^2)$. SE PUEDE DISMINUIR EL IMPACTO DEL PEOR CASO HACIENDO QUE NO SE PUEDAN COGER NI EL PRIMER NI EL ÚLTIMO ELEMENTO.

QUICKSORT TIEMPO

- HACIENDO LAS PRUEBAS PODEMOS VER QUE EL QUICKSORT TARDA UNA MEDIA DE 13 MS.

```
El método diezMejoresQuickSort ha tardado: 14 ms
El método diezMejoresQuickSort ha tardado: 15 ms
El método diezMejoresQuickSort ha tardado: 12 ms
El método diezMejoresQuickSort ha tardado: 22 ms
El método diezMejoresQuickSort ha tardado: 13 ms
El método diezMejoresQuickSort ha tardado: 12 ms
```

MERGESORT

- IMPLEMENTACIÓN:

```
public static ArrayList<Player> diezMejores(ArrayList<Player> listaJugadores) {
		ArrayList<Player> resul = diezMejores(listaJugadores, 0, listaJugadores.size() - 1);
		Collections.reverse(resul);
		return resul;
	}

private static ArrayList<Player> diezMejores(ArrayList<Player> l, int ini, int fin) {
		if (fin - ini <= 10) {
			ArrayList<Player> resul = new ArrayList<Player>();
			for (int i = ini; i <= fin; i++) {
				resul.add(l.get(i));
			}
			Collections.sort(resul);
			return resul;
		}
		int med = (ini + fin) / 2;
		ArrayList<Player> resul1 = diezMejores(l, ini, med);
		ArrayList<Player> resul2 = diezMejores(l, med + 1, fin);
		resul1.addAll(resul2);
		Collections.sort(resul1);
		while (resul1.size() > 10) {
			resul1.remove(0);
		}
		return resul1;
	}
```

MERGESORT

- EN ESTE ALGORITMO, TANTO EL MEJOR COMO EL PEOR CASO TIENEN LA MISMA COMPLEJIDAD: $O(n \log n)$ ENTONCES MERGESORT VA A SER MÁS ESTABLE YA QUE NO DEPENDE DE LA POSICIÓN DEL PIVOTE COMO LO HACE QUICKSORT.

MERGESORT TIEMPO

- MERGESORT TARDA DE MEDIA 3 MS

```
El método diezMejores ha tardado: 2 ms
El método diezMejores ha tardado: 5 ms
El método diezMejores ha tardado: 2 ms
El método diezMejores ha tardado: 3 ms
El método diezMejores ha tardado: 4 ms
El método diezMejores ha tardado: 3 ms
```

- COMO PODEMOS VER, EL MERGESORT SUPERA CON CRECES AL QUICKSORT, AUNQUE SE SUPONE QUE QUICKSORT DEBERÍA SER EL MÁS RÁPIDO, ESTO SE PUEDE DEBER A QUE NO HA COGIDO LOS PIVOTES MÁS ÓPTIMOS.