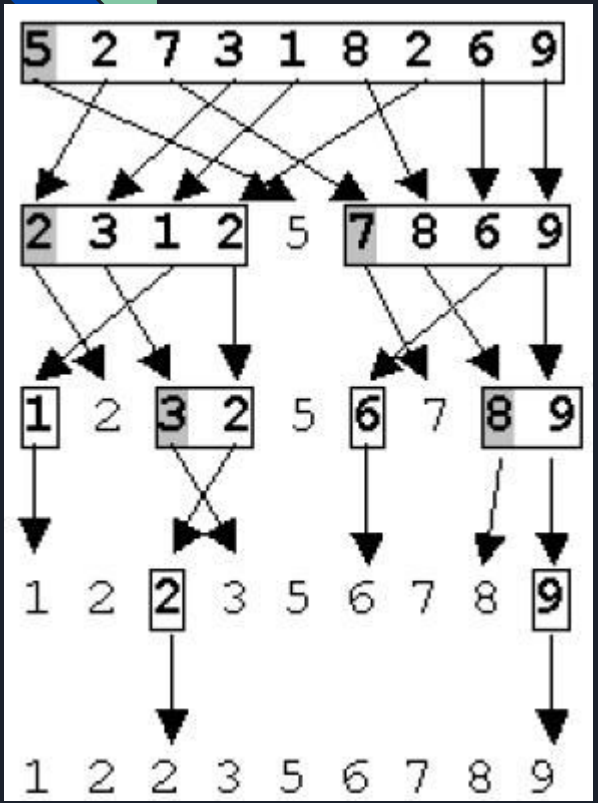
PRACTICA 1: DIVIDE Y VENCERÁS

Manuel Megías Briones
Antonio Aranda Hernandez
Cristina Pérez Ordóñez

Método de ordenación escogido: QuickSort

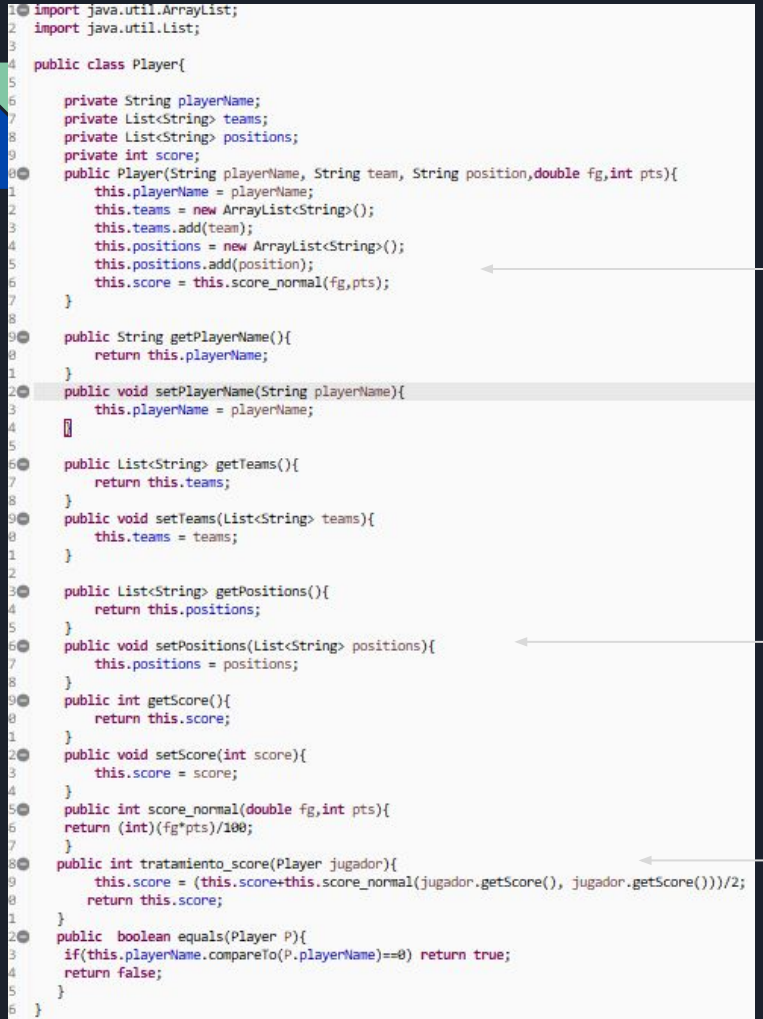


Se utiliza un pivote que se usa para dividir en dos mitades la serie de números.

El pivote deja en el lado izquierdo los números inferiores y en el lado derecho los superiores.

Esto se repite de manera recursiva hasta conseguir la lista ordenada.

Clase Player



```java
import java.util.ArrayList;
import java.util.List;

public class Player{

    private String playerName;
    private List<String> teams;
    private List<String> positions;
    private int score;
    public Player(String playerName, String team, String position,double fg,int pts){
        this.playerName = playerName;
        this.teams = new ArrayList<String>();
        this.teams.add(team);
        this.positions = new ArrayList<String>();
        this.positions.add(position);
        this.score = this.score_normal(fg,pts);
    }

    public String getPlayerName(){
        return this.playerName;
    }
    public void setPlayerName(String playerName){
        this.playerName = playerName;
    }

    public List<String> getTeams(){
        return this.teams;
    }
    public void setTeams(List<String> teams){
        this.teams = teams;
    }

    public List<String> getPositions(){
        return this.positions;
    }
    public void setPositions(List<String> positions){
        this.positions = positions;
    }
    public int getScore(){
        return this.score;
    }
    public void setScore(int score){
        this.score = score;
    }
    public int score_normal(double fg,int pts){
    return (int)(fg*pts)/100;
    }
   public int tratamiento_score(Player jugador){
        this.score = (this.score+this.score_normal(jugador.getScore(), jugador.getScore()))/2;
       return this.score;
   }
   public  boolean equals(Player P){
    if(this.playerName.compareTo(P.playerName)==0) return true;
    return false;
   }
}
```

Constructor

Métodos

Cálculo de *score*

Clase Tratamiento

Leer archivo

Separamos atributos del archivo con ; sustituimos nulos numéricos por 0 y reemplazamos , por .

Si jugador existe, comparamos equipo y posición y si son distintas se añade.

Llamamos al método para calcular *score*

```
import java.io.BufferedReader;

public class Tratamiento {
    public ArrayList<Player> tratamiento(String archivo) throws FileNotFoundException,IOException{
        String SEPARATOR=";";
        ArrayList<Player> lista = new ArrayList<Player>();
        Player aux;
       BufferedReader    br =new BufferedReader(new FileReader("C:/Users/anton/OneDrive/Escritorio/Practica1/NbaStats.csv"));
          String line = br.readLine();
          line = br.readLine();//esta se posiciona en la ultima posicion
          while (line!=null) {
             String [] fields = line.split(SEPARATOR);
             if(fields[8].equals(""))fields[8]= "0";
             if(fields[7].equals("")) fields[7]="0";
             fields[7] =  fields[7].replace(",", ".");
           // Player jugador = new Player((fields[2].isEmpty())?" ": fields[2], (fields[6].isEmpty())?" ": fields[6], (fields[4].isEmpty())?" ":
             //comprobamos que el jugador existe en la lista, si se encuentra en la ultima posicion
             //el aux hay que quitsarlo de aqui ya qe da una excepcion
             Player jugador = new Player(fields[2],fields[6],fields[4],Double.parseDouble(fields[7]),Integer.parseInt(fields[8]));
             if(lista.isEmpty()){
               lista.add(jugador);
               line = br.readLine();
               continue;
             }
             aux = lista.get(lista.size()-1);
             if(jugador.getPlayerName().equals(aux.getPlayerName())){
               if(!jugador.getPositions().get(jugador.getPositions().size()-1).equals(aux.getPositions().get(jugador.getPositions().size()-1))){
                 aux.getPositions().add(jugador.getPositions().get(0));
                 jugador.setPositions(aux.getPositions());
               }
               if(!jugador.getTeams().get(jugador.getTeams().size()-1).equals(aux.getTeams().get(jugador.getTeams().size()-1))){
                 aux.getTeams().add(jugador.getTeams().get(0));
                 jugador.setPositions(aux.getPositions());
               }
               aux.tratamiento_score(jugador);
               //if segundo
             }else{
               lista.add(jugador);
             }//if primero
             line = br.readLine();
          }//while
          br.close();
       return lista;
    }


}
```

Clase Quicksort
(método divide y vencerás)

```java
public class Quicksort {

    public  ArrayList<Player> quicksor(ArrayList<Player> lista) {
        if(lista.size()<=1) return lista;//caso base
        Player pivote=lista.get(0); // tomamos el primer score como referencia
        ArrayList<Player> izquierda = new ArrayList<Player>();
        ArrayList<Player> derecha = new ArrayList<Player>();
        for(int i=1; i < lista.size();i++){
            if(lista.get(i).getScore()< pivote.getScore()){
                izquierda.add(lista.get(i));
            }else{
                derecha.add(lista.get(i));
            }
        }

     ArrayList<Player> izquierdaAux = quicksor(izquierda);
     ArrayList<Player> derechaAux = quicksor(derecha);
     ArrayList<Player> solucion = combinar(izquierdaAux, pivote, derechaAux);
     return solucion;
     }
     public ArrayList<Player> combinar (ArrayList<Player> izquierdaAux,Player pivote,ArrayList<Player> derechaAux){
        ArrayList<Player> aux = new ArrayList<Player>();
        aux.addAll(izquierdaAux);
        aux.add(pivote);
        aux.addAll(derechaAux);
        return aux;
     }
```

caso base

solución

combinación

Clase Quicksort mejorado (método divide y vencerás)

```
public ArrayList<Player> combinar (ArrayList<Player> izquierdaAux,Player pivote,ArrayList<Player> derechaAux){
   ArrayList<Player> aux = new ArrayList<Player>();
   aux.addAll(izquierdaAux);
   aux.add(pivote);
   aux.addAll(derechaAux);
   return aux;
}

public  ArrayList<Player> quicksortMejorado(ArrayList<Player> lista) {
   if(lista.size()<=1) return lista;//caso base
   Player pivote=lista.get(0); // tomamos el primer score como referencia
   ArrayList<Player> izquierda = new ArrayList<Player>();
   ArrayList<Player> derecha = new ArrayList<Player>();
   for(int i=1; i < lista.size();i++){
       if(lista.get(i).getScore()< pivote.getScore()){
           izquierda.add(lista.get(i));
       }else{
           derecha.add(lista.get(i));
       }
   }
ArrayList<Player> izquierdaAux = quicksor(izquierda);
ArrayList<Player> derechaAux = quicksor(derecha);

if(derechaAux.size()>10){
    return combinar(null, null, derechaAux);
}
ArrayList<Player> solucion = combinar(izquierdaAux, pivote, derechaAux);
return solucion;
```

Mejora
Cuando la lista derecha contiene más de 10 jugadores se descarta la lista izquierda.
Y continua únicamente con la lista derecha.

Test

```java
import static org.junit.Assert.assertEquals;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import org.junit.jupiter.api.Test;

public class test {
    @Test
    public void pruebaTratamiento() throws FileNotFoundException,IOException{
        Tratamiento tratamiento_datos = new Tratamiento();
        //insertamos los jugadores, ya tratados
       ArrayList<Player> lista_Jugadores= tratamiento_datos.tratamiento("C:/Users/perez/Desktop/Practica1/NbaStats.csv");
       Quicksort q = new Quicksort();
      long time = System.nanoTime();
      ArrayList<Player> lista =  q.quicksor(lista_Jugadores);
      long time2 = System.nanoTime();
      System.out.println("\nEl tiempo normal es "+(time2-time)+" nanosegundos.\n");
      int i=0;
      System.out.println("Solución: \n");
      while (i<10) {
      System.out.println( lista.get(i).getPlayerName());
      i++;
      }

      long time3 =  System.nanoTime();
      ArrayList<Player> lista2 =  q.quicksor(lista_Jugadores);
      long time4 = System.nanoTime();
      System.out.println("\nEl tiempo del mejorado es "+(time4-time3)+" nanosegundos.\n");
      i=0;
      System.out.println("Solución mejorada: \n");
      while (i<10) {
      System.out.println(lista2.get(i).getPlayerName());
      i++;
      }

        long diferenciaTime=(time2-time)-(time4-time3);
        System.out.println("La diferencia entre ambos tiempos es de "+diferenciaTime +" nanosegundos.");
        assertEquals(lista.get(lista.size()-1), lista2.get(lista2.size()-1));
        assertEquals(lista.get(lista.size()-2), lista2.get(lista2.size()-2));
        assertEquals(lista.get(lista.size()-3), lista2.get(lista2.size()-3));
        assertEquals(lista.get(lista.size()-4), lista2.get(lista2.size()-4));
        assertEquals(lista.get(lista.size()-5), lista2.get(lista2.size()-5));
        assertEquals(lista.get(lista.size()-6), lista2.get(lista2.size()-6));
        assertEquals(lista.get(lista.size()-7), lista2.get(lista2.size()-7));
        assertEquals(lista.get(lista.size()-8), lista2.get(lista2.size()-8));
        assertEquals(lista.get(lista.size()-9), lista2.get(lista2.size()-9));
        assertEquals(lista.get(lista.size()-10), lista2.get(lista2.size()-10));

    }
```

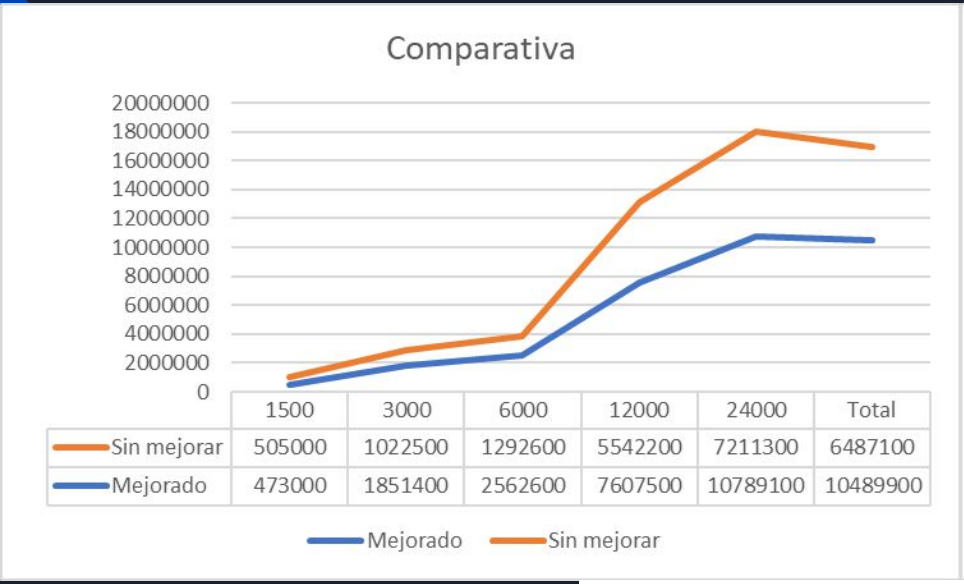
Resultados

```
El tiempo normal es 6093000 nanosegundos.

Solución:

A.W. Holt
Al Jackson
Alex Scales
Alex Stivrins
Andre Dawkins
Andy Panko
Anthony Miller
Antonio Anderson
Art Spector
Barry Sumpter
```

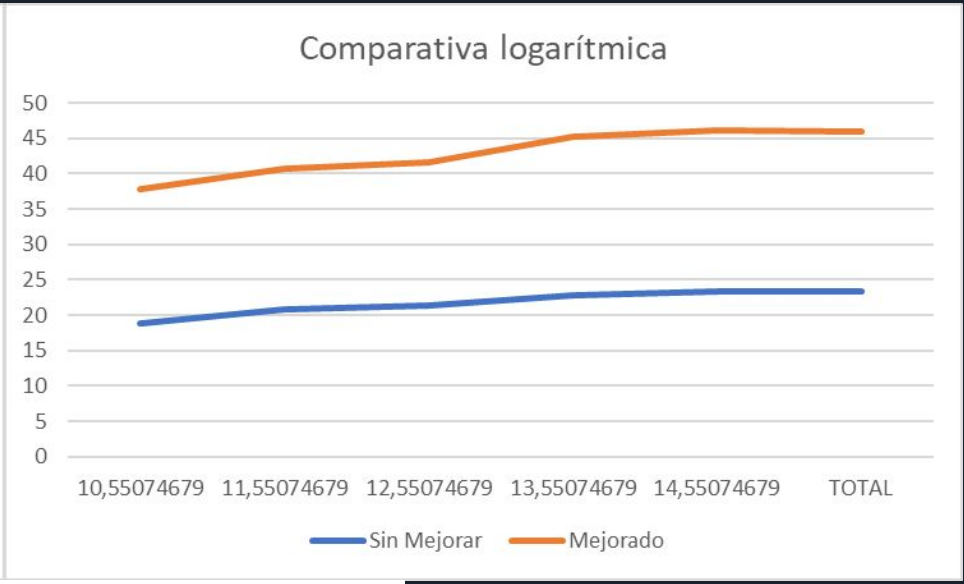
```
El tiempo del mejorado es 4476700 nanosegundos.

Solución mejorada:

A.W. Holt
Al Jackson
Alex Scales
Alex Stivrins
Andre Dawkins
Andy Panko
Anthony Miller
Antonio Anderson
Art Spector
Barry Sumpter
```

```
La diferencia entre ambos tiempos es de 1616300 nanosegundos.
```

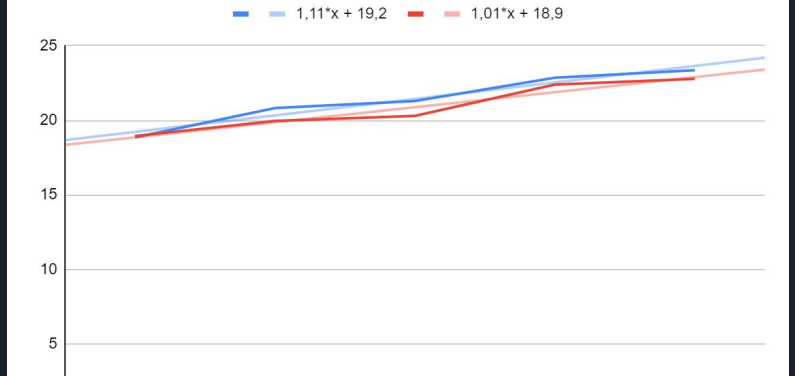
Orden algoritmo



	1500	3000	6000	12000	24000	Total
Sin mejorar	505000	1022500	1292600	5542200	7211300	6487100
Mejorado	473000	1851400	2562600	7607500	10789100	10489900



O(nlogn)



Gracias por su atención