**Análisis de complejidad temporal**

**Selection algorithm:**

|  |  |
| --- | --- |
| **Public void selectionSort(Client objClient) {** | ***# Repeticiones*** |
|  |  |
| **Client newClient = objClient;** | 1 |
| **ArrayList<Game> newArray = objClient.getCodeGame();** | 1 |
|  |  |
| **for(int i = 0; i < newArray.size(); i++) {** | n + 1 |
| **Game min = newArray.get(i);** | n |
| **for(int j = i+1; j < newArray.size(); j++) {** |  |
| **if(newArray.get(j).getStand().compareTo(min.getStand()) < 0) {** |  |
| **Game temp = newArray.get(j);** |  |
| **newArray.remove(j);** |  |
| **newArray.add(j, min);** |  |
| **min = temp;** |  |
| **}** |  |
| **}** |  |
| **newArray.remove(i);** | n |
| **newArray.add(i, min);** | n |
| **}** |  |
| **newClient.setCodeGame(newArray);** | 1 |
|  |  |
| **clientsOfSelection = newClient;** | 1 |
| **}** |  |

Formula de complejidad temporal:

**Insertion algorithm**

|  |  |
| --- | --- |
| **Public void insertionSort(Client clients){** | ***#Repeticiones*** |
|  |  |
| **for(int i = 1; i < clients.getCodeGame().size(); i++) {** | n |
|  |  |
| **for(int j = i; j > 0 && clients.getCodeGame().get(j-1)getStand.compaeTo(clients.getCodeGame().get(j).getStand()) > 0; j--) {** |  |
|  |  |
| **Game tem = clients.getCodeGame().get(j);** |  |
| **clients.getCodeGame().set(j, clients.getCodeGames().get(j-1));** |  |
| **clients.getCodeGame().set(j-1,tem);** |  |
| **}** |  |
| **}** |  |
| **clientsOfInsertion = clients;** | **1** |
| **}** |  |

Formula de complejidad temporal: