Análisis de complejidad temporal

Algoritmo de ordenamiento Insertion sort	*Costos*	*# veces que se repite*	*Prueba de escritorio*
n = clientsGames.size();			n = clientsGames.size() = 4
`public ArrayList<Game> insertionSortOfGames(ArrayList<Game> clientsGames) {`			
`Game temp = null;`	C_1	1	1
`for (int j=1;j<=clientsGames.size()-1;j++) {`	C_2	n	1+1+1+1 = 4
`    for (int k=j;k>0;k--) {`	C_3	(n(n+1)/2) -1	(1+1) (1+1+1) (1+1+1+1) = 9
`        if ((clientsGames.get(k).compareTo(clientsGames.get(k-1)))<0) {`	C_4	(n(n-1)/2)-1	(1+1) (1+1+1) = 5
`            temp = clientsGames.get(k);`	C_5	(n(n-1)/2)-1	(1+1) (1+1+1) = 5
`            clientsGames.set(k, clientsGames.get(k-1));`	C_6	(n(n-1)/2)-1	(1+1) (1+1+1) = 5
`            clientsGames.set(k-1,temp);`	C_7	(n(n-1)/2)-1	(1+1) (1+1+1) = 5
`        }`			
`    }`			
`}`			
`return clientsGames;`	C_8	1	1
`}`			
Total		35	35

$$T(n) = 2 + n + \left(\frac{n(n+1)}{2} - 1\right) + 4\left(\frac{n(n-1)}{2} - 1\right)$$

Algoritmo de ordenamiento Bubble sort	*Costos*	*# veces que se repite*	*Prueba de escritorio*
n = clientsGames.size();			n = clientsGames.size() = 4
public ArrayList<Game> bubbleSortOfGames(ArrayList<Game> clientsGames) {			
Game temp = null;	C_1	1	1
for (int j=1;j<clientsGames.size();j++) {	C_2	n	1+1+1+1 = 4
for (int i=0;i<clientsGames.size()-j;i++) {	C_3	(n(n+1)/2) -1	(1+1+1+1) (1+1+1) (1+1) = 9
if ((clientsGames.get(i).compareTo(clientsGames.get(i+1)))>0) {	C_4	(n(n-1)/2)-1	(1+1) (1+1) (1) = 5
temp = clientsGames.get(i);	C_5	(n(n-1)/2)-1	(1+1) (1+1) (1) = 5
clientsGames.set(i, clientsGames.get(i+1));	C_6	(n(n-1)/2)-1	(1+1) (1+1) (1) = 5
clientsGames.set(i+1,temp);	C_7	(n(n-1)/2)-1	(1+1) (1+1) (1) = 5
}			
}			
}			
return clientsGames;	C_8	1	1
}			
Total		35	35

$$T(n) = 2 + n + \left(\frac{n(n+1)}{2} - 1\right) + 4\left(\frac{n(n-1)}{2} - 1\right)$$

Análisis de complejidad espacial

Algoritmo de ordenamiento Insertion sort	*Almacenamiento*	*Cantidad valores atómicos*
`public ArrayList<Game> insertionSortOfGames(ArrayList<Game> clientsGames) {`		
`Game temp = null;`		
`for (int j=1;j<=clientsGames.size()-1;j++) {`	32 bits	1
`    for (int k=j;k>0;k--) {`	32 bits	1
`        if ((clientsGames.get(k).compareTo(clientsGames.get(k-1)))<0) {`		
`            temp = clientsGames.get(k);`		
`            clientsGames.set(k, clientsGames.get(k-1));`		
`            clientsGames.set(k-1,temp);`		
`        }`		
`    }`		
`}`		
`return clientsGames;`		
`}`		
Total	64 bits	2 = O(1)

Algoritmo de ordenamiento Bubble sort	*Almacenamiento*	*Cantidad valores atómicos*
public ArrayList<Game> bubbleSortOfGames(ArrayList<Game> clientsGames) {		
Game temp = null;		
for (int j=1;j<clientsGames.size();j++) {	32 bits	1
for (int i=0;i<clientsGames.size()-j;i++) {	32 bits	1
if ((clientsGames.get(i).compareTo(clientsGames.get(i+1)))>0) {		
temp = clientsGames.get(i);		
clientsGames.set(i, clientsGames.get(i+1));		
clientsGames.set(i+1,temp);		
}		
}		
}		
return clientsGames;		
}		
Total	64 bits	2 = O(1)