In this document the purpose is to define the complexity of the sorting algorithms that have been used in the solution of the problem.

Time complexity:
Insertion sort:

Line	Value	Repetitions
for(int i = 1; i < games.size(); i++)	C1	n+1
for(int j = i; j > 0 && games.get(j-1).getKey() > games.get(j).getKey(); j--)	C2	n(n+1)/2
Duplex<Integer, Integer> temporal = games.get(j);	C3	n(n+1)/2
games.set(j,games.get(j-1));	C4	n(n+1)/2
games.set(j-1,temporal);	C5	n(n+1)/2

$$O(n) = C_1(n+1) + C_2(\frac{n(n+1)}{2}) + C_3(\frac{n(n+1)}{2}) + C_4(\frac{n(n+1)}{2}) + C_5(\frac{n(n+1)}{2})$$

$$O(n^2)$$

Space complexity:

Type	*Variable*	*Size of 1 atomic value*	*Amount of values*
Input	games	64 bits	n
Aux	i	32 bits	1
Aux	j	32 bits	1
Aux	temporal	64 bits	1

$$O(n) = C_1 n + C_2 + C_2 + C_1$$
$$O(n) = C_1(n+1) + 2C_2$$
$$O(n)$$

Time complexity:
Bubble sort:

Line	Value	Repetitions
boolean changed = true;	C1	1
for(int i = 1; i < findingClients.size()-1 && changed; i++)	C2	n+1

changed = false;	C3	n+1
for(int j = 0; j < findingClients.size()-i; j++)	C4	n(n+1)
if(findingClients.get(j).getKey() > findingClients.get(j+1).getKey())	C5	n(n+1)
changed = true;	C6	n(n+1)
Duplex temp = findingClients.get(j);	C7	n(n+1)
findingClients.set(j,findingClients.get(j+1));	C8	n(n+1)
findingClients.set(j+1, temp);	C9	n(n+1)

$$O(n) = C_1 + C_2(n+1) + C_3(n+1) + C_4(n(n+1)) + C_5(n(n+1)) + C_6(n(n+1$$
$$+ C_7(n(n+1)) + C_8(n(n+1)) + C_9(n(n+1))$$

$$O(n^2)$$

Space complexity:

Type	*Variable*	*Size of 1 atomic value*	*Amount of values*
Input	findingClients	64 bits	n
Aux	i	32 bits	1
Aux	changed	32 bits	1
Aux	j	32 bits	1
Aux	temp	64 bits	1

$$O(n) = C_1 n + C_2 + C_3 + C_4 + C_5$$
$$O(n) = C_1(n+1) + 3C_2$$
$$O(n)$$