TIME ANALYSIS

Clarifications:

- Although there are decisions that lead to “different paths”, the worst case is always sought. Therefore, some parts of the code will not be included in the complexity calculation.
- Only the complexity of the method as such is considered and the complexity of methods such as “whichShelf” is not considered since its only function is to allow comparison.

```
public void bubbleSortGames(String[] info){
    String id = info[0];                                          1
    String[] data = Arrays.copyOfRange(info, 1, info.length);     1
    int n = data.length;                                          1
    if(n==1){                                                     1
       data[0]+=" "+whichShelf(data[0]);
    }
    else{
       for (int i = 0; i < n-1; i++) {                            n-1
          for (int j = 0; j < n - i - 1; j++) {                   n(n+1)/2
             int a = whichShelf(data[j]);                         n(n+1)/2 - n
             data[j]+=" "+a;                                      n(n+1)/2 - n
             int b = whichShelf(data[j+1]);                       n(n+1)/2 - n
             data[j+1]+=" "+b;                                    n(n+1)/2 - n
             if (a >= b) {                                        n(n+1)/2 - n
                String temp = data[j];                            n(n+1)/2 - n
                data[j] = data[j + 1];                            n(n+1)/2 - n
                data[j + 1] = temp;                               n(n+1)/2 - n
             }
          }
       }
    }
    clients.add(new Client(id,data,counter));                     1
    counter++;                                                    1
 }
```

Sum of all lines: $5 - 7n + 9\left(\frac{n(n+1)}{2}\right) = \frac{9n^2}{2} + \frac{9n}{2} - 7n + 5 = \frac{9n^2}{2} - \frac{5n}{2}$

Complexity: $O(n^2)$

```
public void insertionSortGames(String[] info){
    String id = info[0];      1
    int i;   1
    int j;   1
    String aux="";  1
    String[] data = Arrays.copyOfRange(info, 1, info.length);  1
    switch(data.length){   1
      case 1:
        data[0]+=" "+whichShelf(data[0]);
        break;
      case 2:
        int k=whichShelf(data[0]);
        data[0]+=" "+k;
        int q=whichShelf(data[1]);
        data[1]+=" "+q;
        if(q<k){
          String aux2 = data[1];
          data[1]=data[0];
          data[0]=aux2;
        }
        break;
      default:
        for (i = 1; i < data.length; i++) {   n
          aux = data[i];  n - 1
          j = i - 1;    n-1
          int a=whichShelf(data[j]);  n-1
          data[j]+=" "+a;  n-1
          int b=whichShelf(aux);   n-1
          aux+=" "+b;  n-1
          while ((j >= 0) && a >= b ) {   n(n+1)/2 - 1
            data[j + 1] = data[j];  n(n+1)/2 - n
            j = j - 1;  n(n+1)/2 - n
          }
          data[j + 1] = aux;   n-1
        }
    }
    clients.add(new Client(id,data,counter));  1
    counter++;  1
  }
```

Sum of all lines: $6n + 3\left(\frac{n(n+1)}{2}\right) = \frac{3n^2}{2} + \frac{3n}{2} + 6n = \frac{3n^2}{2} + \frac{15n}{2}$

Complexity: $O(n^2)$

SPACE ANALYSIS

Clarifications: m is the size of the String, which may or may not be equal to n.

Bubblesort:

Type	Variable	Size of an atomic value	Number of atomic values
Input	info	m*(16 bits)	n
Auxiliary	data n i j a b temp counter	m*(16 bits) 32 bits 32 bits 32 bits 32 bits 32 bits m*(16 bits) 32 bits	n-1 1 1 1 1 1 1 1
Output			
Total			2n+6 = O(n)

Total Space Complexity = 2n+6 = $\theta(n)$

Auxiliary Space Complexity = n+6 = θ(n)

Auxiliary + Output Space Complexity = n+6 = θ(n)

Insertionsort:

Type	Variable	Size of an atomic value	Number of atomic values
Input	info	m*(16 bits)	n
Auxiliary	i j aux data k q aux2 a b	32 bits 32 bits m*(16 bits) m*(16 bits) 32 bits 32 bits m*(16 bits) 32 bits 32 bits	1 1 1 n-1 1 1 1 1 1
Output			
Total			2n+7 = O(n)

Total Space Complexity = 2n+7 = $\theta(n)$

Auxiliary Space Complexity = n+7 = θ(n)

Auxiliary + Output Space Complexity = n+7 = θ(n)