With an insertion sort:

Pseudocode:

```
1.  //Insertion Sort:
2.
3.      for i ←— 1  to  i <= size-1 step i++          θ (n-1)
4.      do{
5.          j ←— I                                   θ (1)
6.          While  j > 0    and    myArr[j-1] > myArr[j]   θ (n)
7.          do{
8.              temp ←— myArr[j]                     θ(1)
9.              myArr[j] ←— myArr[j-1]               θ(1)
10.             myArr[j-1]←— temp;                   θ(1)
11.             j--                                  θ(1)
12.         }
13.     }
```

Complexity:

Best case of insertion sort is: O(n).

Worst case of insertion sort is: O(n^2)

Code using C programming language:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int size; //The number of the array's elements.
    printf("Enter the size of the array:\t"); //ask for enter the size of an array
    scanf("%d" , &size); // take the size from the user
    int myArr[size]; //enter the size of elements

    printf("Enter an element:\t");
    //Loop to enter the element to array:

    for (int i = 0 ; i < size ; i++)
    {
        scanf("%d" , &myArr[i]);
    } // O(n)


    //Insertion Sort:

    for(int i = 1 ; i <= size-1 ; i++)
    { //O(n)
        int j = i;
        while(j > 0 && myArr[j-1] > myArr[j])
        {
            int temp = myArr[j];
            myArr[j] = myArr[j-1];
            myArr[j-1] = temp;
            j--;
        }
    } //O(n^2)

    int max = myArr[size-1];
    int count = 0 ;//Counter to count how much the largest number is iterated
    //now myArr is sorted:
```

```c
    //now myArr is sorted:

    for (int i = size-1 ; i>=0 ; i--)
    {
        if(max == myArr[i])
        {
            count++;
        }
    }
    printf("The Largest number is iterated :   %d" ,count);



    return 0;
}
```

Without a sort method:

pseudocode:

```
1.  Max ← sizeOfCandy[0]                                          θ(1)
2.  int count ←1; // in case first element is the largest one     θ(1)
3.
4.  For i← 1    to    i < numberOfCandy step i++                  θ(n-1)
5.  do{
6.      if    sizeOfCandy[i] > max  {                             θ(1)
7.      max ← sizeOfCandy[i]                                      θ(1)
8.      Count ←1                                                  θ(1)
9.      }
10.     else if (sizeOfCandy[i]==max){
11.     count++                                                   θ(1)
12.     }
13. }
14. Print the counter.                                            Θ(1)
```

Complexity:

In the best case: O(n)

In the worst case: O(n)

Code using C:

```c
#include <stdio.h>
#include <stdlib.h>

int main(){

    int numberOfCandy;
    // Enter number of candies
    printf("Enter number of candies: \t");
    scanf("%d", &numberOfCandy);
    // Enter size of candies
    printf("Enter candies: \t");
    int sizeOfCandy [numberOfCandy];

    for( int i=0;i<numberOfCandy;i++){
            printf("Enter the next candy: \t");
            scanf("%d", &sizeOfCandy[i]);
    } // o(n)

    int max=sizeOfCandy[0];
    int count =1; // in case first element is the largest one

    for(int i = 1; i < numberOfCandy; i++){
         if (sizeOfCandy[i]>max){
         max = sizeOfCandy[i];
         count=1; // count from beginning
         }else if (sizeOfCandy[i]==max){
         count++;
         }
    }

    printf("The number that the largest number has been iterated is :    %d", count);
     return 0;
}

```