Insert Sort

El loop principal (`for`) recorre cada elemento. Luego tiene un loope secundario (`while`) que ayuda a insertar cada elemento en el lugar correcto del arreglo, entre sus predecesores (esto es, hay un índice `j` que recorre el arreglo en sentido inverso)

```
procedure insert(T[1..n])
    for i = 2 to n
        // Me paro delante de los numeros que quiero analizar.
        x = T[i]
        j = i - 1
        while j > 0 and x < T[j] do
            // Recorro los números hacia atras,
            // Siempre que mi x (el valor que estoy comparando)
            // sea menor a los valores que me encuentro (T[J]),
            // muevo hacia adelante el valor que me encuentro (T[j])
            T[j+1] = T[j]
            j = j - 1
        // en el final, asígno al último valor de j
        // el que estoy analizando (x)
        T[j+1] = x
```

probemos ejecutar este código con $T = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]$

1. $T = [1, 3, 4, 1, 5, 9, 2, 6, 5, 3]$
2. $T = [1, 3, 4, 1, 5, 9, 2, 6, 5, 3]$
3. $T = [1, 1, 3, 4, 5, 9, 2, 6, 5, 3]$
4. $T = [1, 1, 3, 4, 5, 9, 2, 6, 5, 3]$
5. $T = [1, 1, 3, 4, 5, 9, 2, 6, 5, 3]$
6. $T = [1, 1, 2, 3, 4, 5, 9, 6, 5, 3]$
7. $T = [1, 1, 2, 3, 4, 5, 6, 9, 5, 3]$
8. $T = [1, 1, 2, 3, 4, 5, 5, 6, 9, 3]$
9. $T = [1, 1, 2, 3, 3, 4, 5, 5, 6, 9]$

Si lo ejecutamos con $U = [1, 2, 3, 4, 5, 6]$, queda igual en todas las itereaciones del for.

Si lo ejecutamos con $V = [6, 5, 4, 3, 2, 1]$
1. $V = [5, 6, 4, 3, 2, 1]$
2. $V = [4, 5, 6, 3, 2, 1]$
3. $V = [3, 4, 5, 6, 2, 1]$
4. $V = [2, 3, 4, 5, 6, 1]$
5. $V = [1, 2, 3, 4, 5, 6]$

Observación: En una lista ya ordenada (cómo es el caso de U) vemos que el tiempo de ejecución es lineal: sólo requiere recorrer los elementos de U, pero no comporarlos con el resto.