# insertion_sort

March 9, 2021

## 1 Ordenamiento por Inserción (Insertion Sort)

**Complejidad Algoritmica:** $O(n^2)$

En este algoritmo se ordena *"en su lugar"*, es decir que no se crea una nueva lista con los elementos ordenados si no que en lugar de eso se modifican los valores en memoria.

### 1.1 Funcionamiento

1. El primer elemento esta ordenado por defición, el resto de elementos de la lista se ordenarán a partir de este.
2. Se recorren los demás elementos, comparándolos con el que esta ordenado por definición. Si el elemento que se esta comparando es **mayor** que uno de los elementos ya ordenados se mueve dicho elemento un índice a la derecha y se inserta el nuevo elemento
3. Se repite el paso anterior con los todos los elementos de la lista, es decir, se toma un elemento sin ordenar, se compara con cada uno de los que no están ordenados y se inserta en el lugar correspondiente.

### 1.2 Implementación

1. Crear la lista con los elementos que se quieren ordenar

[1]:
```
elements_list = [3, 14, 5, 2, 1, 8, 13, 24]
```

2. Recorrer los elementos e irlos ordenando en base a los elementos que ya están ordenados.

[2]:
```
# Recorre los elementos de la lista saltandose el primero pues este esta␣
↪ordenado por definición
for i in range(1, len(elements_list)):
    current_value = elements_list[i]
    current_position = i

    print(f'FOR: current_value = {current_value}\tcurrent_position =␣
↪{current_position}')

    # Mientras la posición actual sea > 0 y el elemento anterior sea mayor que␣
↪el elemento actual
    # se hace un intercambio
```

```
   while current_position > 0 and elements_list[current_position - 1] >␣
↪current_value:
       # Pasa el elemento actual a la posición  anterior
       elements_list[current_position] = elements_list[current_position - 1]

       # Se decrementa esta variable para evitar que se asignen elementos a␣
↪índices
       # que no existen y el array se desborde.
       # Por ejemplo, en caso de que todos los elementos anteriores sean␣
↪mayores al
       # elemento actual el while se ejecutará hasta poner el elemento actual
       # en la primera posición y luego se detendrá
       current_position -= 1

       print(f'--> WHILE: current_value = {current_value}\tcurrent_position =␣
↪{current_position}')

   # Dado que si se entra en el while los elementos se corren una posición a␣
↪la izquierda
   # esta línea se encarga de asignar el valor actual al espacio que quedaría␣
↪cada vez
   # que se intercabien los elementos en el while
   elements_list[current_position] = current_value
   print(elements_list)
```

```
FOR: current_value = 14 current_position = 1
[3, 14, 5, 2, 1, 8, 13, 24]
FOR: current_value = 5  current_position = 2
--> WHILE: current_value = 5    current_position = 1
[3, 5, 14, 2, 1, 8, 13, 24]
FOR: current_value = 2  current_position = 3
--> WHILE: current_value = 2    current_position = 2
--> WHILE: current_value = 2    current_position = 1
--> WHILE: current_value = 2    current_position = 0
[2, 3, 5, 14, 1, 8, 13, 24]
FOR: current_value = 1  current_position = 4
--> WHILE: current_value = 1    current_position = 3
--> WHILE: current_value = 1    current_position = 2
--> WHILE: current_value = 1    current_position = 1
--> WHILE: current_value = 1    current_position = 0
[1, 2, 3, 5, 14, 8, 13, 24]
FOR: current_value = 8  current_position = 5
--> WHILE: current_value = 8    current_position = 4
[1, 2, 3, 5, 8, 14, 13, 24]
FOR: current_value = 13 current_position = 6
--> WHILE: current_value = 13   current_position = 5
[1, 2, 3, 5, 8, 13, 14, 24]
```

```
FOR: current_value = 24 current_position = 7
[1, 2, 3, 5, 8, 13, 14, 24]
```

3. Imprimir la lista ordenada

[3]:
```
print(elements_list)
```

```
[1, 2, 3, 5, 8, 13, 14, 24]
```