



Trabajo práctico nº1

Introducción a la programación

Comisión-5

Semestre 2. 2025

Integrantes del grupo

**Agustín Gutierrez
Camila Elizabeth Benítez Marín**

INTRODUCCIÓN:

El trabajo práctico consiste en la implementación de algoritmos de ordenamiento y la visualización de los mismos desde el navegador. En este trabajo se completó el código del visualizador con tres algoritmos: **Bubble sort, Selection sort, Insertion sort**, cumpliendo con el contrato que utiliza el visualizador. Bajo un modelo de ejecución paso a paso. Cada llamada a la función `step()` debe realizar la unidad mínima de trabajo (una comparación o un intercambio). Esto requiere utilizar variables globales para continuar el estado del algoritmo entre llamadas, simulando los bucles `for` tradicionales.

El Contrato de la Función `step()`

La función `step()` debe retornar un diccionario con la siguiente información:

`a` (int): Índice del primer elemento involucrado (comparado o intercambiado).

`b` (int): Índice del segundo elemento involucrado (comparado o intercambiado).

`swap` (bool): Es `True` si se realizó un intercambio de datos. Es `False` si solo fue una comparación.

`done` (bool): Es `True` si el algoritmo ha finalizado el ordenamiento.

Bubble Sort (Ordenamiento Burbuja)

```
15 def step():
16     #TODO:
17     global items, n, i, j
18     # 1) Elegir índices a y b a comparar en este micro-paso (según tu Bubble).
19     a=j
20     b=j+1 #adyacente del indice, elemento de al lado
21     # 2) Si corresponde, hacer el intercambio real en items[a], items[b] y marcar swap=True
22     if i < n-1:
23         if items[a]>items[b]:#sin ciclos for porque funciona "paso a paso"
24             items[a],items[b]= items[b],items[a] #intercambio de menor a mayor
25             swap=True
26         else:
27             swap=False
28
29     j=j+1#aumento el indice
30
31     if j >= n-i-1:#si hay que reiniciar el indice
32         j=0
33         i=i+1
34     #Devolver {"a": a, "b": b, "swap": swap, "done": False}.
35     return {"a": a, "b": b, "swap": swap, "done": False}
36     # Cuando no queden pasos, devolver {"done": True}.
37     if i >=n-1:
38         return {"done": True}
```

El **Bubble Sort** compara y potencialmente intercambia elementos adyacentes, haciendo que el elemento más grande "burbujee" hasta su posición final en cada pasada.

Variables de Estado Globales:

i: Contador del bucle externo. Rastrea cuántos elementos ya están ordenados al final de la lista.

j: Cursor del bucle interno. Recorre la porción no ordenada comparando adyacentes ($a=j$ y $b=j+1$)

swap: Bandera que indica si hubo algún intercambio en la pasada actual. Se usa para la optimización: si no hay swaps, el algoritmo termina.

La dificultad principal con este algoritmo fue a la hora de completarlo, ya que a pesar de tener ejemplos de internet, teníamos que adaptarlo y no podíamos usar los ciclos For, porque había que hacerlo "paso a paso" .

Insertion Sort (Ordenamiento por Inserción)

```
13 def step():
14     global items, n, i, j
15
16     # Si ya recorrimos todos los elementos, terminamos
17     if i >= n:
18         return {"done": True}
19
20     # Si j es None, iniciamos desplazamiento para el elemento en posición i
21     if j is None:
22         j = i
23         a = j - 1
24         b = j
25         return {"a": a, "b": b, "swap": False, "done": False}
26
27     # Calculamos a y b en cada paso antes de comparar
28     a = j - 1
29     b = j
30
31     # Si el elemento anterior es mayor, hacemos un swap (desplazamiento a la izquierda)
32     if j > 0 and items[a] > items[b]: # comparación
33         items[a], items[b] = items[b], items[a] # intercambio
34         j = j - 1
35         return {"a": a, "b": b, "swap": True, "done": False}
36
37     # Si no hay más que desplazar, avanzamos al siguiente elemento
38     i = i + 1
39     j = None
40     return {"a": a, "b": b, "swap": False, "done": False}
```

El **Insertion Sort** toma el elemento en la posición i y lo inserta en su lugar correcto dentro de la sublista ordenada a su izquierda, desplazando los elementos mayores.

Variables de Estado Globales:

i: Puntero del bucle externo. Marca el elemento que se va a insertar.

j: Puntero del bucle interno. Cursor que se mueve hacia la izquierda ($j-1$ y j) para desplazar elementos y encontrar la posición de inserción.

La dificultad principal al implementar este algoritmo fue a la hora de actualizar correctamente los índices, especialmente el índice *i*, *a* y *b* (estos índices hacen que avance la comparación de las barras en el visualizador). Estos problemas en el visualizador hacían que no se recorran completamente las barras y las ordene.

Selection Sort (Ordenamiento por Selección)

```
19 def step():
20     global items, n, i, j, min_idx, fase
21
22     if i >= n - 1:
23         return {"done": True}
24
25     if fase == "buscar":
26
27         if j < n:
28             a = min_idx
29             b = j
30
31             if items[j] < items[min_idx]:
32                 min_idx = j
33                 a = min_idx
34
35             j += 1
36
37             return {"a": a, "b": b, "swap": False, "done": False}
38
39         else:
40             fase = "swap"
41             return {"a": i, "b": min_idx, "swap": False, "done": False}
42
43     elif fase == "swap":
44
45         swap_realizado = False
46         a = i
47         b = min_idx
48
49         if min_idx != i:
50             items[i], items[min_idx] = items[min_idx], items[i]
51             swap_realizado = True
52
53         # Reinicio
54         i += 1
55         j = i + 1
56         min_idx = i
57         fase = "buscar"
58
59         return {"a": a, "b": b, "swap": swap_realizado, "done": False}
```

El **Selection Sort** encuentra el elemento mínimo en la porción no ordenada (desde *i*) y lo intercambia con el elemento en la posición *i*. Utiliza la variable *fase* para controlar la ejecución.

Variables de Estado Globales:

i: Puntero del bucle externo. Marca el inicio de la porción no ordenada y la posición final del elemento mínimo.

j: Puntero del bucle interno. Cursor que recorre la porción no ordenada buscando el mínimo.

min_idx: Guarda el índice del valor mínimo encontrado en la pasada actual.

fase: Controla el flujo: "buscar" (encuentra el mínimo) o "swap" (realiza el intercambio y reinicia).

La principal dificultad fue al implementar este algoritmo en el formato "paso a paso" sin usar ciclos for, fue en la coordinación precisa entre las dos fases del algoritmo y el reinicio de los índices.