Ordenamiento

E. Rivas

Abril de 2014

Sobre la materia

$$\text{EDyA} = \text{ED} \wedge \text{A}$$

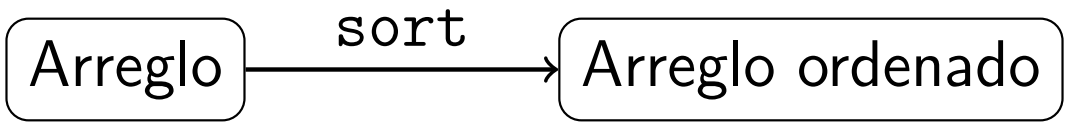
ED: listas, árboles, pilas, colas, heaps, hashtables...

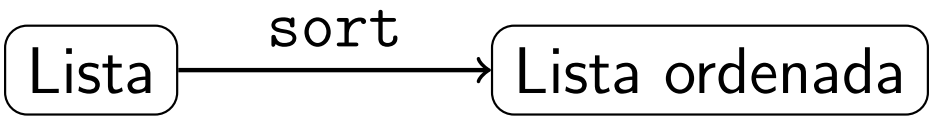
A: ???

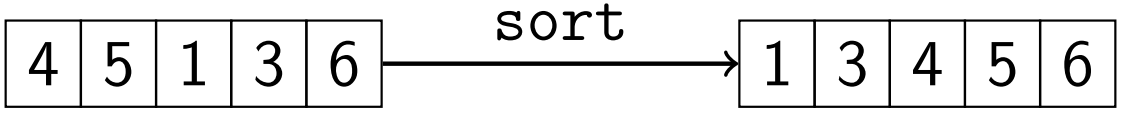
¡Hasta hoy!

Problema de ordenamiento

Buscar una función `sort` tal que:







Función sort - forma

Pueden existir varios sabores para la función que buscamos:

```
void sort(int data[], int sz);

int *sort(int data[], int sz);

void sort(SList *l);

SList *sort(SList *l);
```

Función `sort` - contenido

Y puede funcionar de varias maneras. Cada una de estas maneras es un algoritmo.

Hoy veremos algunos algoritmos de ordenamiento clásicos:

- Burbuja
- Inserción
- Selección

En las clases siguientes, veremos otros más:

- Quicksort
- Mergesort
- Heapsort

Burbuja

Una manera de verificar que esté ordenado el arreglo es viendo que cada elemento sea menor al elemento que está a la derecha:

```
sorted = 1;
for (i = 0; i < sz - 1; i++)
  if (data[i] > data[i+1])
    sorted = 0;
```

Al terminar este código, `sorted` valdrá `1` si el arreglo está ordenado, y `0` en caso contrario.

Burbuja

En caso de que un elemento tenga un elemento más chico a su derecha, podemos intentar repararlo intercambiando los elementos.

El algoritmo de la burbuja repite esta reparación hasta asegurarse que el arreglo está ordenado:

```
do {
  sorted = 1;
  for (i = 0; i < sz - 1; i++)
    if (data[i] > data[i+1]) {
      sorted = 0;
      swap(&data[i], &data[i+1]);
    }
} while (sorted != 1);
```

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

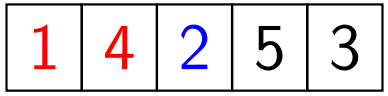
El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.



Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

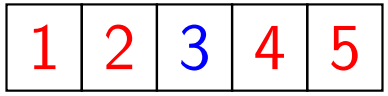
El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.



Inserción

El procedimiento general de este algoritmo funciona asumiendo que dividimos el arreglo en dos: la parte ordenada y la parte desordenada. Sus pasos son:

- Tomar un elemento de la parte sin ordenar.
- Insertarlo en la parte ordenada del arreglo, en la posición que le corresponde.
- Volver al primer paso, hasta que no queden elementos en la parte sin ordenar.

Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

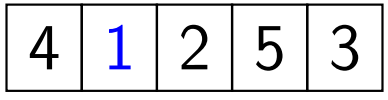
- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).

4	1	2	5	3

Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

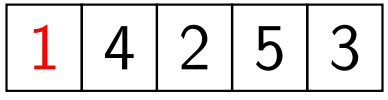
- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

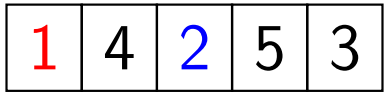
- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).

Selección

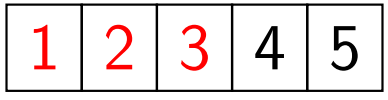
En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).

Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

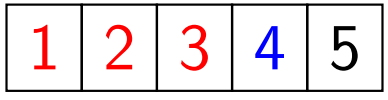
- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

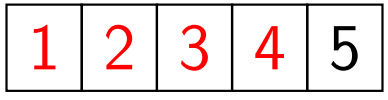
- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

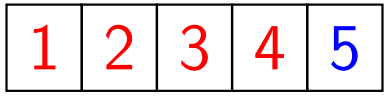
- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

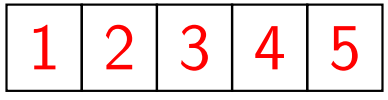
- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Selección

En este algoritmo también asumimos el arreglo divido en una parte ordenada y una parte desordenada. Pero esta vez en cada paso pasamos el mínimo:

- Buscamos el mínimo elemento en la parte desordenada.
- Intercambiamos el mínimo con el primer elemento de la parte desordenada.
- Volvemos al primer paso, considerando que el arreglo desordenado con un elemento menos (su mínimo).



Ejercicio

- Escriba la función `void sort(int data[], int sz)` usando selección o inserción.

```
void sort(int data[], int sz) {
  int i, j, min;
  for (i = 0; i < sz; i++) {
    min = i;
    for (j = i + 1; j < sz; j++) {
      if (data[j] < data[min])
        min = j;
    }
    swap(&datos[i], &datos[min]);
  }
}
```

Complejidad

Hemos visto tres algoritmos de ordenamiento hasta ahora... ¿cuál elegir?

¿Cuál es el más rápido?

Por cada proposición de C, se generan varias instrucciones que ejecuta el microprocesador. Si contamos la cantidad de instrucciones que ejecuta el procesador, tenemos una manera de comparar los algoritmos.

¿Y el espacio? ¿Con listas es igual?

Generalidad

¿Qué pasa si en lugar de ordenar enteros deseamos ordenar otro tipo de datos?

Abstraemos el tipo de dato que deseamos ordenar usando `void *`:

```
void sort(void *data[], int sz);
```

¿Qué nos está faltando? Una función que compare este tipo de datos.

```
int cmp(void *, void *);
```

Generalidad

La función `int cmp(void *a, void *b)` devolverá:

- Un negativo si `a` va antes de `b`.
- `0` si `a` y `b` son equivalentes.
- Un positivo si `a` va luego de `b`.

La función `sort` queda con la signatura:

```
typedef int (*CmpFunc)(void *, void *);

void sort(void *data[], int sz, CmpFunc cmp);
```

Conclusión

- Definimos el problema de ordenamiento.
- Presentamos tres algoritmos:
  - Burbuja.
  - Selección.
  - Inserción.
- Discutimos qué puede ser la “complejidad” de un algoritmo.
- Discutimos cómo generalizar el ordenamiento a tipos de datos diversos.