

# Ordenación y Selectionsort

Clase 01

IIC 2133 - Sección 3

Prof. Eduardo Bustos

# Sumario

**Introducción**

Correctitud de algoritmos

Selection Sort

Cierre

# El Misterio de EDD

- Falló Canvas y no tenemos acceso a la lista de estudiantes del sistema
- Sin embargo, hicimos correr unas hojas por la sala para que se anotaran durante la primera clase
- La Universidad pregunta si ese día el estudiante Misterio Zallen estuvo en la sala B21
- ¿Cómo responder lo más rápido posible?

# El Misterio de EDD

¿ Zallen Misterio ∈

Apellido	Nombre
Alen	Misterio
Misterio	Misterio
Zalen	Berenice
Gonzalópez	D
Turing	Alan
Misterio	Yadran
Zeta	Hache
Ararán	Jota
Alenn	Cristina
...	...

pág. 1/376

?

# El Misterio de EDD

¿ Zallen Misterio €

Apellido	Nombre
Abarca	Yadran
Abusleme	Nicole
Arenas	Camila
Arenas	D
Bañados	Richard
Beterraga	Brócoli
Blanco	Ximena
Brahms	Johannes
Castillo	Raquel
...	...

pág. 1/376

?

# El Misterio de EDD

¿ 245.789.546 €

?

Valor
1
2
3
56
57
64
68
99
124
125
...
pág. 1/376

# Secuencias ordenadas

## Definición

Una secuencia de números  $a_1, a_2, \dots, a_n$  se dice **ordenada** (no decrecientemente) si cumple que

$$a_1 \leq a_2 \leq \dots \leq a_n$$

¿Qué es entonces **ordenar** una secuencia de números arbitraria?

# Una propuesta de algoritmo de ordenación

**input** : Lista de nombres  $L$

**output**: Lista ordenada de nombres  $L'$

Misterio ( $L$ ):

- 1    Iniciamos  $L'$  vacía
  - 2    Encontrar el menor valor en  $L$
  - 3    Borrar el valor encontrado
  - 4    Escribirlo al final de  $L'$  (en el primer espacio disponible)
  - 5    Si quedan valores en  $L$ , volver a la línea 2
- return**  $L'$

¿Es **correcto** este algoritmo?



# Objetivos de la clase

- ☐ Comprender el concepto de ordenación como operación abstracta
- ☐ Interpretar de forma intuitiva un algoritmo en pseudocódigo
- ☐ Demostrar correctitud de un algoritmo dado
- ☐ Determinar complejidad del algoritmo de ordenación Selection Sort
- ☐ Distinguir ventajas comparativas de EDD's en las operaciones empleadas en Selection Sort

# Sumario

Introducción

**Correctitud de algoritmos**

Selection Sort

Cierre

# Correctitud de algoritmos

En este curso nos centraremos en dos propiedades para decir que un algoritmo es **correcto**:

- Termina en una cantidad finita de pasos
- Cumple su propósito

En un problema de ordenación, un algoritmo será correcto si **ordena** los datos en una cantidad finita de pasos

# Recordatorio: Inducción simple

## Principio de inducción simple

Para una afirmación  $P(x)$  sobre los naturales, si  $P(x)$  cumple que:

1.  $P(0)$  es verdadero,
2. para todo  $n \in \mathbb{N}$ , si  $P(n)$  es verdadero, entonces  $P(n+1)$  es verdadero,

entonces para todo  $n \in \mathbb{N}$  se tiene que  $P(n)$  es verdadero.

## Notación

- $P(0)$  se llama el **caso base**.
- En el paso 2.
  - $P(n)$  se llama la **hipótesis de inducción**.
  - $P(n+1)$  se llama la **tesis de inducción** o paso inductivo.

# Recordatorio: Inducción simple

## Inducción simple en la práctica

Para una afirmación  $P(x)$  sobre los naturales

1. Identificamos caso base  $n_0$  y verificamos que cumple  $P$
2. Suponemos que  $P(n)$  es cierta para un  $n$  **cualquiera**
3. Demostramos que  $P(n+1)$  es verdadero usando  $P(n)$

Si logramos completar estos pasos, concluimos que  $P$  es cierta para  $n \geq n_0$

No olvidar

- La propiedad puede ser cierta desde un  $n_0 \neq 0$
- Puede haber varios casos base

# Soltemos la mano...

## Ejemplo

Demuestre que el siguiente algoritmo es correcto.

**input** : Lista de nombres  $L$

**output**: Lista ordenada de nombres  $L'$

Misterio ( $L$ ):

- 1    Iniciamos  $L'$  vacía
  - 2    Encontrar el menor valor en  $L$
  - 3    Borrar el valor encontrado
  - 4    Escribirlo al final de  $L'$  (en el primer espacio disponible)
  - 5    Si quedan valores en  $L$ , volver a la línea 2
- return**  $L'$

# Soltemos la mano...

## Demostración (finitud)

Primero demostraremos que termina en una cantidad finita de pasos.

Sea  $n$  la cantidad (finita) de valores en la lista original  $L$ .

En cada iteración del algoritmo se borra un valor de  $L$  y se escribe en la lista nueva  $L'$ .

Luego de  $n$  iteraciones, todos los valores en  $L$  fueron borrados. Debido a la línea 5, que verifica si existen valores en  $L$ , el algoritmo no sigue iterando.

Por lo tanto, el algoritmo termina en una cantidad finita de pasos. □

# Soltemos la mano...

## Demostración (ordenación)

Demostraremos por **inducción** que efectivamente genera una lista ordenada.

1. Probamos el caso base  $P(n_0)$
2. Suponemos verdadera la hipótesis inductiva  $P(n)$ .  
P.D. Tesis inductiva  $P(n + 1)$ ...

¿Qué propiedad probaremos?

¿Cuál es el parámetro  $n$  que haremos avanzar?

¿Cuáles son los casos base?



# Soltemos la mano...

## Demostración (ordenación)

Consideremos la propiedad

$P(n) \quad := \quad$  Los primeros  $n$  valores borrados de  $L$  son los  
menores de  $L$  y están ordenados en  $L'$

1. **Caso base.**  $P(1)$  corresponde al estado de las listas luego de borrar el menor elemento de  $L$ . Este elemento es el menor de todos debido al criterio de selección de la línea 2 del algoritmo. Dado que  $L'$  solo tiene un elemento, a saber  $a_1$ ,  $L'$  está ordenada.

# Soltemos la mano...

## Demostración (ordenación)

Consideremos la propiedad

**P(n)**     $:=$     Los primeros  $n$  valores borrados de  $L$  son los menores de  $L$  y están ordenados en  $L'$

2. **Hipótesis inductiva (H.I.)** Suponemos que los primeros  $n$  valores borrados de  $L$  son los menores y se encuentran ordenados en  $L'$ .

**P.D.** Los primeros  $n + 1$  elementos borrados de  $L$  son los menores y se encuentran ordenados en  $L'$ .

Por **H.I.** los primeros  $n$  elementos borrados corresponden a los  $n$  menores de  $L$  y están ordenados en  $L'$ . Asignándoles las etiquetas  $a_i$  a estos elementos, tenemos que

$$a_1 \leq a_2 \leq \cdots \leq a_n \leq \text{cualquier elemento restante en } L$$

# Soltemos la mano...

## Demostración (ordenación)

Al borrar el elemento  $n + 1$  de  $L$ , al que llamamos  $a_{n+1}$ , por el criterio de selección de la línea 2 sabemos que este es el menor de los que restan en  $L$  y por la **H.I.** sabemos que

$$a_1 \leq a_2 \leq \dots \leq a_n \leq a_{n+1} \leq \text{el resto en } L$$

Al agregar  $a_{n+1}$  al final de  $L'$ , dado que por **H.I.** los primeros  $n$  elementos están ordenados de forma no decreciente,  $L'$  queda ordenada con  $n + 1$  elementos.

Con esto se concluye que el algoritmo ordena.



# Sumario

Introducción

Correctitud de algoritmos

**Selection Sort**

Cierre

# El algoritmo SelectionSort

El algoritmo Misterio se conoce como **selection sort**.

**input** : Secuencia de datos  $A$

**output**: Nueva secuencia de datos  $B$ , ordenada

SelectionSort ( $A$ ):

- 1 Definir secuencia  $B$ , inicialmente vacía
  - 2 Buscar el menor dato  $x$  en  $A$
  - 3 Sacar  $x$  de  $A$  e insertarlo al final de  $B$
  - 4 Si quedan valores en  $A$ , volver a la línea 2
- return**  $B$

Paréntesis: un recurso muy útil

visualizador de algoritmos

# ¿Cuál es la complejidad de SelectionSort?

## Ejemplo

Determine la complejidad de SelectionSort

**input** : Secuencia de datos  $A$

**output**: Nueva secuencia de datos  $B$ , ordenada

SelectionSort ( $A$ ):

- 1 Definir secuencia  $B$ , inicialmente vacía
  - 2 Buscar el menor dato  $x$  en  $A$
  - 3 Sacar  $x$  de  $A$  e insertarlo al final de  $B$
  - 4 Si quedan valores en  $A$ , volver a la línea 2
- return**  $B$

# ¿Cuál es la complejidad de SelectionSort?

Dada la secuencia de input  $A$ , diremos que  $|A| = n$  es su largo cuando esta posee  $n$  elementos.

Podemos determinar la complejidad de SelectionSort en función de  $n$  a través de dos estrategias.

De forma intuitiva

- Buscar el menor dato en  $A$  significa revisar **todos** los elementos:  $\mathcal{O}(n)$
- La búsqueda del menor elemento se hace una vez por dato, i.e.  $n$  veces
- Combinando los resultados obtenemos  $\mathcal{O}(n^2)$



# ¿Cuál es la complejidad de SelectionSort?

De forma explícita

- Buscar el menor elemento de  $A$  implica revisar  $n$  elementos en la primera iteración
- En la siguiente iteración, como  $A$  tiene un elemento menos implica revisar  $n - 1$  elementos
- Este proceso se repite hasta agotar  $A$  cuando tiene 1 elemento
- El tiempo del algoritmo es la suma de los tiempos de cada iteración

$$T(n) = n + (n - 1) + \dots + 2 + 1 = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

- Concluimos que  $T(n) \in \mathcal{O}(n^2)$



# Un paréntesis sobre la escritura de datos

No olvidemos el espíritu del curso: **estructuras de datos**

¿Qué tan costosa es la inserción del dato al final de  $B$  en SelectionSort?

Recordemos las dos estructuras básicas que discutimos: **arreglos** y **listas ligadas**

Mencionamos sus diferencias en cuanto al acceso por índice, pero ahora agregamos otra operación: **inserción**, i.e. añadir un dato a la estructura

En el contexto de SelectionSort nos interesa específicamente la **inserción del último elemento**

- En una lista ligada, si se cuenta con un puntero adicional al último elemento, la inserción toma  $\Theta(1)$
- En un arreglo
  - si no se supera el espacio reservado para este, toma  $\Theta(1)$
  - si se agotó el espacio, hay que reubicar todo el arreglo:  $\Theta(n)$

# Un paréntesis sobre la escritura de datos

Con esto, las operaciones que conocemos tienen las siguientes complejidades

Operación	Arreglo	Lista ligada
Acceso por índice	$\Theta(1)$	$\Theta(n)$
Inserción (final)	$\Theta(1)$ sin reubicar $\Theta(n)$ reubicando	$\Theta(1)$ con puntero al final

No perdamos de vista la pregunta de  
qué estructura es la **más adecuada** en cada escenario

Cabe notar que para el caso de los arreglos, en SelectionSort sabemos exactamente cuánto espacio necesitamos reservar

Más aún, veremos una forma de usar la menor cantidad de memoria posible con arreglos

# Complejidad de memoria en SelectionSort

- Para secuencias de datos muy grandes, puede ser costoso reservar espacio para una copia completa de los datos
- Cabe notar que SelectionSort se puede hacer en **un solo arreglo**, ya que en todo momento

$$|A| + |B| = n$$

- Como *reciclam*os el espacio dentro del mismo arreglo para llevar la nueva secuencia  $B$ , no es necesario reubicar el arreglo y toda inserción al final de  $B$  toma tiempo  $\Theta(1)$
- Esta estrategia no necesita memoria adicional... o más precisamente, necesita  $\mathcal{O}(1)$  de memoria adicional
- Los algoritmos que tienen esta propiedad se conocen como algoritmos **in place**

# Sumario

Introducción

Correctitud de algoritmos

Selection Sort

**Cierre**

# Objetivos de la clase

- ☐ Comprender el concepto de ordenación como operación abstracta
- ☐ Interpretar de forma intuitiva un algoritmo en pseudocódigo
- ☐ Demostrar correctitud de un algoritmo dado
- ☐ Determinar complejidad del algoritmo de ordenación Selection Sort
- ☐ Distinguir ventajas comparativas de EDD's en las operaciones empleadas en Selection Sort