La pequeña ovejería

- Don Juan tiene 600 ovejas
- Cada oveja tiene asignado un número único
- El rebaño está dividido en 4 lotes de 150 ovejas c/u
- Para cada lote, sus números están registrados en una hoja
- Los números de un lote están en cualquier orden


TAIWAN
4160 T
4022 T
3117 m/T
3058 T
5193 S
2103 R
288 B
s/n T
3150 T
4254 m
5008 m/S
246 m
3094 T
2055 m
229 T
3003 T
6004 T
2140 m
263 m
2209 T
296 m
3091 m
2098 T
2125 B
4195 T
6106 m
6191 T
2006 m
267 m/n
2084 m/T
3031 T
2039 T
6063 m/n
s/n m/T
248 m
283 m
3137 m
2078 T
252 m
4083 m
2009 m
3014 T
2036 T
2241 m
6247 T
s/n T
2117 m
4177 m/T
268 m
3195 T
2199 M
6175 T
2181 T
293 T
274 T
5088 M
2063 T
4120 M
2021 T
269 m/n
625 T
251 R
6025 S
3108 T
s/n M
3197 M
3308 T
4163 R
6155 T
4058 M
2167 T
3032 T
6181 M/N
253 M
4114 T
6169 PARDA
6121 PARDA
4021 T
6222 T
2109 m
3180 m
6197 parda
237 B
4206 T
6055 p
s/n p
6154 p/n
3029 m/T

La pequeña ovejería

Don Juan suele encontrar algunas ovejas separadas de sus lotes

¿Cómo puede saber fácilmente a qué lote pertenece una oveja?

Secuencias ordenadas

Una secuencia de números $x_1, \cdots, x_n$ se dice **ordenada** (no decrecientemente) si cumple que $x_1 \leq \cdots \leq x_n$

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

El algoritmo de ordenación de Don Juan

1. De la hoja original, tomar el número más pequeño
2. Tacharlo en la hoja original
3. Escribirlo al final (en el primer espacio disponible) de la hoja nueva
4. Si quedan números en la hoja original, volver al paso 1.

¿Es correcto el algoritmo de Don Juan?

Ahora … a trabajar ustedes

Demuestra que el algoritmo de Don Juan es correcto

Es decir,

- Termina en una cantidad finita de pasos
- Cumple su propósito: **ordena** los datos

El algoritmo selection sort

Para la secuencia inicial de datos, A:

1. Definir una secuencia ordenada, 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 elementos en A, volver a 2.

¿Cuál es la complejidad de selection sort?

Raciocinio para determinar la complejidad de selection sort

Buscar el menor dato en A significa revisar A entero: $O(n)$

Este proceso se hace una vez por cada dato: n veces

La complejidad es entonces $n \cdot O(n) = O(n^2)$

Otra forma de calcular la complejidad de selection sort

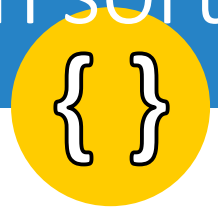
También se puede hacer de manera explícita:

Buscar el mínimo cuesta n, y el siguiente $n-1$, y así:

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

$$T(n) \in O(n^2)$$

Complejidad de memoria de selection sort



Selection Sort se puede hacer en un solo **arreglo**, ya que $|A| + |B| = n$

Eso significa que no necesita nada de memoria adicional

Los algoritmos que hacen esto se conocen como *in place*

Don Juan tiene ahora otro problema

Don Juan quiere cambiar 5 ovejas del lote A al lote B

Necesita actualizar el cambio en ambas hojas

¿Cómo lo hace para no tener que volver a ordenar todo?

Inserción en una lista ordenada

Insertar pocos elementos ordenadamente es … ¿barato?

¿Cómo podemos usar este hecho para ordenar?

El algoritmo *insertionSort*

Para la secuencia inicial de datos, A:

1. Definir una secuencia ordenada, B, inicialmente vacía
2. Tomar el primer dato x de A y sacarlo de A
3. Insertar x en B de manera que B quede ordenado
4. Si quedan elementos en A, volver a 2.

Insertion Sort

Insertion Sort es correcto

Podemos demostrarlo por inducción

Finitud

En cada paso se saca un elemento de A

Cuando no quedan elementos en A, el algoritmo termina

La inserción requiere como máximo recorrer todo B

Como A y B son finitos, el algoritmo termina en tiempo finito

Correctitud

PD: Al terminar el i-ésimo paso, B se encuentra ordenada

Por **inducción** sobre i

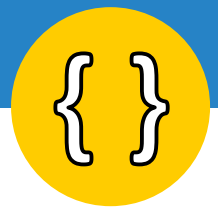
Caso Base: Después del paso 1, B tiene un solo dato $\rightarrow$ B está ordenada

Hipótesis Inductiva: Después del paso i, B está ordenada

Toca el paso $i+1$. Extraemos el primer elemento de A, y lo insertamos ordenadamente en B. Termina el paso $i+1$. Si la **inserción** fue correcta, entonces B está ordenada.

En particular, al terminar el algoritmo después del paso n, B está ordenada.

¿Cómo se hace una inserción?



Depende de la estructura de datos usada para almacenar la lista

Se suele usar **arreglos**, pero también se puede usar **listas ligadas**

En cualquier caso, el algoritmo no necesita memoria adicional

Los dos pasos de la inserción

Primero, hay que buscar donde corresponde insertar el elemento

Luego, hay que llevar a cabo la inserción

¿Cuál es la complejidad usando **arreglos**?

¿Y con **listas ligadas**?

Insertar en un arreglo

El primer paso podemos hacerlo en $O(\log n)$ con búsqueda binaria

Para insertar hay que desplazar todos los elementos, lo que es $O(n)$

Por lo tanto, en **arreglos insertar** es $O(n)$

Insertar en una lista (doblemente) ligada

Para el primer paso es necesario revisar la lista entera, en $O(n)$

Teniendo el nodo donde corresponde insertar, hacerlo es $O(1)$

Por lo tanto, en **listas ligadas insertar** es $O(n)$

Complejidad de *insertionSort*

Es necesario realizar n inserciones

Cada una cuesta $O(n)$ independiente de la estructura

Por lo que la complejidad es $n \cdot O(n) = O(n^2)$