

# TALLER DE ORDENAMIENTO

(SORTING PARA LOS AMIGOS)

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

20 de julio de 2020



¿QUÉ ONDA LA CLASE DE HOY?

¿CUÁLES ERAN LOS ALGORITMOS? ... SORTING...  
EH...



¿CUÁLES ERAN LOS ALGORITMOS? ... SORTING...  
EH...



Primer momento lúdico: “Una imagen vale más que mil diapos”



# SELECTION SORT

## Idea

Seleccionar el mínimo elemento, llamémoslo  $m$ , del contenedor y ponerlo al principio. Después seleccionar el mínimo elemento sin tener en cuenta  $m$  y ponerlo segundo... etc. Se puede hacer in-place.

# SELECTION SORT

## Idea

Seleccionar el mínimo elemento, llamémoslo  $m$ , del contenedor y ponerlo al principio. Después seleccionar el mínimo elemento sin tener en cuenta  $m$  y ponerlo segundo... etc. Se puede hacer in-place.

## Invariante

# SELECTION SORT

## Idea

Seleccionar el mínimo elemento, llamémoslo  $m$ , del contenedor y ponerlo al principio. Después seleccionar el mínimo elemento sin tener en cuenta  $m$  y ponerlo segundo... etc. Se puede hacer in-place.

## Invariante

- ▶ En la  $k$ -ésima iteración, los primeros  $k$  elementos ya están ordenados en su posición final.
- ▶ El arreglo es una permutación del arreglo original.



# SELECTION SORT

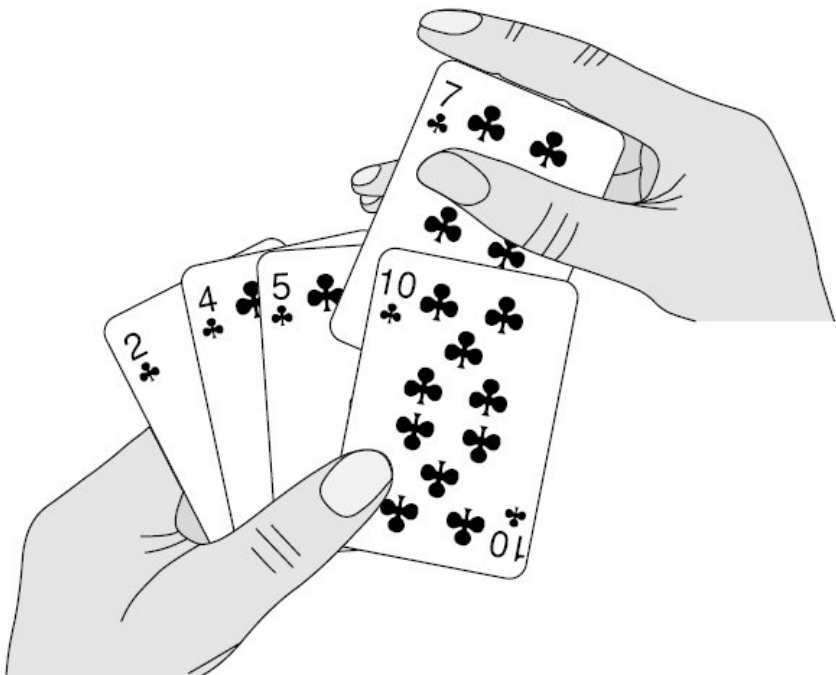
## Idea

Seleccionar el mínimo elemento, llamémoslo  $m$ , del contenedor y ponerlo al principio. Después seleccionar el mínimo elemento sin tener en cuenta  $m$  y ponerlo segundo... etc. Se puede hacer in-place.

## Invariante

- ▶ En la  $k$ -ésima iteración, los primeros  $k$  elementos ya están ordenados en su posición final.
- ▶ El arreglo es una permutación del arreglo original.

**Complejidad:**  $\mathcal{O}(n^2)$



# INSERTION SORT

## Idea

Insertar el  $i$ -ésimo elemento del contenedor en la posición que le corresponde en el arreglo  $0..i$ . Se puede hacer in-place.

# INSERTION SORT

## Idea

Insertar el  $i$ -ésimo elemento del contenedor en la posición que le corresponde en el arreglo  $0..i$ . Se puede hacer in-place.

## Invariante

- ▶ Los elementos del arreglo de  $0..i$  son los mismos que en el arreglo original, pero están ordenados.
- ▶ El arreglo es una permutación del arreglo original.

# INSERTION SORT

## Idea

Insertar el  $i$ -ésimo elemento del contenedor en la posición que le corresponde en el arreglo  $0..i$ . Se puede hacer in-place.

## Invariante

- ▶ Los elementos del arreglo de  $0..i$  son los mismos que en el arreglo original, pero están ordenados.
- ▶ El arreglo es una permutación del arreglo original.

**Complejidad:**  $\mathcal{O}(n^2)$



# MERGE SORT

## Idea

Para ordenar un arreglo, lo parto en dos mitades  $A_1$  y  $A_2$ . Ordeno las dos mitades  $A_1$  y  $A_2$  y después las junto en  $O(n)$ . Para ordenar a  $A_1$  y a  $A_2$  me llamo recursivamente.

Se puede hacer in-place, pero a los efectos de la materia es una complicación innecesaria.

# MERGE SORT

## Idea

Para ordenar un arreglo, lo parto en dos mitades  $A_1$  y  $A_2$ . Ordeno las dos mitades  $A_1$  y  $A_2$  y después las junto en  $O(n)$ . Para ordenar a  $A_1$  y a  $A_2$  me llamo recursivamente.

Se puede hacer in-place, pero a los efectos de la materia es una complicación innecesaria.

**Complejidad:**  $\mathcal{O}(n \log n)$





# HEAP SORT

## Idea

Transformar el arreglo en un heap usando el algoritmo de Floyd en  $O(n)$ . Una vez construido el heap, el máximo, llamémoslo  $M$ , se obtiene en  $O(1)$ , y remover el máximo es  $O(\log(n))$ . Al remover el máximo del heap, queda un espacio vacío en el arreglo. Llenar ese espacio vacío, que está al final de l arreglo con  $M$ . O sea, hacélo in-place.

# HEAP SORT

## Idea

Transformar el arreglo en un heap usando el algoritmo de Floyd en  $O(n)$ . Una vez construido el heap, el máximo, llamémoslo  $M$ , se obtiene en  $O(1)$ , y remover el máximo es  $O(\log(n))$ . Al remover el máximo del heap, queda un espacio vacío en el arreglo. Llenar ese espacio vacío, que está al final de l arreglo con  $M$ . O sea, hacélo in-place.

## Invariante

- ▶ En la  $i$ -ésima iteración, los primeros  $n-i$  elementos del arreglo conforman un heap y los últimos  $i$  elementos están ordenados
- ▶ El arreglo es una permutación del arreglo original

# HEAP SORT

## Idea

Transformar el arreglo en un heap usando el algoritmo de Floyd en  $O(n)$ . Una vez construido el heap, el máximo, llamémoslo  $M$ , se obtiene en  $O(1)$ , y remover el máximo es  $O(\log(n))$ . Al remover el máximo del heap, queda un espacio vacío en el arreglo. Llenar ese espacio vacío, que está al final de l arreglo con  $M$ . O sea, hacélo in-place.

## Invariante

- ▶ En la  $i$ -ésima iteración, los primeros  $n-i$  elementos del arreglo conforman un heap y los últimos  $i$  elementos están ordenados
- ▶ El arreglo es una permutación del arreglo original

**Complejidad:**  $O(n \log n)$



## Idea

Elijo un elemento del arreglo, al que voy a llamar pivote. Pongo a todos los elementos menores al pivote a la izquierda y a los elementos mayores al pivote a la derecha. Me llamo recursivamente con izquierda y con derecha.

Tips: Fijate que podes elegir el pivote de muchas formas: el primer elemento, la mediana, random.

## Idea

Elijo un elemento del arreglo, al que voy a llamar pivote. Pongo a todos los elementos menores al pivote a la izquierda y a los elementos mayores al pivote a la derecha. Me llamo recursivamente con izquierda y con derecha.

Tips: Fijate que podes elegir el pivote de muchas formas: el primer elemento, la mediana, random.

**Complejidad peor caso:**  $\mathcal{O}(n^2)$

**Complejidad caso promedio:**  $\mathcal{O}(n \log n)$

# IMPLEMENTACIÓN DE ALGORITMOS

- ▶ En este taller los vamos a invitar a que implementen algunos de los algoritmos de ordenamiento que vieron en la teórica.



# IMPLEMENTACIÓN DE ALGORITMOS

- ▶ En este taller los vamos a invitar a que implementen algunos de los algoritmos de ordenamiento que vieron en la teórica.
- ▶ Cada grupo tendrá que implementar un algoritmo de complejidad  $\mathcal{O}(n^2)$  y uno de complejidad  $\mathcal{O}(n \log n)$ .

# IMPLEMENTACIÓN DE ALGORITMOS

- ▶ En este taller los vamos a invitar a que implementen algunos de los algoritmos de ordenamiento que vieron en la teórica.
- ▶ Cada grupo tendrá que implementar un algoritmo de complejidad  $\mathcal{O}(n^2)$  y uno de complejidad  $\mathcal{O}(n \log n)$ .
- ▶ Al final del taller las diferentes implementaciones de cada uno de los algoritmos competirán entre sí.

# IMPLEMENTACIÓN DE ALGORITMOS

- ▶ En este taller los vamos a invitar a que implementen algunos de los algoritmos de ordenamiento que vieron en la teórica.
- ▶ Cada grupo tendrá que implementar un algoritmo de complejidad  $\mathcal{O}(n^2)$  y uno de complejidad  $\mathcal{O}(n \log n)$ .
- ▶ Al final del taller las diferentes implementaciones de cada uno de los algoritmos competirán entre sí.
- ▶ Vamos a usar C++ como en toda la materia, no? NO! Vamos a usar Python!

# PARÁ PARÁ... DIJISTE PYTHON?



No se preocupen, hay un apunte con todo lo que necesitan saber.  
Ademas, ahora hacemos una DEMO!

Veamos un poco como funciona



# ARMADO DE GRUPOS

- ▶ Cada grupo puede estar compuesto de 1 a 3 personas, y deberán elegir un nombre único y representativo (por favor no usar símbolos ni acentos).
- ▶ En la página de la materia en 'Clases de laboratorio' → 'Taller de Sorting' un archivo comprimido llamado 'material\_alumnos.zip'.
- ▶ Dejar el presentar solamente en los algoritmos que implementen
  - ▶ algorithms.py
  - ▶ test.py
  - ▶ list.py
  - ▶ list\_algorithms.py

# IMPLEMENTACIÓN: ALGORITHMS.PY

- ▶ Deben escribir sus implementaciones en Python en el archivo 'algorithms.py'.

## EJEMPLO

```
from list_algorithms import presentar
import random
grupo = 'grupo_1'

@presentar
def insertion_sort (a):
    return a

@presentar
def quicksort (a):
    return a
```

# COMO CORRERLO POR CONSOLA: TEST.PY

**Help:** \$ python test.py --help

**Uso:** \$ test.py algoritmo [opciones]

Opciones:

- ▶ **-h, - -help** muestra las opciones
- ▶ **-l, - -listita** ejecuta tu algoritmo con una lista chiquita
- ▶ **-L, - -listota** ejecuta tu algoritmo con una lista grandota
- ▶ **-c, - -lcustom** ejecuta tu algoritmo con una lista custom pasada por parámetro de la forma: [3, 12, 414, 12]
- ▶ **-e, - -estimarconstantes** Si tu algoritmo es  $n * \log(n)$  o  $n^2$ , “trata” de estimar el orden de complejidad.

**Ejemplo:** python test.py quicksort -l



# YA TERMINÉ

Tienen que mandar **únicamente** el archivo **'algorithms.py'** a:  
**algo2.dc+tallersorting@gmail.com**

# PREGUNTAS???



# ALGORITMOS... EN SUS MARCAS... LISTOS... YA!



ó



---

Las imágenes son de carácter ilustrativo.