



POLITÉCNICO COLOMBIANO  
JAIME ISAZA CADAVID  
INSTITUCIÓN UNIVERSITARIA

Educación para  
*vivir mejor*

# Algoritmos y Programación 4

Clase 11

Algoritmos de ordenamiento (Parte 1)



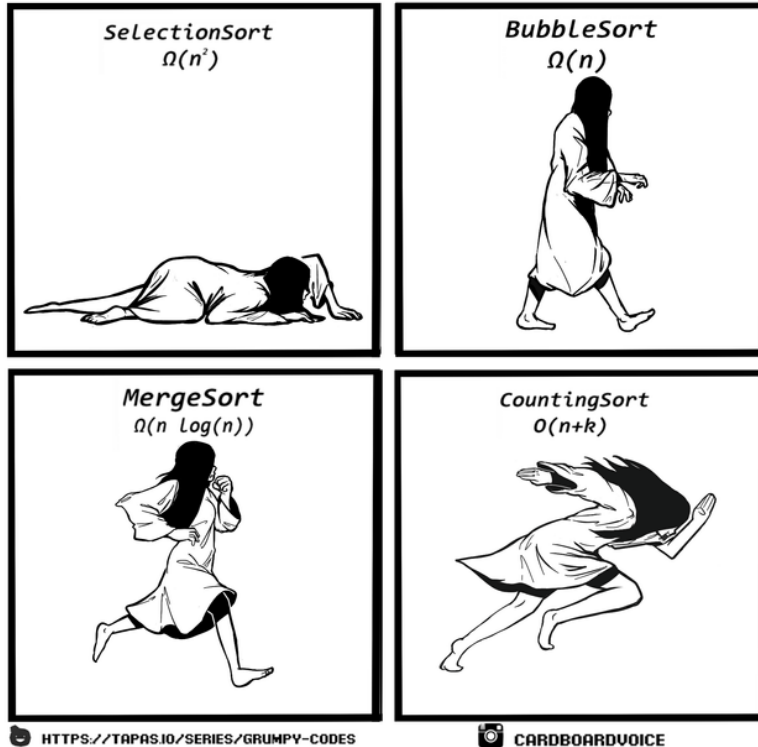
GOBERNACIÓN DE ANTIOQUIA



VIGILADA MINEDUCACIÓN

# Análisis de desempeño

## "SORTING ALGORITHMS"



## INEFFECTIVE SORTS

```

DEFINE HALFHEARTEDMERGESORT(LIST):
  IF LENGTH(LIST) < 2:
    RETURN LIST
  PIVOT = INT(LENGTH(LIST) / 2)
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
  // UMMMMM
  RETURN [A, B] // HERE. SORRY.
    
```

```

DEFINE FASTBOGOSORT(LIST):
  // AN OPTIMIZED BOGOSORT
  // RUNS IN O(N LOG N)
  FOR N FROM 1 TO LOG(LENGTH(LIST)):
    SHUFFLE(LIST):
    IF ISSORTED(LIST):
      RETURN LIST
  RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
    
```

```

DEFINE JOBININTERVIEWQUICKSORT(LIST):
  OK SO YOU CHOOSE A PIVOT
  THEN DIVIDE THE LIST IN HALF
  FOR EACH HALF:
    CHECK TO SEE IF IT'S SORTED
    NO, WAIT, IT DOESN'T MATTER
    COMPARE EACH ELEMENT TO THE PIVOT
    THE BIGGER ONES GO IN A NEW LIST
    THE EQUAL ONES GO INTO, UH
    THE SECOND LIST FROM BEFORE
    HANG ON, LET ME NAME THE LISTS
    THIS IS LIST A
    THE NEW ONE IS LIST B
    PUT THE BIG ONES INTO LIST B
    NOW TAKE THE SECOND LIST
    CALL IT LIST, UH, A2
    WHICH ONE WAS THE PIVOT IN?
    SCRATCH ALL THAT
    IT JUST RECURSIVELY CALLS ITSELF
    UNTIL BOTH LISTS ARE EMPTY
    RIGHT?
    NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
    
```

```

DEFINE PANICSORT(LIST):
  IF ISSORTED(LIST):
    RETURN LIST
  FOR N FROM 1 TO 10000:
    PIVOT = RANDOM(0, LENGTH(LIST))
    LIST = LIST[PIVOT:] + LIST[:PIVOT]
  IF ISSORTED(LIST):
    RETURN LIST
  IF ISSORTED(LIST): // THIS CAN'T BE HAPPENING
    RETURN LIST
  IF ISSORTED(LIST): // COME ON COME ON
    RETURN LIST
  // OH JEEZ
  // I'M GONNA BE IN SO MUCH TROUBLE
  LIST = []
  SYSTEM("SHUTDOWN -H +5")
  SYSTEM("RM -RF ./")
  SYSTEM("RM -RF ~/*")
  SYSTEM("RM -RF /")
  SYSTEM("RD /S /Q C:\*") // PORTABILITY
  RETURN [1, 2, 3, 4, 5]
    
```

# Sorting

- Reordenar una arreglo de  $n$  elementos de acuerdo a una clave (*key*).

## Problema:

Ordenar alfabéticamente, de acuerdo a los apellidos, la lista de estudiantes de la escuela de magia Hogwarts.



item →

key →

Last ▾	First	House	Year
Longbottom	Neville	Gryffindor	1998
Weasley	Ron	Gryffindor	1998
Abbott	Hannah	Hufflepuff	1998
Potter	Harry	Gryffindor	1998
Chang	Cho	Ravenclaw	1997
Granger	Hermione	Gryffindor	1998
Malfoy	Draco	Slytherin	1998
Diggory	Cedric	Hufflepuff	1996
Weasley	Ginny	Gryffindor	1999
Parkinson	Pansy	Slytherin	1998

Sombrero  
seleccionador  
[\[link\]](#)



Last ▾	First	House	Year
Abbott	Hannah	Hufflepuff	1998
Chang	Cho	Ravenclaw	1997
Granger	Hermione	Gryffindor	1998
Diggory	Cedric	Hufflepuff	1996
Longbottom	Neville	Gryffindor	1998
Malfoy	Draco	Slytherin	1998
Parkinson	Pansy	Slytherin	1998
Potter	Harry	Gryffindor	1998
Weasley	Ron	Gryffindor	1998
Weasley	Ginny	Gryffindor	1999

↑  
sorted by key

# Sorting

- El **sorting** (**ordenamiento** o **clasificación**) es el proceso fundamental de tomar una colección de elementos que están en un orden aleatorio y reorganizarlos siguiendo un criterio específico.



	Last ▾	First	House	Year
	Abbott	Hannah	Hufflepuff	1998
	Chang	Cho	Ravenclaw	1997
	Granger	Hermione	Gryffindor	1998
	Diggory	Cedric	Hufflepuff	1996
	Longbottom	Neville	Gryffindor	1998
key →	Malfoy	Draco	Slytherin	1998
	Parkinson	Pansy	Slytherin	1998
item →	Potter	Harry	Gryffindor	1998
	Weasley	Ron	Gryffindor	1998
	Weasley	Ginny	Gryffindor	1999

↑  
sorted by key

## Análisis del problema:

- Datos de entrada (desordenados):** La lista de estudiantes tal como llegaron.
- Sorting (el proceso):** Es la acción de reorganizar esa lista.
- Clave (el criterio):** El apellido del estudiante (la columna **Last**).
- Datos de salida (ordenados):** La lista organizada alfabéticamente por apellido.

# Sorting

## Algunas aplicaciones del ordenamiento:

- **Tiendas en línea (E-commerce):** Ordenar los resultados de búsqueda por precio (menor a mayor), valoración (más estrellas) o relevancia.
- **Correo electrónico:** Organizar la bandeja de entrada por fecha (el más reciente primero) o por remitente (alfabéticamente).
- **Aplicaciones de música (Spotify):** Ordenar una biblioteca o playlist por título de la canción, nombre del artista o fecha de lanzamiento.
- **Explorador de archivos:** Clasificar los archivos de una carpeta por nombre (alfabético), fecha de modificación, tamaño o tipo (.pdf, .jpg, etc.).

Video name	Views (billions) ▾
"Despacito" <sup>[23]</sup>	6.96
"Baby Shark Dance" <sup>[28]</sup>	6.55
"Shape of You" <sup>[29]</sup>	4.97
"See You Again" <sup>[30]</sup>	4.72
"Masha and the Bear – Recipe for	4.33
"Uptown Funk" <sup>[38]</sup>	3.94

numerical order (descending)

International Departures				
Flight No	Destination	Time	Gate	Remarks
CX769	Berlin	7:50	A-11	Gate closing
QF2474	London	7:50	A-12	Gate closing
BA372	Paris	7:55	B-10	Boarding
A16554	New York	8:00	C-33	Boarding
NL3160	San Francisco	8:00	F-15	Boarding
BA8903	Manchester	8:05	B-12	Gate lounge open
BA700	Los Angeles	8:10	C-12	Check-in open
QF3371	Hong Kong	8:15	F-10	Check-in open
MA4885	Barcelona	8:15	F-12	Check-in at kiosks
CX7221	Copenhagen	8:20	B-22	Check-in at kiosks

chronological order

All Contacts	
Search	
All Contacts	
Ally Kazmucha	E
Amanda	F
Amanda Jozaitis	H
Amanda VanVoorhis	J
Amy Bruemmer	K
Amy M	L
Amy Riehle	M
Andrew Wray	N
Andy Hlynek	O
Anil Kumar	P

lexicographic order



# Tipos básicos de algoritmos de ordenamiento

- Existen diferentes tipos de algoritmos para llevar a cabo ordenamiento.
  - **Basados en comparación:** Son los más comunes. Funcionan comparando dos elementos a la vez para decidir cuál va primero.
  - **No basados en comparación:** Son un tipo especial que no compara elementos entre sí. En lugar de eso, ordenan basándose en propiedades de los propios elementos (como los dígitos de un número). **Radix Sort** es el ejemplo clásico.

Tipo	Algoritmo
Basados en comparación	<ul style="list-style-type: none"><li>• Selection Sort</li><li>• Bubble Sort</li><li>• Insertion Sort</li><li>• Merge Sort</li><li>• Quick Sort</li></ul>
No basados en comparación	<ul style="list-style-type: none"><li>• Radix Sort</li></ul>



# Sorting

- Los algoritmos basados en comparación se pueden clasificar de acuerdo al enfoque empleado para la comparación:
  - **Iterativos:** Usan **bucles** para recorrer la lista y ordenar los elementos paso a paso.
  - **Recursivos:** Usan la estrategia de **divide y vencerás** tomando el problema grande y dividiéndolo en problemas más pequeños hasta que son fáciles de ordenar para luego unir las soluciones las soluciones.

Enfoque	Algoritmo
Iterativos	<ul style="list-style-type: none"><li>• Selection Sort</li><li>• Bubble Sort</li><li>• Insertion Sort</li></ul>
Recursivos	<ul style="list-style-type: none"><li>• Merge Sort</li><li>• Quick Sort</li></ul>



# Bubble sort (Ordenamiento de burbuja)

- **Funcionamiento:** Este método compara repetidamente cada par de elementos adyacentes (**vecinos**). Si están en el orden incorrecto, los intercambia. Repite este proceso (**pasada**) hasta que en una pasada completa no se haga ningún intercambio.
- **Analogía:** El elemento grande es como una burbuja que flota hasta el final de la matriz.



Bubble sort ([link](#))

29	10	14	37	13
----	----	----	----	----

10	14	29	13	37
----	----	----	----	----

10	14	13	29	37
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----



# Bubble sort (Ordenamiento de burbuja)

- **Procedimiento:** Dado un arreglo de  $n$  elementos:
  1. Comparar pares de elementos adyacentes.
  2. Intercambiar los elementos si se encuentran fuera de orden (en el orden incorrecto).
  3. Repetir (el procedimiento) hasta el final del arreglo.
    - (Como resultado) El elemento de mayor valor se ubicará en la última posición.
  4. Reducir  $n$  en uno y volver al Paso 1. (Repetir el proceso excluyendo el último elemento ya ordenado).

```
def bubble_sort(arr):  
    n = len(arr)  
    for i in range(n):  
        swapped = False  
        for j in range(0, n - i - 1):  
            if arr[j] > arr[j + 1]:  
                arr[j], arr[j + 1] = arr[j + 1], arr[j]  
                swapped = True  
        if not swapped:  
            break  
    return arr
```



# Bubble sort (Ordenamiento de burbuja)

- **Ejemplo:** Ordenar el array [29, 10, 14, 37, 13]

29	10	14	37	13
----	----	----	----	----

10	14	29	13	37
----	----	----	----	----

10	14	13	29	37
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

10	29	14	37	13
----	----	----	----	----

10	14	29	13	37
----	----	----	----	----

10	14	13	29	37
----	----	----	----	----

10	14	29	37	13
----	----	----	----	----

10	14	29	13	37
----	----	----	----	----

10	14	29	37	13
----	----	----	----	----

10	14	29	13	37
----	----	----	----	----

10	14	13	29	37
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
```

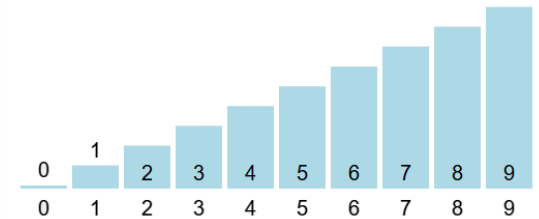
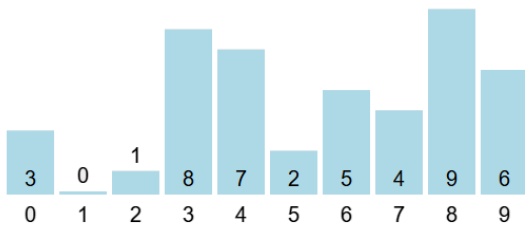
arr	i	j	arr[j] > arr[j+1]	swapped
[29, 10, 14, 37, 13]	...	...	...	...

Prueba de escritorio ([link](#))

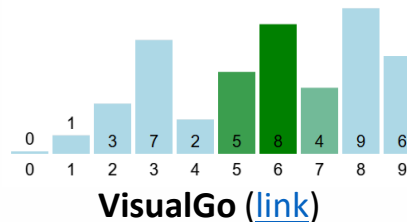


# Bubble sort (Ordenamiento de burbuja)

- **Ejemplo:** Ordenar el array  $[3, 0, 1, 8, 7, 2, 5, 4, 9, 6]$

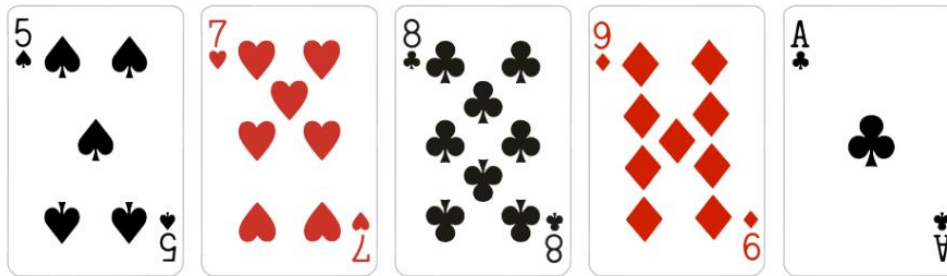


→ **Bubble sort** →



# Selection sort (Ordenamiento por selección)

- **Funcionamiento:** Este algoritmo, en cada iteración, busca el elemento más pequeño (en orden ascendente) del subarreglo no ordenado y lo intercambia con el primer elemento de dicho subarreglo. El proceso se repite reduciendo el rango de búsqueda hasta que todo el arreglo queda ordenado.
- **Analogía:** En un mazo de cartas (en desorden), se selecciona la carta más baja del mazo desordenado y se mueve al lado ordenado. El proceso se repite hasta ordenar todo el mazo.



Generador de barajas ([link](#))

5	7	8	9	1
---	---	---	---	---

1	7	8	9	5
---	---	---	---	---

1	5	8	9	7
---	---	---	---	---

1	5	7	9	8
---	---	---	---	---

1	5	7	8	9
---	---	---	---	---

1	5	7	8	9
---	---	---	---	---



# Selection sort (Ordenamiento por selección)

- **Procedimiento:** Dado un arreglo de  $n$  elementos:
  1. Buscar el elemento mínimo en la parte no ordenada
  2. Intercambiar con el elemento de la primera posición no ordenada.
  3. El inicio del arreglo se va llenando con los elementos más pequeños.

```
def selection_sort(arr):  
    n = len(arr)  
    for i in range(n):  
        min_idx = i  
        for j in range(i + 1, n):  
            if arr[j] < arr[min_idx]:  
                min_idx = j  
            arr[i], arr[min_idx] = arr[min_idx], arr[i]  
    return arr
```



# Selection sort (Ordenamiento por selección)

- **Ejemplo:** Ordenar el array [29, 10, 14, 37, 13]

29	10	14	37	<u>13</u>
----	----	----	----	-----------

10	29	14	37	<u>13</u>
----	----	----	----	-----------

10	13	<u>14</u>	37	29
----	----	-----------	----	----

10	13	14	37	<u>29</u>
----	----	----	----	-----------

29	<u>10</u>	14	37	13
----	-----------	----	----	----

10	29	14	37	<u>13</u>
----	----	----	----	-----------

10	13	<u>14</u>	37	29
----	----	-----------	----	----

29	<u>10</u>	14	37	13
----	-----------	----	----	----

10	29	14	37	<u>13</u>
----	----	----	----	-----------

29	<u>10</u>	14	37	13
----	-----------	----	----	----

10	29	14	37	13
----	----	----	----	----

10	13	14	37	29
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n-1):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

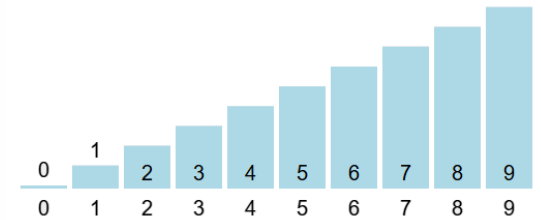
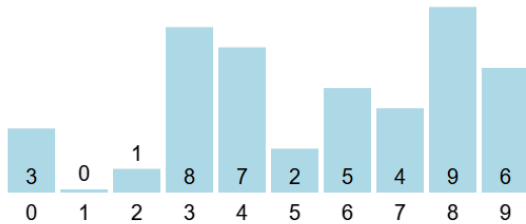
arr	i	j	min_idx	swapped
[29, 10, 14, 37, 13]	...	...	...	...

Prueba de escritorio ([link](#))

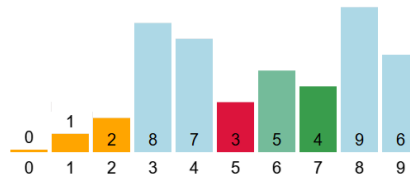


# Selection sort (Ordenamiento por selección)

- **Ejemplo:** Ordenar el array  $[3, 0, 1, 8, 7, 2, 5, 4, 9, 6]$



→ **Selection sort** →

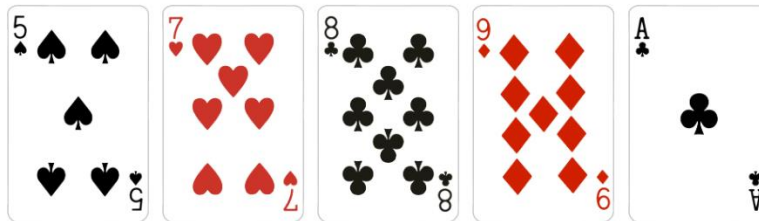


VisualGo ([link](#))



# Insertion sort (Ordenamiento por inserción)

- **Funcionamiento:** Este algoritmo divide el arreglo en una parte ordenada (a la izquierda) y una no ordenada. En cada iteración, toma el primer elemento de la parte no ordenada y lo "inserta" en su lugar correcto dentro de la parte ordenada, desplazando los elementos mayores hacia la derecha.
- **Analogía:** En un mazo desordenado de cartas, se van tomando una por una y colocándolas en el lugar donde corresponden según su valor.



5	7	8	9	1
---	---	---	---	---

5	7	8	9	1
---	---	---	---	---

5	7	8	9	1
---	---	---	---	---

5	7	8	9	1
---	---	---	---	---

1	5	7	8	9
---	---	---	---	---

1	5	7	8	9
---	---	---	---	---



# Insertion sort (Ordenamiento por inserción)

- **Procedimiento:** Dado un arreglo de  $n$  elementos:
  1. Tomar el siguiente elemento de la parte no ordenada.
  2. Compararlo con los elementos de la parte ordenada (a su izquierda).
  3. Desplazar hacia la derecha los elementos mayores para dejar espacio.
  4. Insertar el elemento en la posición correcta dentro de la parte ordenada.
  5. Repetir hasta que todo el arreglo esté ordenado.

```
def insertion_sort(arr):  
    for i in range(1, len(arr)):  
        key = arr[i]  
        j = i - 1  
        while j >= 0 and key < arr[j]:  
            arr[j + 1] = arr[j]  
            j -= 1  
        arr[j + 1] = key  
    return arr
```



# Insertion sort (Ordenamiento por inserción)

- **Ejemplo:** Ordenar el array [29, 10, 14, 37, 13]

29	<u>10</u>	14	37	13
<u>10</u>	29	14	37	13

10	29	<u>14</u>	37	13
10	<u>14</u>	29	37	13

10	14	29	<u>37</u>	13
----	----	----	-----------	----

10	14	29	37	<u>13</u>
----	----	----	----	-----------

10	14	29	<u>13</u>	37
----	----	----	-----------	----

10	14	<u>13</u>	29	37
----	----	-----------	----	----

10	<u>13</u>	14	29	37
----	-----------	----	----	----

10	29	14	37	13
----	----	----	----	----

10	14	29	37	13
----	----	----	----	----

10	14	29	37	13
----	----	----	----	----

10	13	14	29	37
----	----	----	----	----

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
```

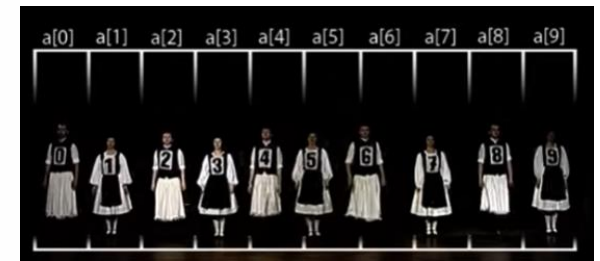
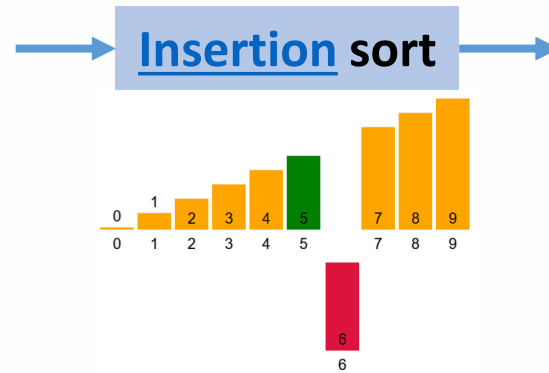
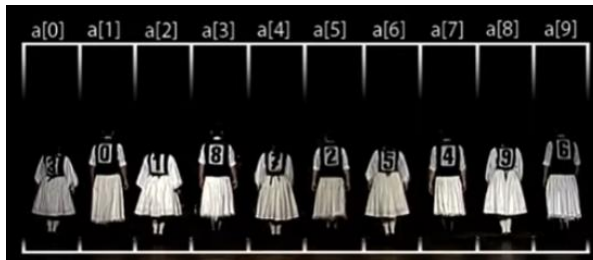
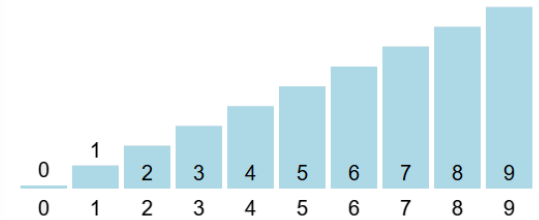
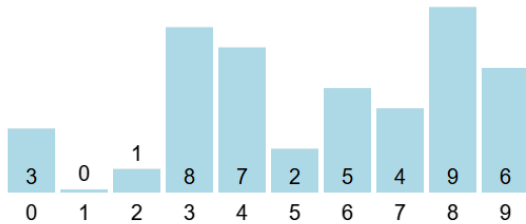
arr	i	j	key	shift
[29, 10, 14, 37, 13]	...	...	...	...

Prueba de escritorio ([link](#))



# Insertion sort (Ordenamiento por inserción)

- **Ejemplo:** Ordenar el array  $[3, 0, 1, 8, 7, 2, 5, 4, 9, 6]$



VisualGo ([link](#))



# Referencias

## Referencias

- Este material ha sido adaptado de las presentaciones disponibles en:
  - <https://introcs.cs.princeton.edu/python/>
  - <https://algs4.cs.princeton.edu/21elementary/>
  - <https://introcs.cs.princeton.edu/python/42sort/>
  - <https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1218/>
  - <https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1252/>
  - <https://web.stanford.edu/class/archive/cs/cs106a/cs106a.1204/>
  - <https://stanford-cs161.github.io/winter2025/>
  - <https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1138/>
  - [https://www.explainxkcd.com/wiki/index.php/3026: Linear Sort](https://www.explainxkcd.com/wiki/index.php/3026:_Linear_Sort)
  - <https://www.cs-comics.com/algorithms-page-1/>
  - <https://www.cs.cmu.edu/~15122/>



# Referencias

## Referencias

- Este material ha sido adaptado de las presentaciones disponibles en:
  - <https://www.cs.cmu.edu/~tcortina/15-121sp10/>
  - <https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-spring-2020/>
  - <https://ocw.mit.edu/courses/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/>
  - <https://debuggi.ng/25wi/>





POLITÉCNICO COLOMBIANO  
JAIME ISAZA CADAVID  
INSTITUCIÓN UNIVERSITARIA

Educación para  
*vivir mejor*

# Gracias



Politécnico Colombiano Jaime Isaza Cadavid



@PolitecnicoJIC



GOBERNACIÓN DE ANTIOQUIA



[www.politecnicojic.edu.co](http://www.politecnicojic.edu.co) / Medellín - Apartadó - Rionegro

VIGILADA MINEDUCACIÓN