Análisis de Algoritmos y Estructura de Datos

Algoritmos de búsqueda y ordenamiento

Prof. Violeta Chang C

Semestre 2 – 2023

Algoritmos de búsqueda y ordenamiento

- **Contenidos**:
  - Algoritmos de búsqueda
  - Algoritmos de ordenamiento
- **Objetivos**:
  - Explicar algoritmos de búsqueda y ordenamiento
  - Entender cálculo de complejidad de algoritmos de búsqueda y ordenamiento
  - Realizar trazas de algoritmos de búsqueda y ordenamiento

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Ruta de la sesión

Algoritmos de búsqueda

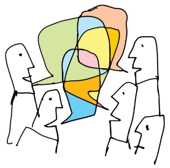
Algoritmos de ordenamiento

Actividad de cierre

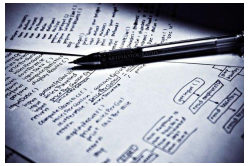
Fechas próximas

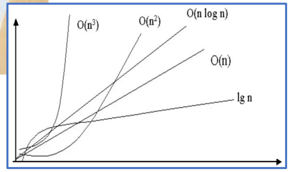
DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Contexto



Solución → Seudocódigo → Implementación





Buscar

Ordenar

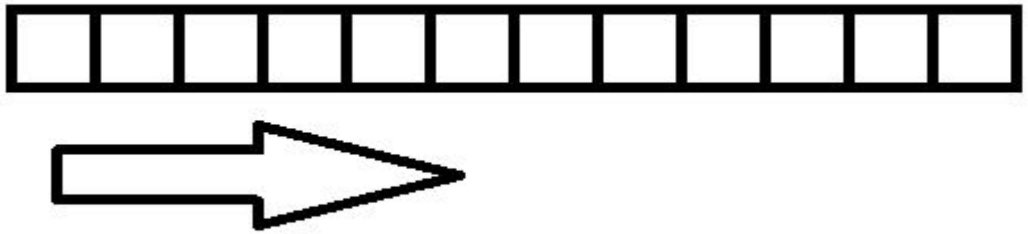
Algoritmos de búsqueda

Conjunto de datos del mismo tipo

Valor exacto que se busca

V/F si lo encuentra o posición en conjunto de datos de valor encontrado

Búsqueda lineal/secuencial



- Se pregunta desde la posición 1 a la n por cada elemento
- Se recorre la lista elemento por elemento hasta encontrarlo

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Búsqueda lineal/secuencial

1	2	3	4	5	6	7	8	9	10	11
C	O	R	O	N	A	V	I	R	U	S
A	A	A	A	A	A					

Búsqueda lineal/secuencial

1	2	3	4	5	6	7	8	9	10	11
C	O	R	O	N	A	V	I	R	U	S
A	A	A	A	A	A					

```
busquedaSecuencial(arreglo,datoBuscado):num
    n←tamaño(arreglo)
    Para i←1 hasta n paso 1
        Si arreglo(i)=datoBuscado entonces
            devolver(i)
    devolver(0)
```

Búsqueda lineal/secuencial

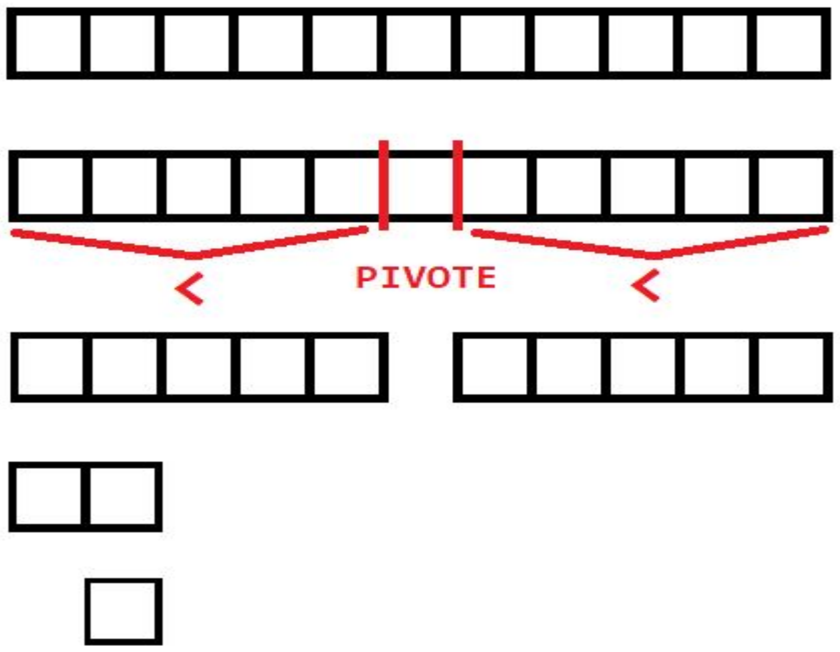
1	2	3	4	5	6	7	8	9	10	11
C	O	R	O	N	A	V	I	R	U	S
A	A	A	A	A	A					

```
busquedaSecuencial(arreglo,datoBuscado):num
    n←tamaño(arreglo)
    Para i←1 hasta n paso 1
        Si arreglo(i)=datoBuscado entonces
            devolver(i)
    devolver(0)
```

O(1)
O(n)
O(1)

O(n)

Búsqueda binaria



- Datos ordenados: elementos de la izquierda de un elemento son menores y a la derecha son mayores
- En la búsqueda se descarta parte de los datos

Búsqueda binaria

```
busquedaBinaria(arreglo,inicio, final,datoBuscado):num
    centro←piso((inicio+final)/2)
    Si arreglo(centro)=datoBuscado entonces
        devolver(centro)
    sino
        Si arreglo(centro)>datoBuscado entonces
            devolver(busquedaBinaria(arreglo,inicio,centro-1,datoBuscado))
        sino
            devolver(busquedaBinaria(arreglo,centro+1,final,datoBuscado))
```

Búsqueda binaria

```
busquedaBinaria(arreglo,inicio, final,datoBuscado):num
    centro←piso((inicio+final)/2)
    Si arreglo(centro)=datoBuscado entonces
        devolver(centro)
    sino
        Si arreglo(centro)>datoBuscado entonces
            devolver(busquedaBinaria(arreglo,inicio,centro-1,datoBuscado))
        sino
            devolver(busquedaBinaria(arreglo,centro+1,final,datoBuscado))
```

$$T(n) = T(n/2) + O(1)$$

Búsqueda binaria

```
busquedaBinaria(arreglo,inicio, final,datoBuscado):num
    centro←piso((inicio+final)/2)
    Si arreglo(centro)=datoBuscado entonces
        devolver(centro)
    sino
        Si arreglo(centro)>datoBuscado entonces
            devolver(busquedaBinaria(arreglo,inicio,centro-1,datoBuscado))
        sino
            devolver(busquedaBinaria(arreglo,centro+1,final,datoBuscado))
```

$$T(n) = T(n/2) + O(1)$$

- $T(n) = aT\left(\frac{n}{b}\right) + O(n^k), si\ n \geq b$

$$O(\ ) = \begin{cases} n^k & si\ a < b^k \\ n^k \log_b n & si\ a = b^k \\ n^{\log_b a} & si\ a > b^k \end{cases}$$

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Búsqueda binaria

```
busquedaBinaria(arreglo,inicio, final,datoBuscado):num
    centro←piso((inicio+final)/2)
    Si arreglo(centro)=datoBuscado entonces
        devolver(centro)
    sino
        Si arreglo(centro)>datoBuscado entonces
            devolver(busquedaBinaria(arreglo,inicio,centro-1,datoBuscado))
        sino
            devolver(busquedaBinaria(arreglo,centro+1,final,datoBuscado))
```

$$T(n) = T(n/2) + O(1)$$

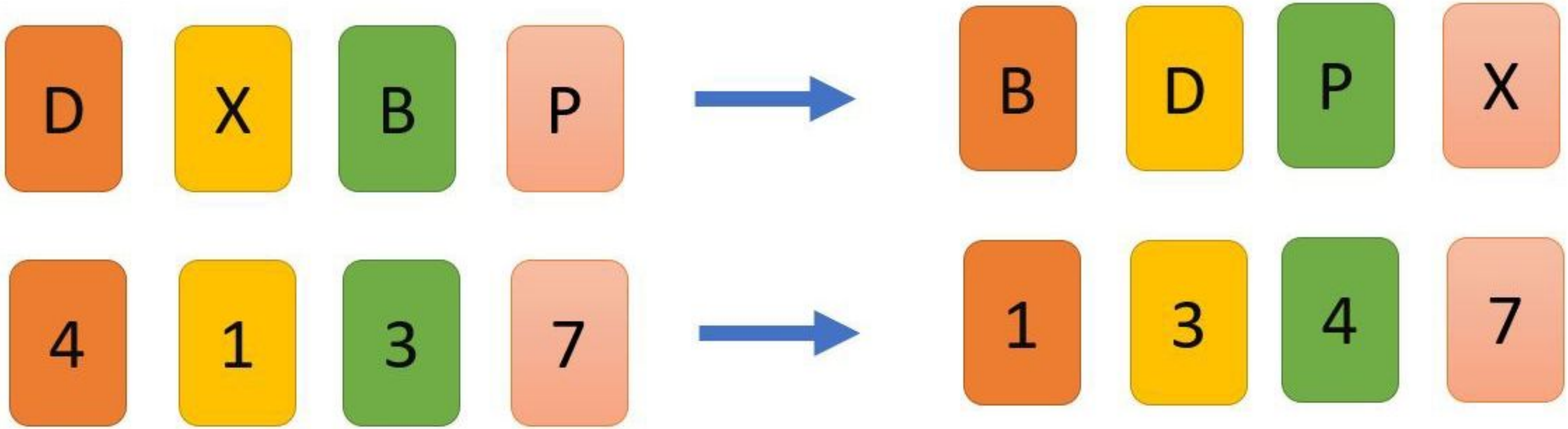
- $T(n) = aT\left(\frac{n}{b}\right) + O(n^k), si\ n \geq b$

$$O(\ ) = \begin{cases} n^k & si\ a < b^k \\ n^k \log_b n & si\ a = b^k \\ n^{\log_b a} & si\ a > b^k \end{cases} \qquad \Rightarrow O(log_2 n)$$

Algoritmos de búsqueda

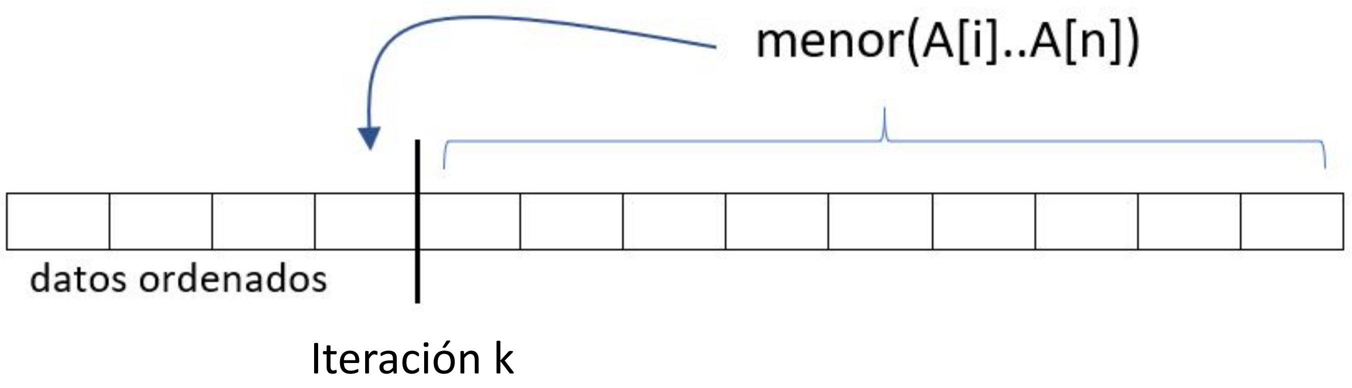
- Si los datos están ordenados, ¿hay alguna ventaja para usar la búsqueda lineal?
- Si los datos no están ordenados, ¿se puede usar búsqueda binaria?
- Si los datos están ordenados por nombre, ¿se puede utilizar búsqueda binaria por RUT?
- La mejor complejidad revisada es logarítmica, ¿será posible hacer búsqueda en orden O(1)?

Algoritmos de ordenamiento



Ordenamiento por selección

- **Selecciona** el elemento que va en la posición *i-ésima* entre los elementos restantes
- En la iteración *k* del algoritmo, se tienen los primeros *k* elementos ordenados



Intuitivo y más usado para arreglos pequeños
Se recorre el arreglo buscando el mínimo

Ordenamiento por selección

- Ejemplo de traza

1	2	3	4	5	6	
5	8	7	1	3	6	*i=1*
1	8	7	5	3	6	*i=2*
1	3	7	5	8	6	*i=3*
1	3	5	7	8	6	*i=4*
1	3	5	6	8	7	*i=5*
1	3	5	6	7	8	*i=6*

Ordenamiento por selección

```
ordenamientoSeleccion(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←1 hasta n-1 paso -1
        indiceMejor←i
        valorMejor←arregloA(i)
        Para j←i+1 hasta n paso 1
            Si arregloA(j)<valorMejor entonces
                indiceMejor←j
                valorMejor←arregloA(j)
        Si indiceMejor<>i entonces
            arregloA←intercambiar(arregloA,i,indiceMejor)
    devolver(arregloA)
```

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Ordenamiento por selección

```
ordenamientoSeleccion(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←1 hasta n-1 paso -1
        indiceMejor←i
        valorMejor←arregloA(i)
        Para j←i+1 hasta n paso 1
            Si arregloA(j)<valorMejor entonces
                indiceMejor←j
                valorMejor←arregloA(j)
        Si indiceMejor<>i entonces
            arregloA←intercambiar(arregloA,i,indiceMejor)
    devolver(arregloA)
```

$O(1)$

$O(n^2)$

$O(1)$

Ordenamiento por selección

```
ordenamientoSeleccion(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←1 hasta n-1 paso -1
        indiceMejor←i
        valorMejor←arregloA(i)
        Para j←i+1 hasta n paso 1
            Si arregloA(j)<valorMejor entonces
                indiceMejor←j
                valorMejor←arregloA(j)
        Si indiceMejor<>i entonces
            arregloA←intercambiar(arregloA,i,indiceMejor)
    devolver(arregloA)
```

$O(1)$

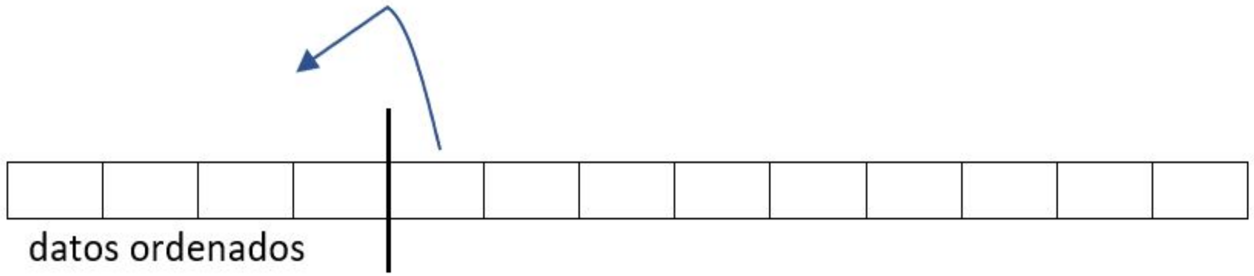
$O(n^2)$

$O(1)$

$\Rightarrow O(n^2)$

Ordenamiento por inserción

- Toma el siguiente elemento en la lista y lo **inserta** en la posición que corresponde en los elementos ordenados
- En la iteración *k* del algoritmo, se tienen *k* elementos ordenados (no necesariamente los *k* primeros)



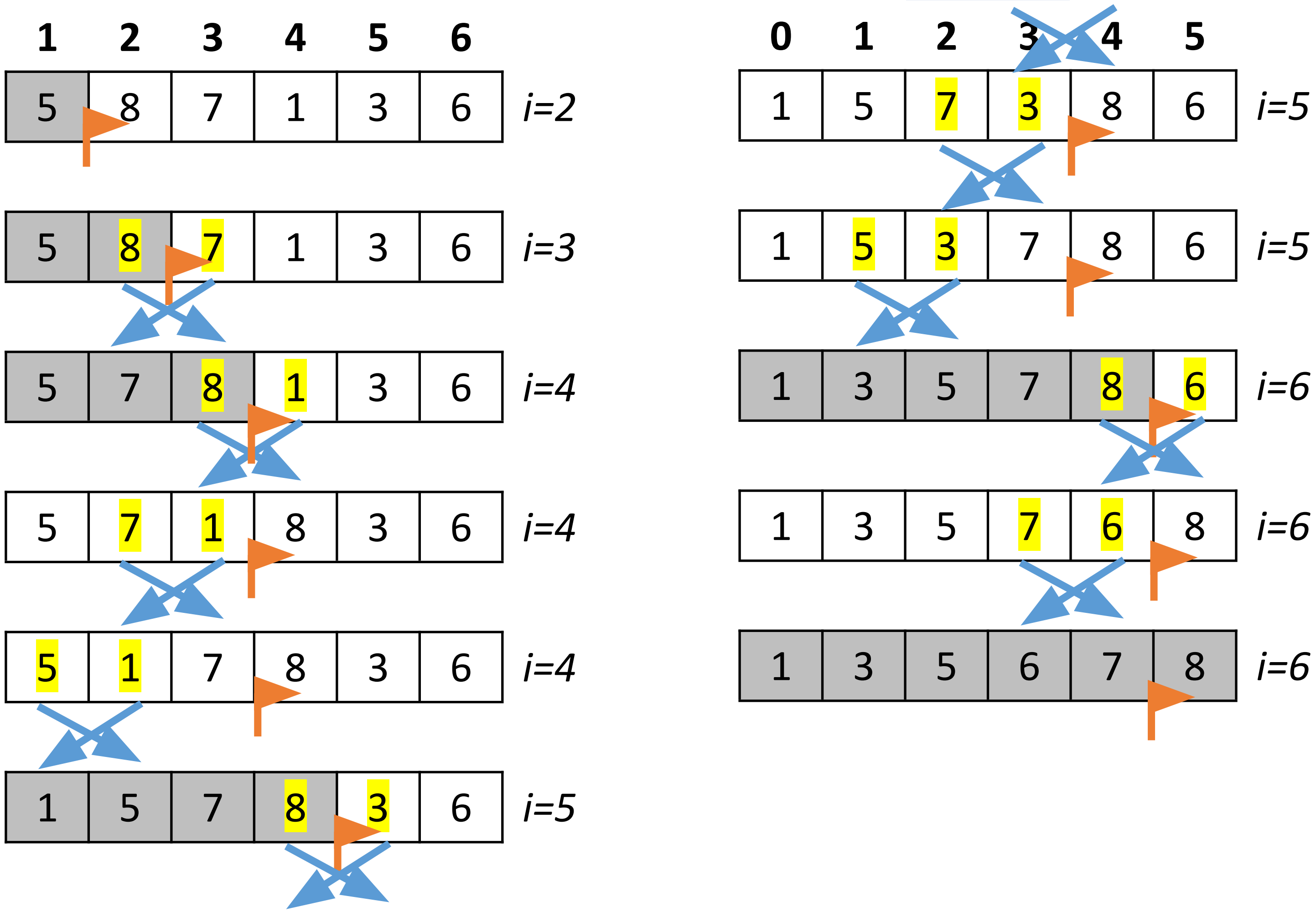
Iteración k

Si se encuentran dos elementos que no estén en orden (a[i]>a[i+1] por ejemplo), se intercambian
Se repite el paso anterior n-1 veces

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Ordenamiento por inserción

- Ejemplo de traza



Ordenamiento por inserción

```
ordenamientoInsercion(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←2 hasta n paso 1
        j←i
        Mientras j>=2 Y arregloA(j)<arregloA(j-1) hacer
            arregloA←intercambiar(arregloA,j,j-1)
            j←j-1
    devolver(arregloA)
```

Ordenamiento por inserción

```
ordenamientoInsercion(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←2 hasta n paso 1
        j←i
        Mientras j>=2 Y arregloA(j)<arregloA(j-1) hacer
            arregloA←intercambiar(arregloA,j,j-1)
            j←j-1
    devolver(arregloA)
```

$O(1)$

$O(n^2)$

$O(1)$

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Ordenamiento por inserción

```
ordenamientoInsercion(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←2 hasta n paso 1
        j←i
        Mientras j>=2 Y arregloA(j)<arregloA(j-1) hacer
            arregloA←intercambiar(arregloA,j,j-1)
            j←j-1
    devolver(arregloA)
```

$O(1)$

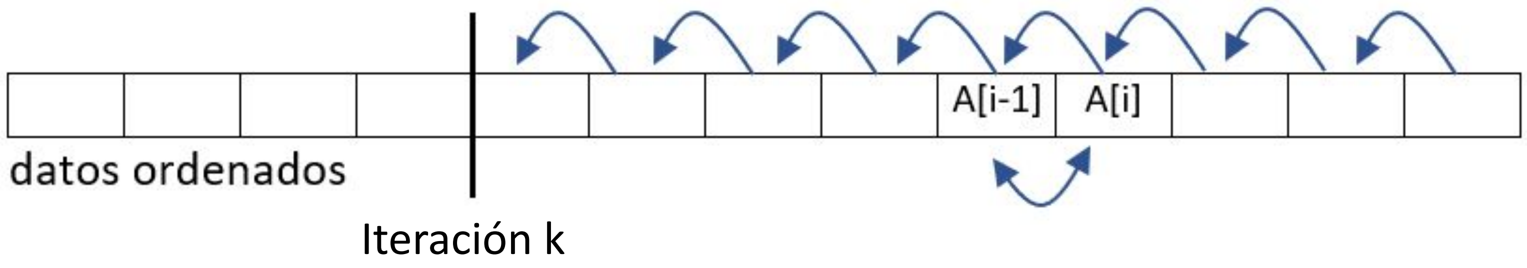
$O(n^2)$

$O(1)$

$\Rightarrow O(n^2)$

Ordenamiento por burbuja

- Itera desde n llevando hasta la i-ésima posición el elemento i, intercambiando los elementos de ser necesario. Desde el final hasta el inicio.
- En la iteración *k* del algoritmo, se tienen los primeros *k* elementos ordenados de acuerdo al criterio de orden



Se realiza un intercambio cada vez que se encuentra una posición en la que no se cumple el criterio de orden

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Ordenamiento por burbuja

- Ejemplo de traza

1	2	3	4	5	6	i	j
5	8	7	1	3	6	*i=1*	*4*
5	8	1	7	3	6	*i=1*	*3*
5	1	8	7	3	6	*i=1*	2
1	5	8	7	3	6	*i=2*	
1	5	8	7	3	6	*i=2*	*5*
1	5	8	3	7	6	*i=2*	*4*

1	2	3	4	5	6	i	j
1	5	3	8	7	6	*i=2*	*3*
1	3	5	8	7	6	*i=3*	
1	3	5	8	7	6	*i=3*	*6*
1	3	5	8	6	7	*i=3*	*5*
1	3	5	6	8	7	*i=4*	
1	3	5	6	8	7	*i=4*	*6*

1	2	3	4	5	6	
1	3	5	6	7	8	*i=4*
1	3	5	6	7	8	*i=5*
1	3	5	6	7	8	*i=5*

Ordenamiento por burbuja

29	45	93	55	48	23	36	25	14	89	40	87
14	29	45	93	55	48	23	36	25	40	89	87
14	23	29	45	93	55	48	25	36	40	87	89
14	23	25	29	45	93	55	48	36	40	87	89
14	23	25	29	36	45	93	55	48	40	87	89
14	23	25	29	36	40	45	93	55	48	87	89
14	23	25	29	36	40	45	48	93	55	87	89
14	23	25	29	36	40	45	48	93	55	87	89
14	23	25	29	36	40	45	48	55	93	87	89
14	23	25	29	36	40	45	48	55	87	93	89
14	23	25	29	36	40	45	48	55	87	89	93
14	23	25	29	36	40	45	48	55	87	89	93

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Ordenamiento por burbuja

```
ordenamientoBurbuja(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←n hasta 1 paso -1
        Para j←1 hasta i-1 paso 1
            Si arregloA(j)>arregloA(j+1) entonces
                arregloA←intercambiar(arregloA,j,j+1)
    devolver(arregloA)
```

Ordenamiento por burbuja

```
ordenamientoBurbuja(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←n hasta 1 paso -1
        Para j←1 hasta i-1 paso 1
            Si arregloA(j)>arregloA(j+1) entonces
                arregloA←intercambiar(arregloA,j,j+1)
    devolver(arregloA)
```

O(1)

O(n^2)

O(1)

Ordenamiento por burbuja

```
ordenamientoBurbuja(arregloA):arreglo
    n←tamaño(arregloA)
    Para i←n hasta 1 paso -1
        Para j←1 hasta i-1 paso 1
            Si arregloA(j)>arregloA(j+1) entonces
                arregloA←intercambiar(arregloA,j,j+1)
    devolver(arregloA)
```

$\}$ O(1)

$\}$ O(n^2)

$\}$ O(1)

$\Rightarrow O(n^2)$

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

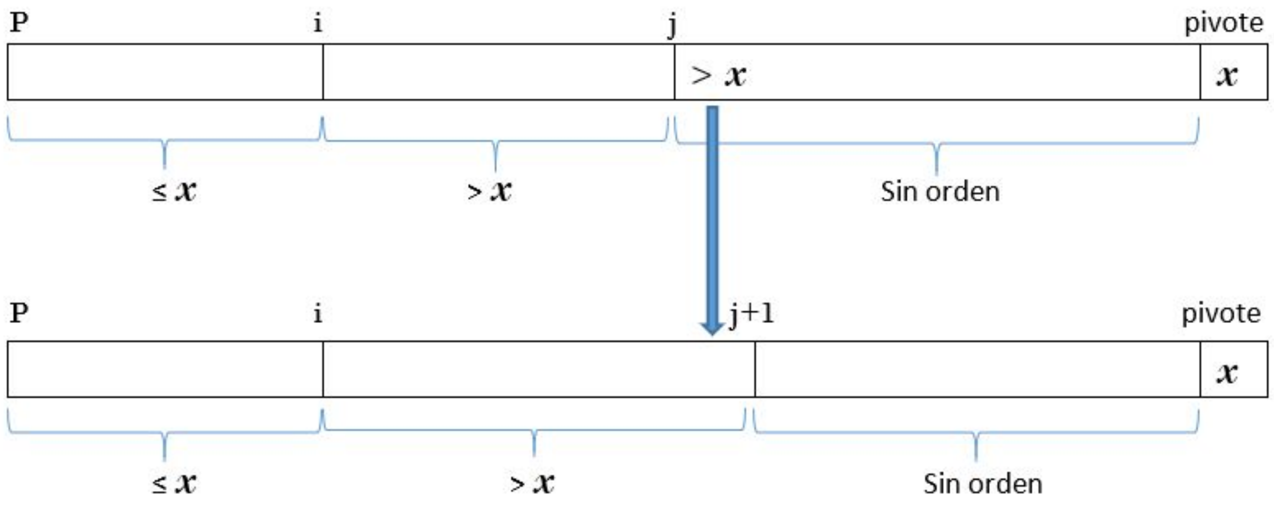
Ordenamiento rápido (Quicksort)

- **Particiona** el problema ordenando alrededor de un pivote
- Aplica el algoritmo de manera recursiva sobre cada mitad del arreglo alrededor del pivote (subarreglo situado a cada lado del pivote). El caso base es cuando se tienen arreglos con menos de dos elementos.
- Al final de cada iteración, el algoritmo tiene dividido los elementos entre un conjunto mayor y un conjunto menor, además de un conjunto de datos a los que no se conoce el orden, que son los que no se han revisado.
- **La elección del pivote** puede afectar el rendimiento del algoritmo. No hay un único mecanismo de elección. Si el pivote está justo en la mitad de los datos (datos bien distribuidos) funcionará muy bien

Quicksort: partición del arreglo

- Alternativa 1: el elemento a revisar es **mayor** que el pivote

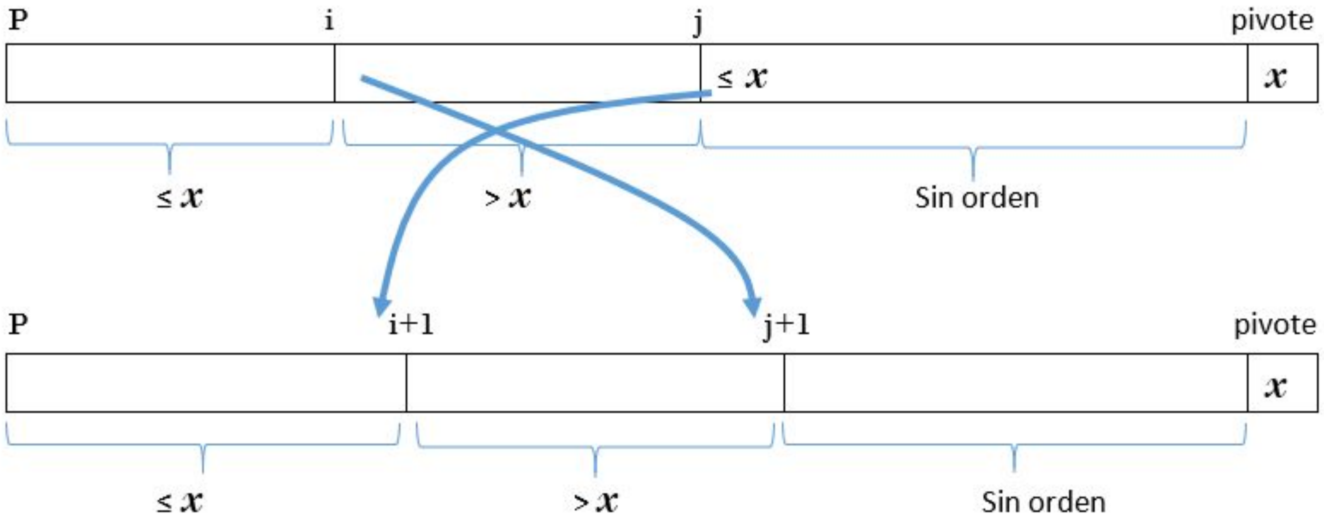
```
particiona(A,p,r): num
x ← A[r]
i ← p - 1
para j ← p hasta r-1
    si A[j] <= x entonces
        i ← i + 1
        intercambiar(A[i],A[j])
intercambiar(A[i+1],A[r])
devolver(i+1)
```



Quicksort: partición del arreglo

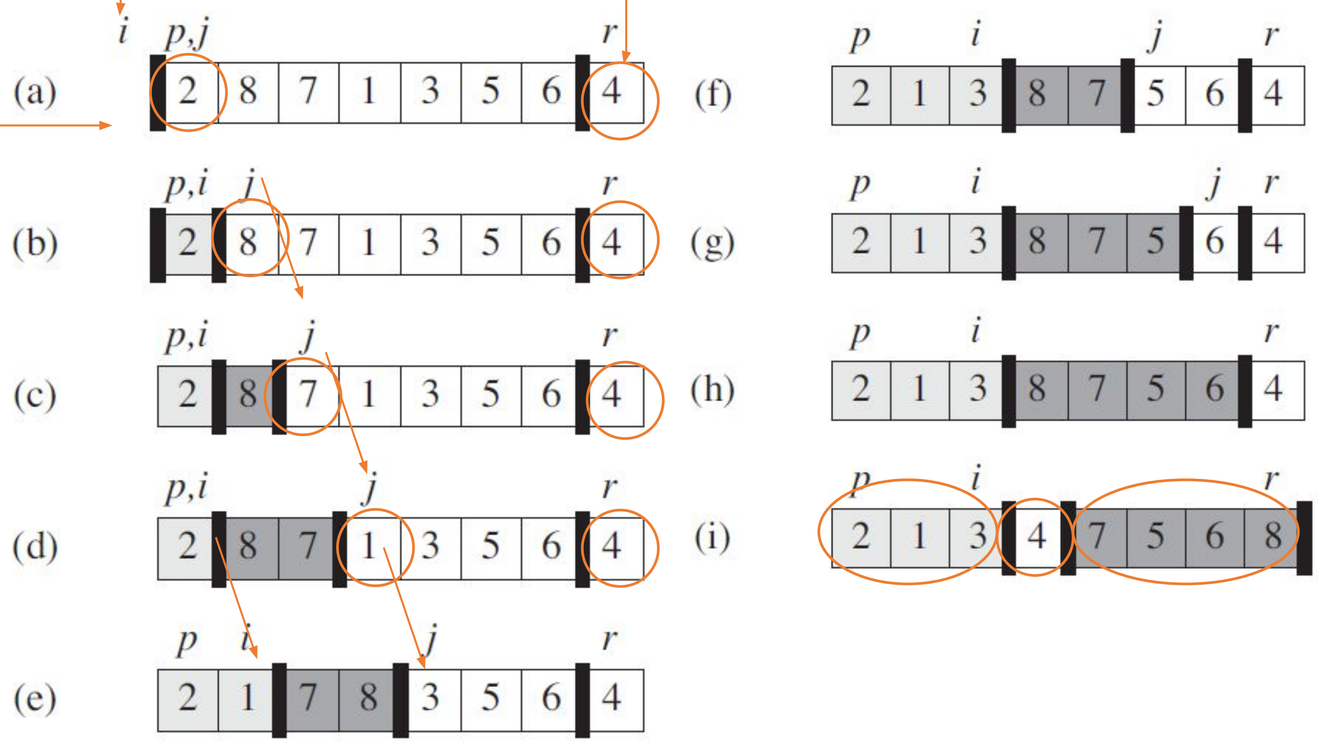
- Alternativa 1: el elemento a revisar es **menor o igual** que el pivote

```
particiona(A,p,r): num
x ← A[r]
i ← p - 1
para j ← p hasta r-1
    si A[j] <= x entonces
        i ← i + 1
        intercambiar(A[i],A[j])
intercambiar(A[i+1],A[r])
devolver(i+1)
```



Quicksort: partición del arreglo

```
particiona(A,p,r): num
x ← A[r]
i ← p - 1
para j ← p hasta r-1
    si A[j] <= x entonces
        i ← i + 1
        intercambiar(A[i],A[j])
intercambiar(A[i+1],A[r])
devolver(i+1)
```



Cormen et al, *Introduction to Algorithms*

Ordenamiento rápido (Quicksort)

```
quickSort(arregloA,inicio,fin)
    Si inicio<fin entonces
        pivote←particiona(arregloA,inicio,fin)
        quickSort(arregloA,inicio,pivote-1)
        quickSort(arregloA,pivote+1,fin)
```

Ordenamiento rápido (Quicksort)

```
quickSort(arregloA,inicio,fin)
    Si inicio<fin entonces
        pivote←particiona(arregloA,inicio,fin)
        quickSort(arregloA,inicio,pivote-1)
        quickSort(arregloA,pivote+1,fin)
```

$$T(n) = 2T(n/2) + O(n)$$

Ordenamiento rápido (Quicksort)

```
quickSort(arregloA,inicio,fin)
    Si inicio<fin entonces
        pivote←particiona(arregloA,inicio,fin)
        quickSort(arregloA,inicio,pivote-1)
        quickSort(arregloA,pivote+1,fin)
```

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^k), si\ n \geq b$$

$$O() = \begin{cases} n^k & si\ a < otro\ caso \\ n^k \log_b n & si\ a = b^k \\ n^{\log_b a} & si\ a > b^k \end{cases}$$

$$T(n) = 2T(n/2) + O(n)$$

Ordenamiento rápido (Quicksort)

```
quickSort(arregloA,inicio,fin)
    Si inicio<fin entonces
        pivote←particiona(arregloA,inicio,fin)
        quickSort(arregloA,inicio,pivote-1)
        quickSort(arregloA,pivote+1,fin)
```

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^k), si\ n \geq b$$

$$O() = \begin{cases} n^k & si\ a < otro\ caso \\ n^k \log_b n & si\ a = b^k \\ n^{\log_b a} & si\ a > b^k \end{cases}$$

$$T(n) = 2T(n/2) + O(n)$$

$\Rightarrow$ O(nlog_2n) ...en caso ***promedio***

Resumen

- Búsqueda:
  - secuencial (desordenados) es O(n)
  - binaria (ordenados) es O(*log*n)
- Ordenamiento:
  - algoritmos eficientes y fáciles de implementar para volúmenes de datos pequeños
  - quicksort es el más eficiente en caso promedio **O(n*log*n)**, pero en el peor caso **O(n^2)**
  - *heapsort* y *mergesort* garantiza **O(n*log*n)** pero trabajan sobre estructuras de datos diferentes (heap y unionFind, respectivamente)

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Algoritmos de ordenamiento

Algoritmo	Peor Caso	Caso Promedio
Inserción	$O(n^2)$	$O(n^2)$
Selección	$O(n^2)$	$O(n^2)$
Burbuja	$O(n^2)$	$O(n^2)$
Quicksort	$O(n^2)$	$O(n\log_2 n)$
HeapSort	$O(n\log_2 n)$	$O(n\log_2 n)$
MergeSort	$O(n\log_2 n)$	$O(n\log_2 n)$

Cuadrático

Logarítmico

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Actividad de cierre

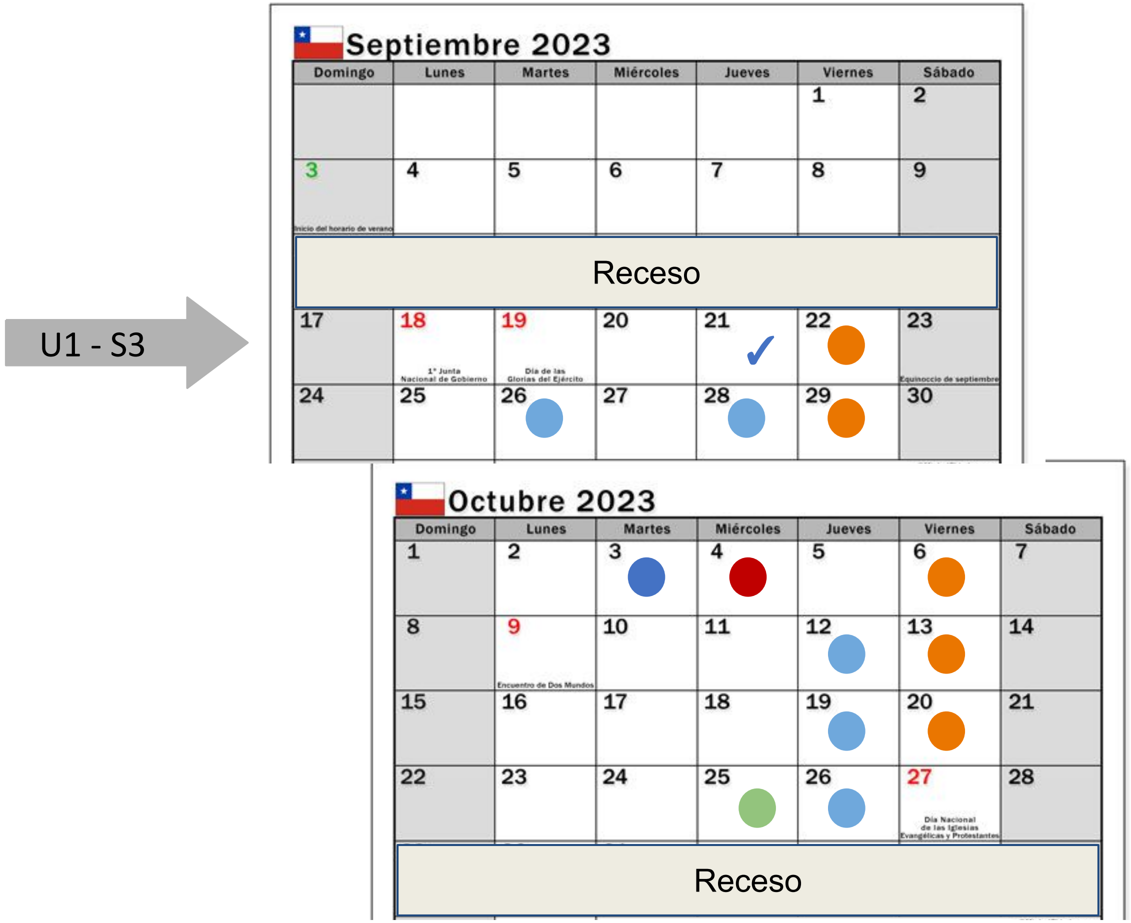
- Ir a menti.com e ingresar código 3952 3535

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Próximas fechas...

- Resumen de la semana:
  - Algoritmos de búsqueda
  - Algoritmos de ordenamiento

~~cátedra~~ – refuerzo – laboratorio



- Próxima semana:
  - Punteros y tipo de dato abstracto