ANÁLISIS Y DISEÑO DEALGORITMOS I

Periodo I – 2023

Jesús Aranda
jesus.aranda@correounivalle.edu.co

Universidad del Valle
Escuela de Ingeniería de Sistemas y Computación

Este documento es una adaptación del material original del profesor Oscar Bedoya

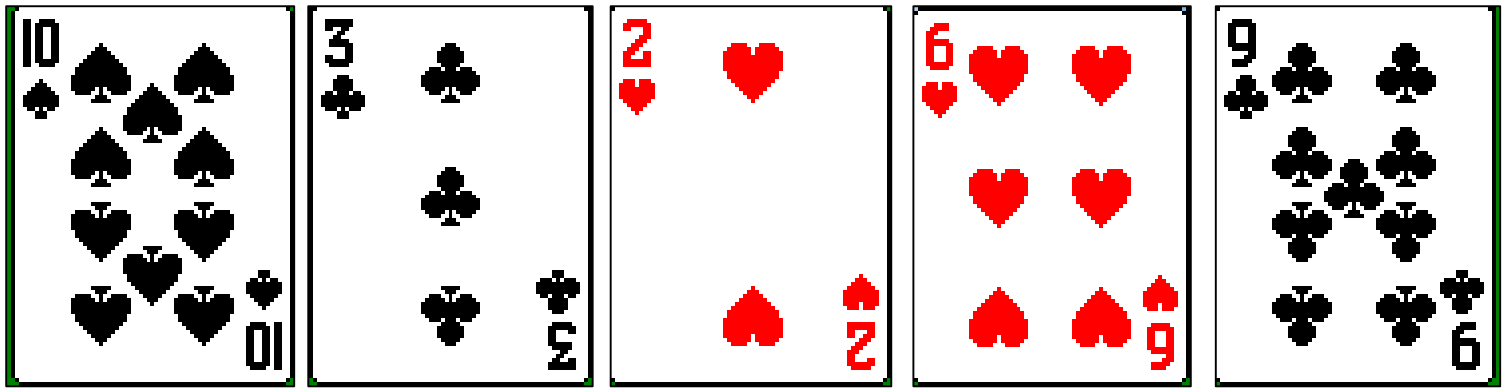
Universidad del Valle

Análisis de un algoritmo de ordenamiento

Insertion sort

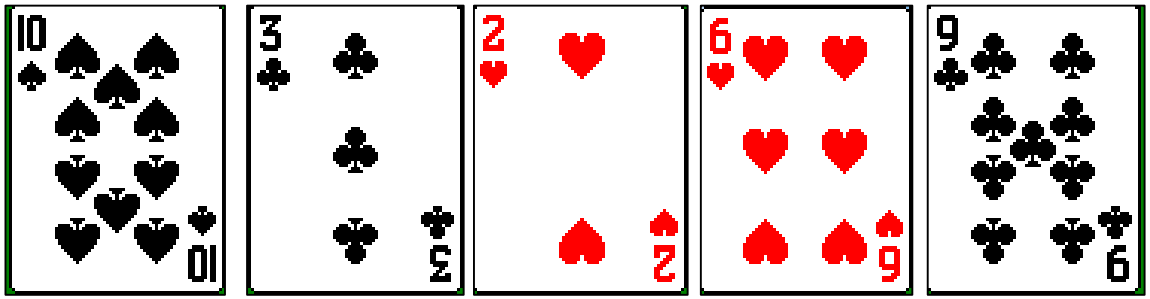
Algoritmos en la computación

Insertion sort
(Idea)



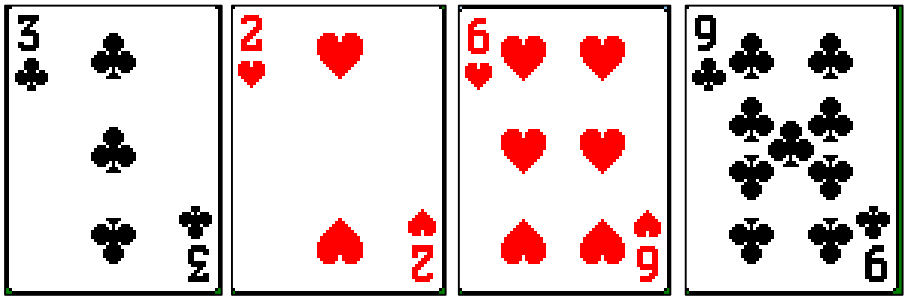
Algoritmos en la computación

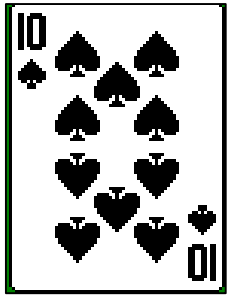
Insertion sort
(Idea)



Algoritmos en la computación

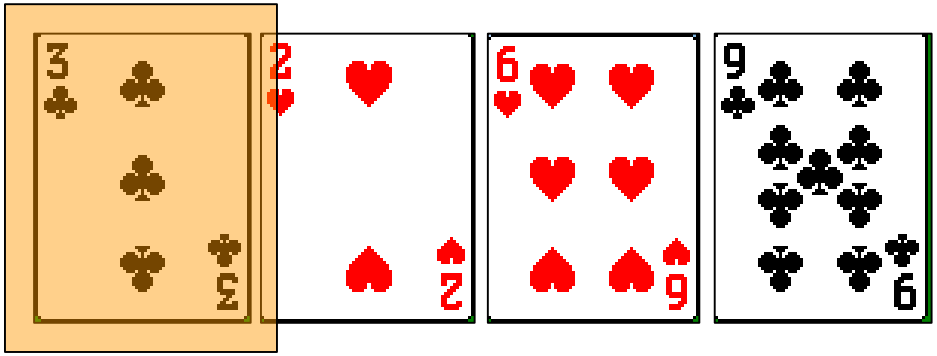
Insertion sort

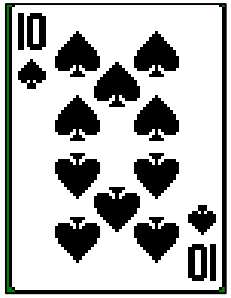




Algoritmos en la computación

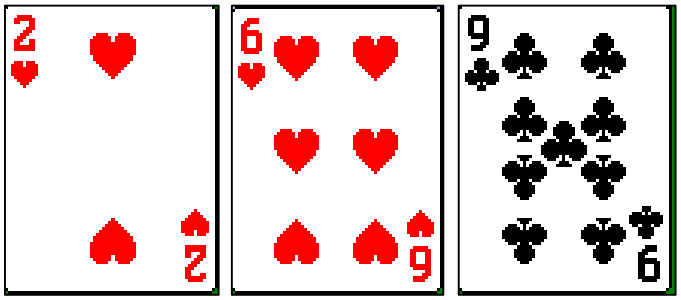
Insertion sort

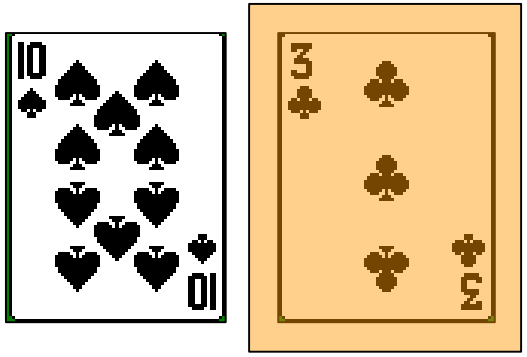




Algoritmos en la computación

Insertion sort

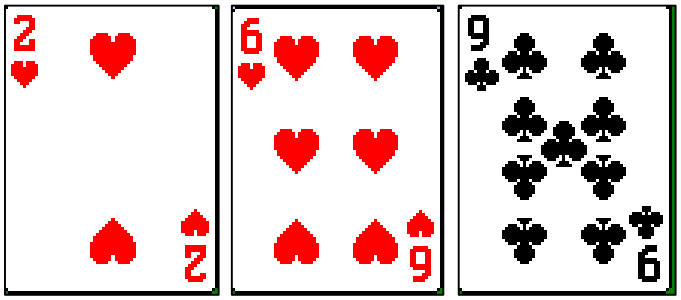


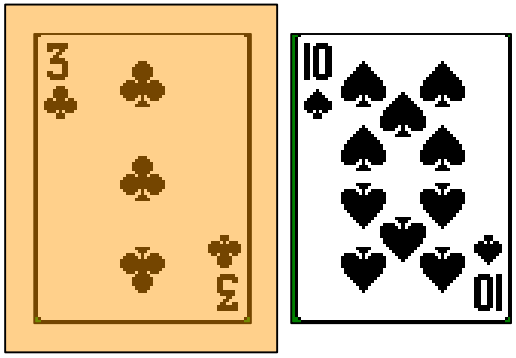


Se recorre de derecha a izquierda buscando el lugar que debe ocupar

Algoritmos en la computación

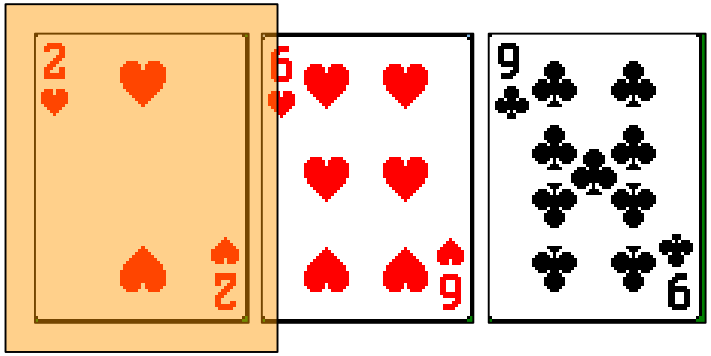
Insertion sort

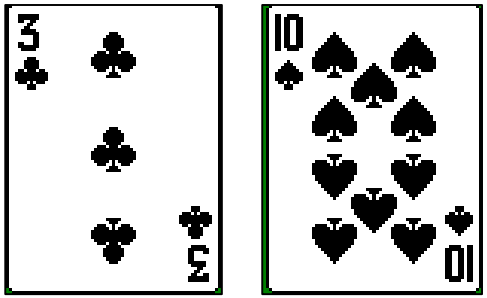




Algoritmos en la computación

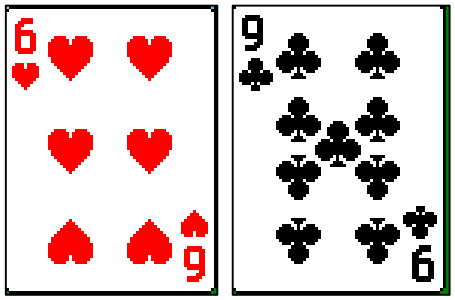
Insertion sort

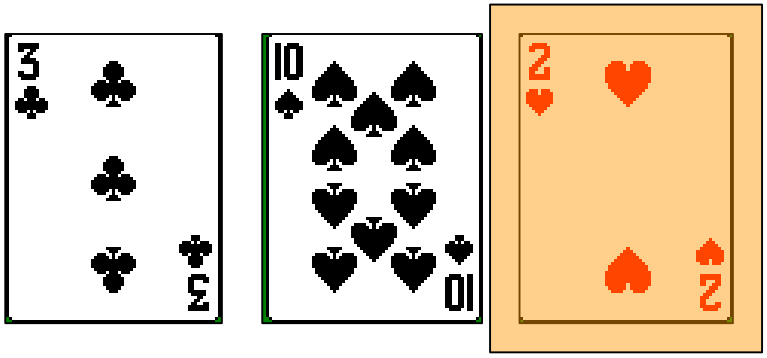




Algoritmos en la computación

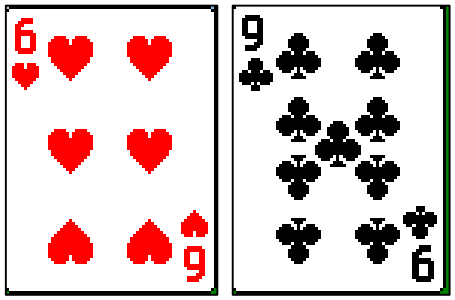
Insertion sort

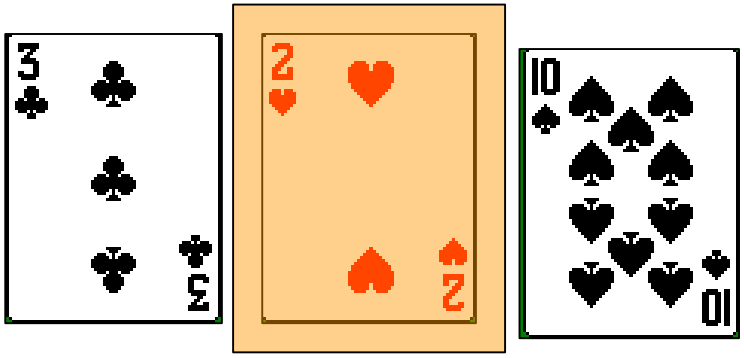




Algoritmos en la computación

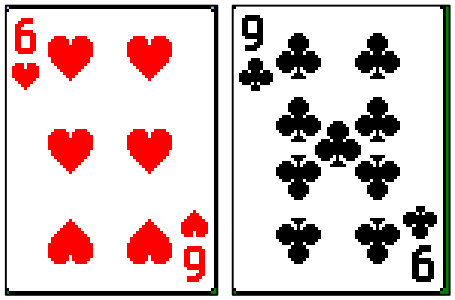
Insertion sort

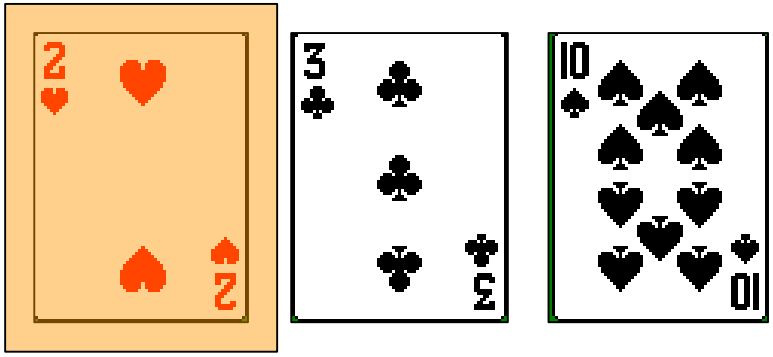




Algoritmos en la computación

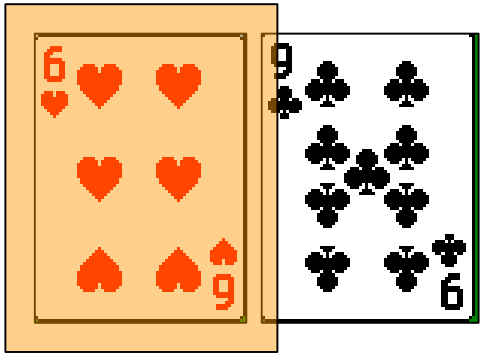
Insertion sort

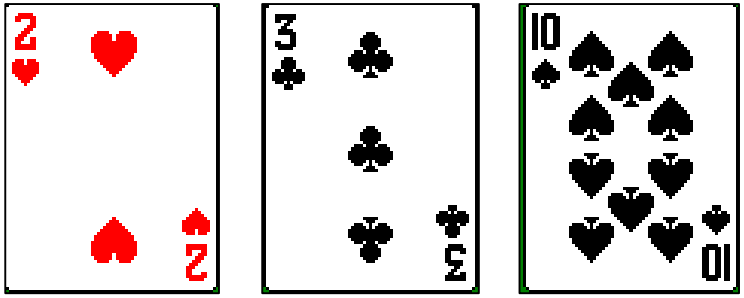




Algoritmos en la computación

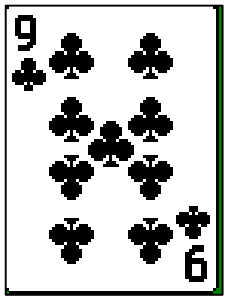
Insertion sort

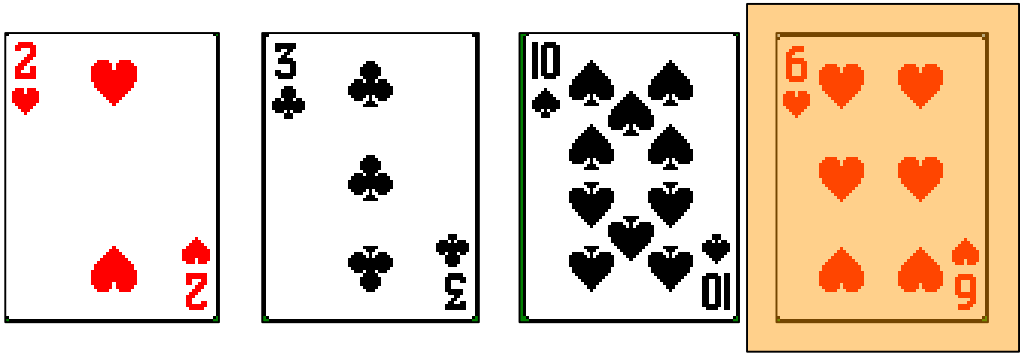




Algoritmos en la computación

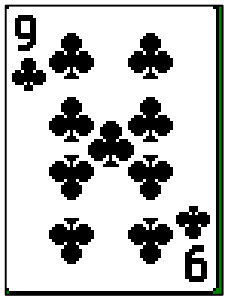
Insertion sort

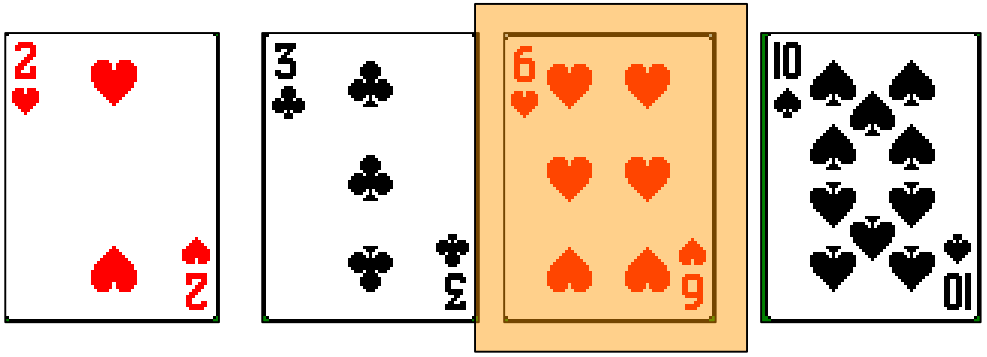




Algoritmos en la computación

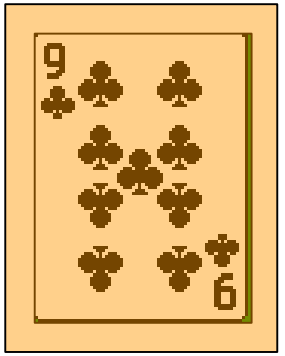
Insertion sort

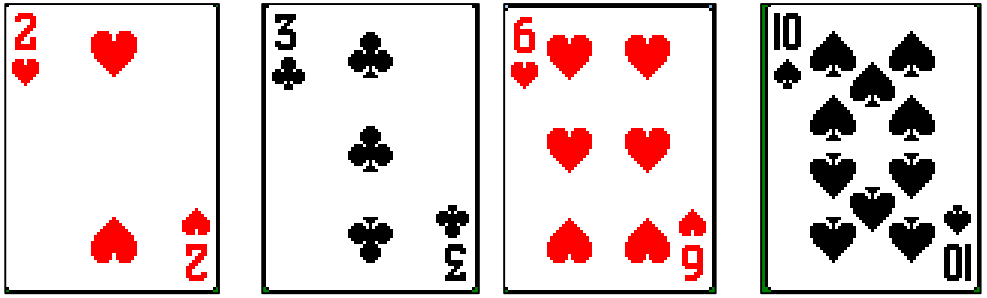




Algoritmos en la computación

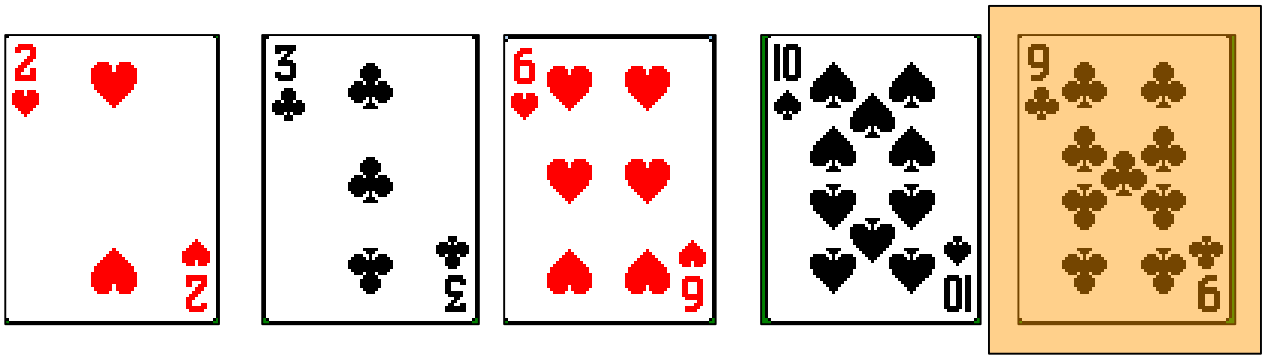
Insertion sort





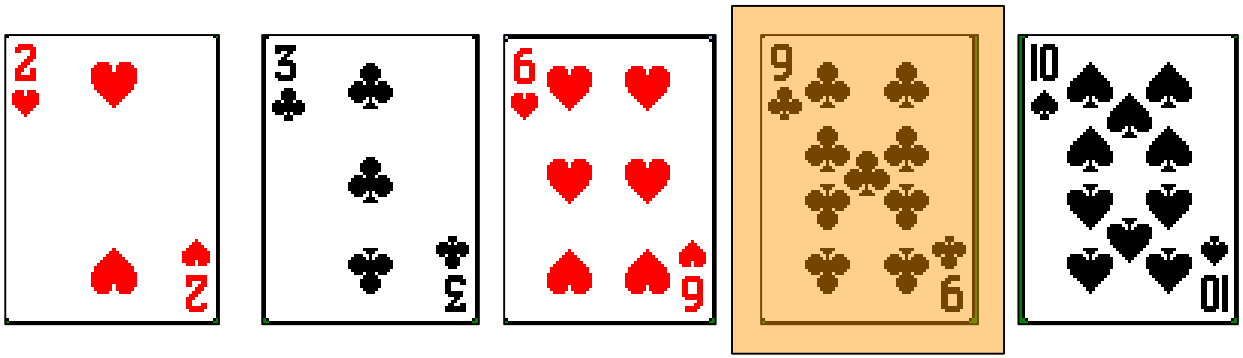
Algoritmos en la computación

Insertion sort



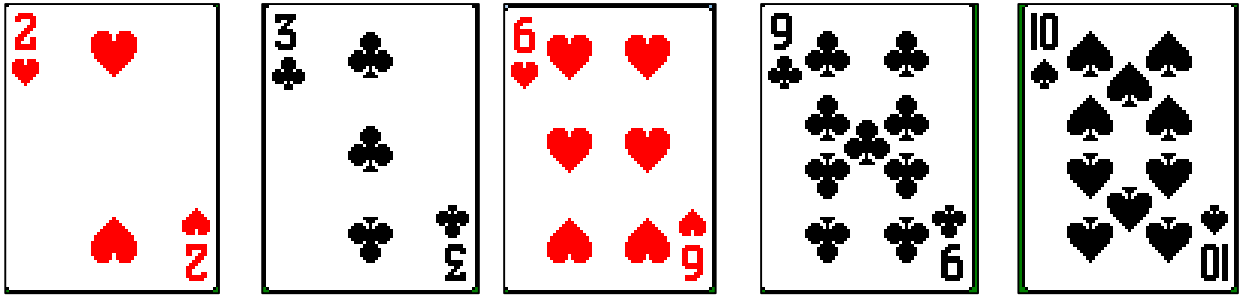
Algoritmos en la computación

Insertion sort



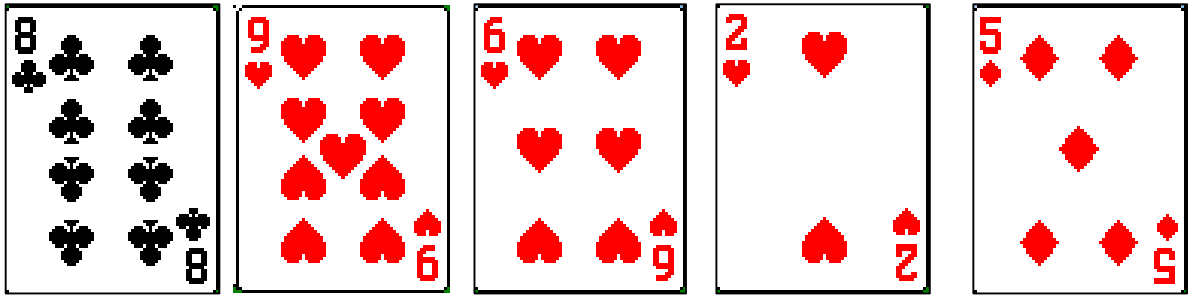
Algoritmos en la computación

Insertion sort



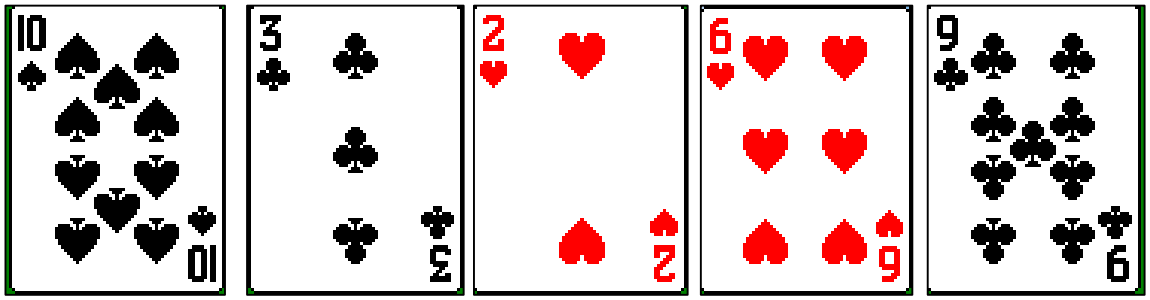
Algoritmos en la computación

Insertion sort



Algoritmos en la computación

Insertion sort



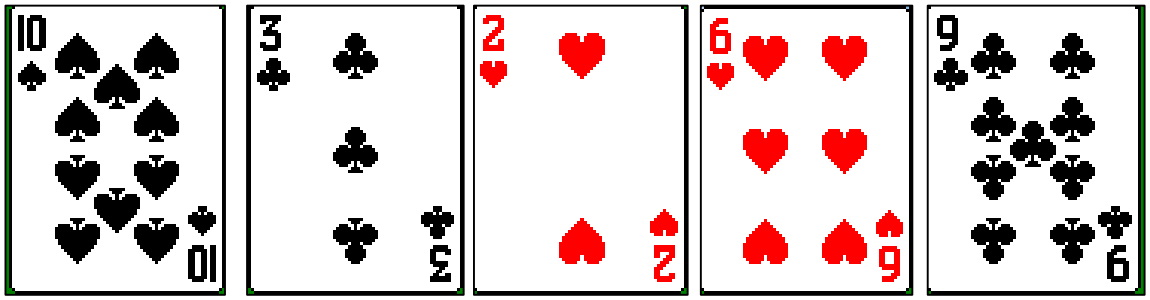
A:	10	3	2	6	9

Algoritmos en la computación

Insertion sort

Desarrolle el algoritmo

INSERTION-SORT(A)



A:	10	3	2	6	9

Algoritmos en la computación

Seudocódigo

```
INSERTION-SORT(A)
1  for j←2 to length[A]
2        do key←A[j]
3           i←j-1
4           while i >0 and A[i] > key
5                 do A[i+1]←A[i]
6                    i←i-1
7           A[i+1]←key
```

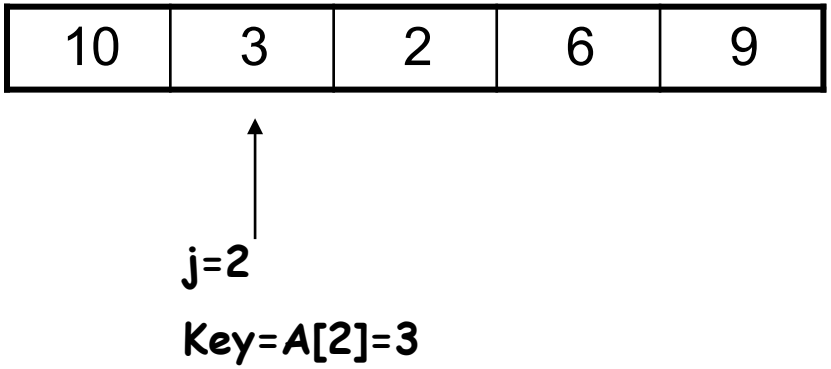
- Asumiremos al usar seudocódigo que los arreglo se indexan desde 1. Ejemplo: si tenemos el arreglo A= [4, 6,7], entonces A[1] vale 4, A[2] vale 5 y A[3] vale 7. El tamaño de A es 3.

Algoritmos en la computación

10	3	2	6	9

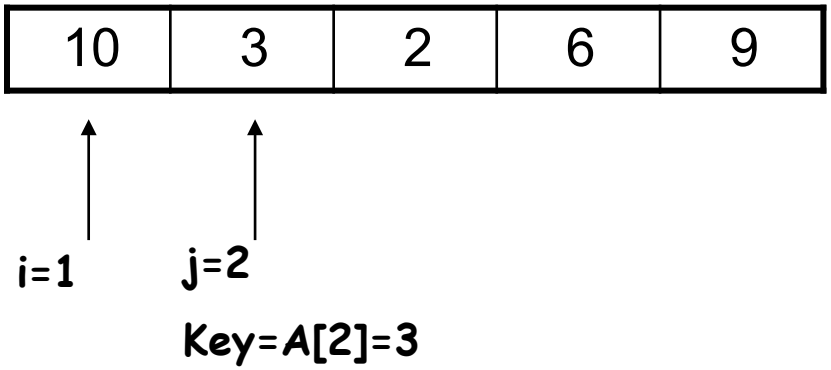
```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



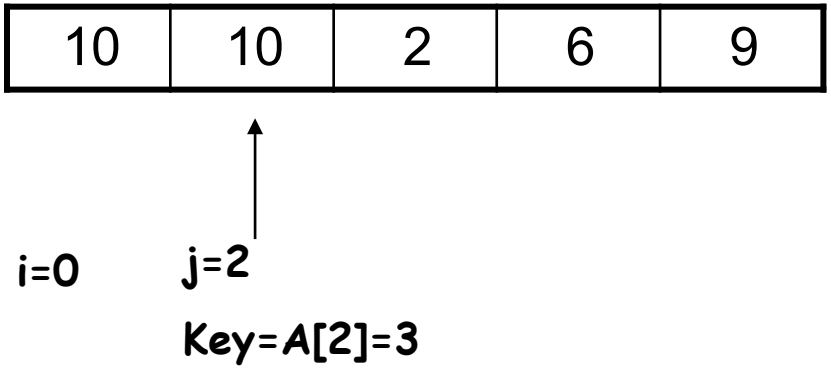

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

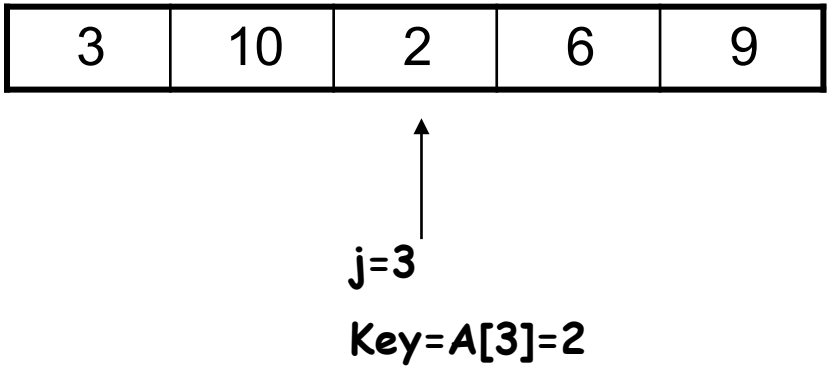
Algoritmos en la computación

3	10	2	6	9

A[1]=3

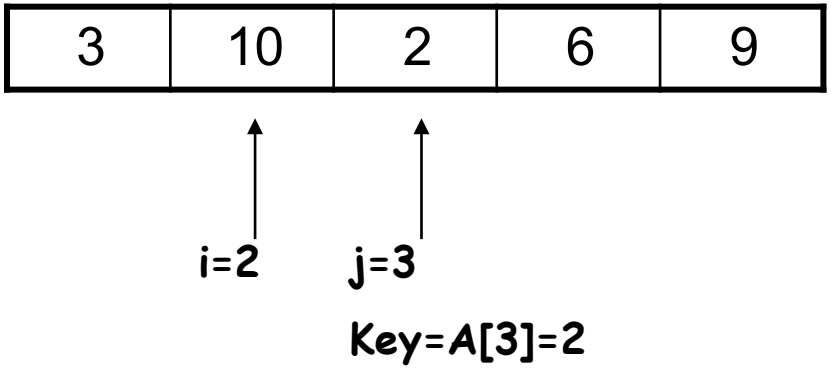
```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



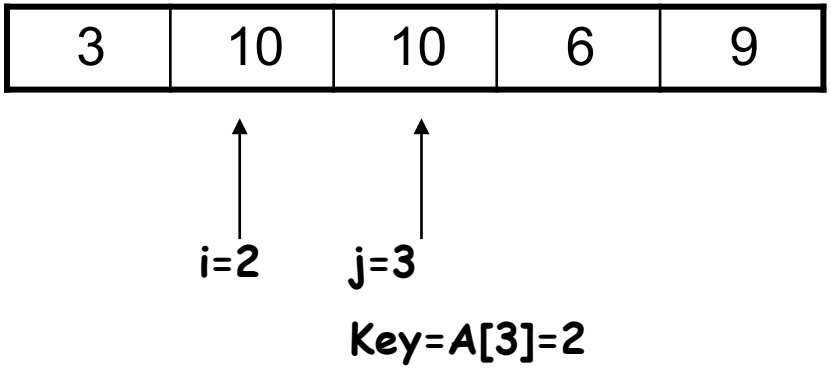

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



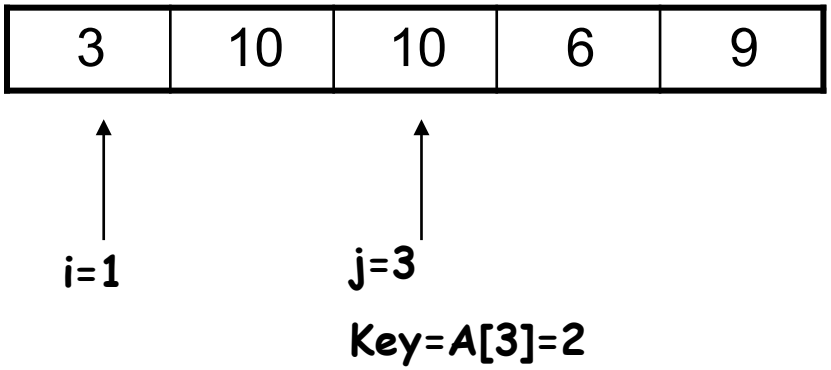

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



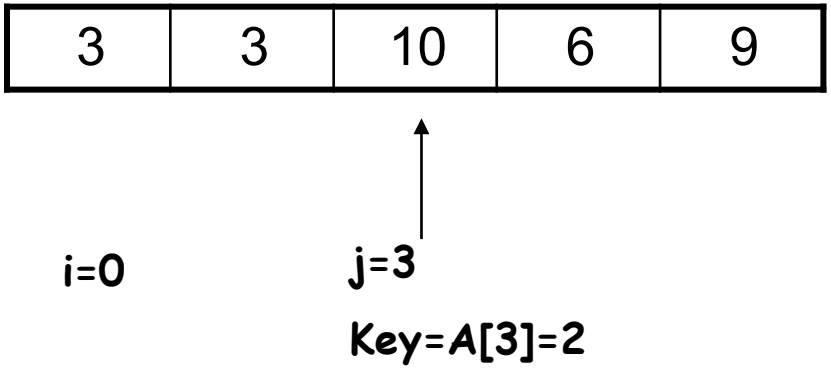

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



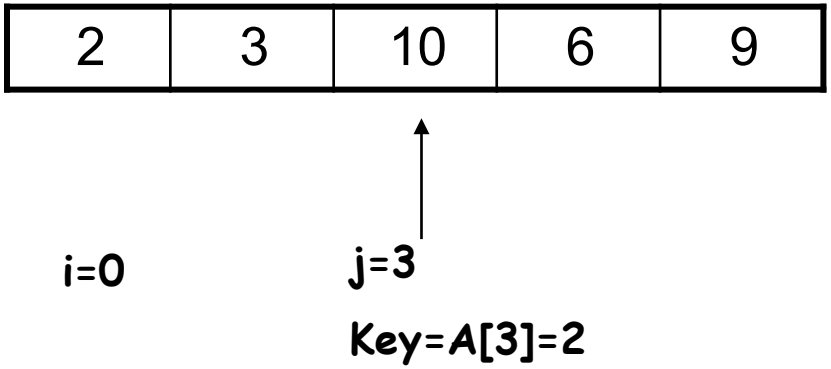

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



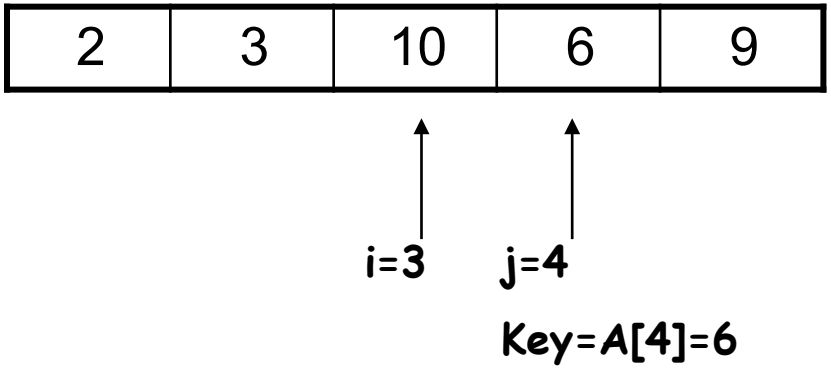

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



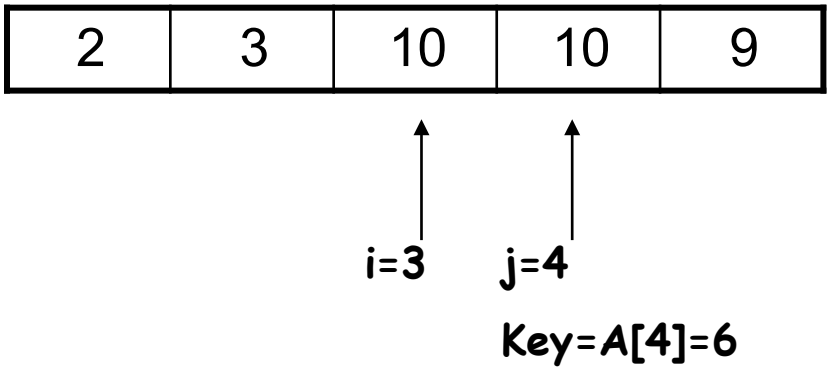

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



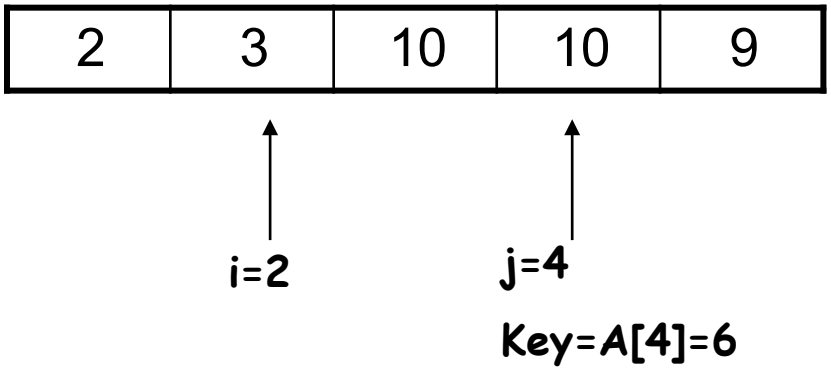

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



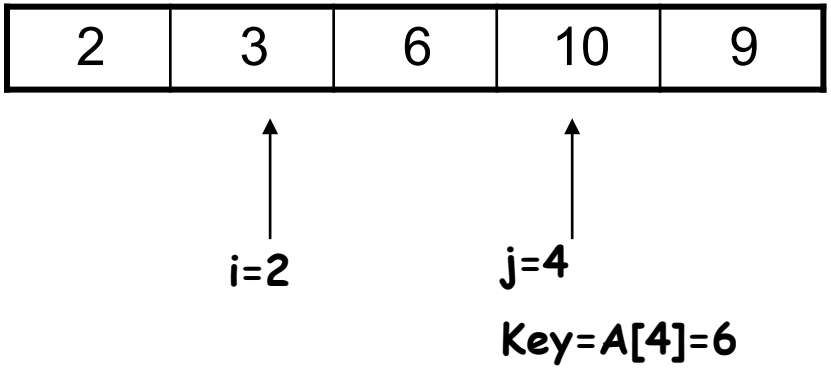

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



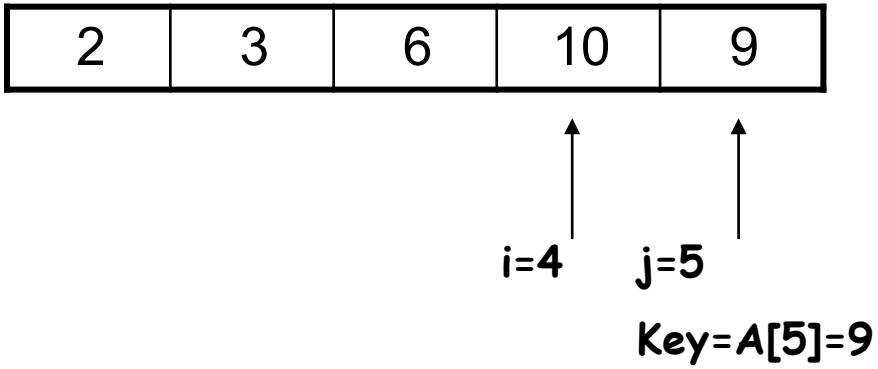

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



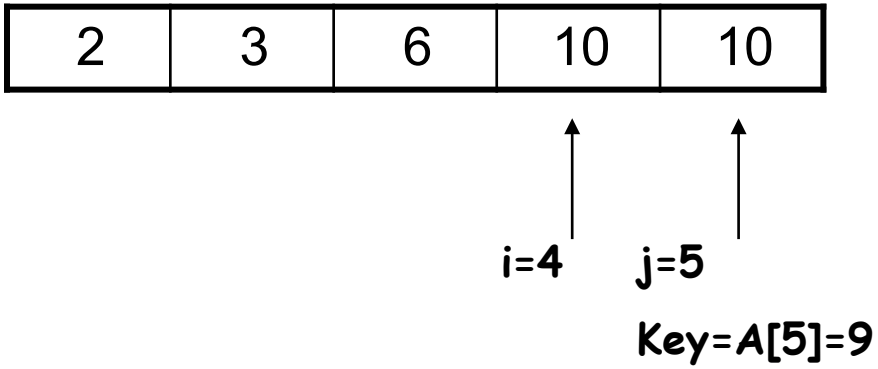

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



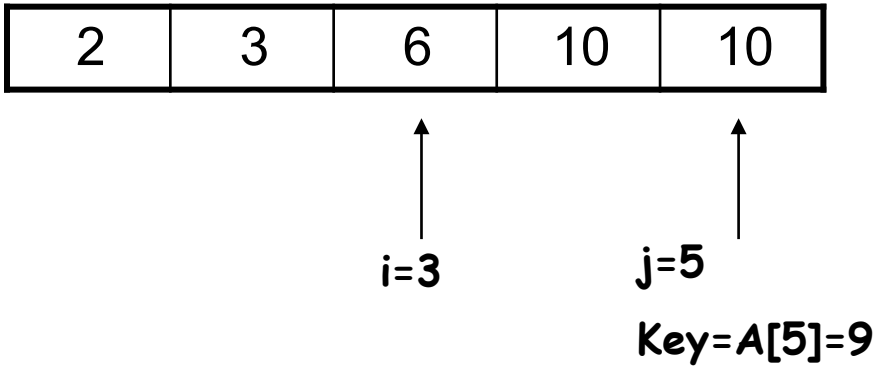

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación



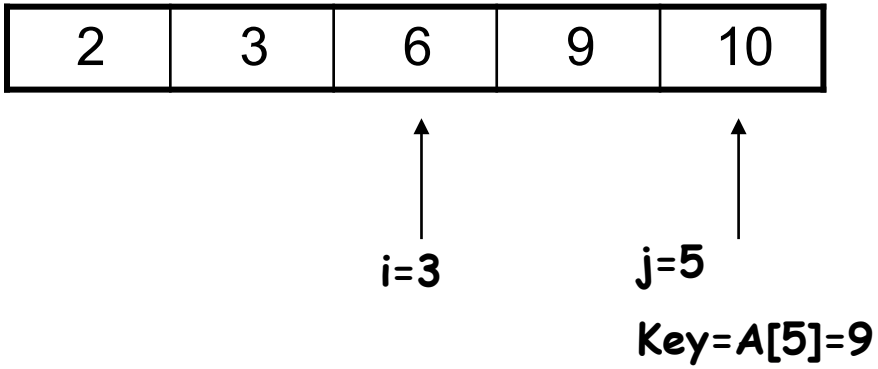

```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←  A[j]
3          i←j-1
4          while i >0 and A[i] > key
5                do A[i+1]←A[i]
6                   i←i-1
7          A[i+1]←key
```

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2       do key ←   A[j]
3              i←j-1
4              while i >0 and A[i] > key
5                     do A[i+1]←A[i]
6                            i←i-1
7              A[i+1]←key
```

Algoritmos en la computación

¿Qué es hacer análisis de algoritmos?

- Calcular tiempo de computación
- Espacio (memoria)
- Analizar las estructuras de datos utilizadas
- Identificar el tipo y número de datos de entrada al algoritmo

+ medidas de análisis (tiempo de ejecución)

medidas experimentales

Algoritmos en la computación

¿Qué es hacer análisis de algoritmos?

- Calcular tiempo de computación*
- Espacio (memoria)
- Analizar las estructuras de datos utilizadas
- Identificar el tipo y número de datos de entrada al algoritmo

+ medidas de análisis (tiempo de ejecución)

medidas experimentales

Algoritmos en la computación

El tiempo de computo depende del tamaño de la entrada, los tiempos serán diferentes si se ordenan 10 números que si se ordenan 10000.

Además, es posible que para dos entradas de igual tamaño, el tiempo sea diferente. Esto depende, de qué tan ordenado ya se encontraba la secuencia de entrada

Algoritmos en la computación

El tiempo de computo T de un algoritmo depende del tamaño de la entrada,

T(n)=f(n), donde n es el tamaño de la entrada

Algoritmos en la computación

El tiempo de computo T de un algoritmo depende del tamaño de la entrada,

$T(n)=f(n)$, donde n es el tamaño de la entrada

$T_1(n)=3n^2$

$T_2(n)=6n^3$

por ejemplo, para n=100, se tiene:

$T_1(n)=3*100^2=30.000$

$T_2(n)=6*100^3=6.000.000$

Algoritmos en la computación

El tiempo de computo T de un algoritmo depende del tamaño de la entrada,

$T(n)=f(n)$, donde n es el tamaño de la entrada

$T_1(n)=3n^2$

$T_2(n)=6n^3$

por ejemplo, para n=100, se tiene:

$T_1(n)=3*100^2=30.000$

$T_2(n)=6*100^3=6.000.000$

Operaciones primitivas

Pasos

Instrucciones

Algoritmos en la computación

Note que los pasos ejecutados se calculan independientemente de la máquina y de la implementación

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]		
2 do key←A[j]		
3 i←j-1		
4 while i >0 and A[i] > key		
5 do A[i+1]←A[i]		
6 i←i-1		
7 A[i+1]←key		

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	
2 do key←A[j]	c_2	
3 i←j-1	c_3	
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

$C_1, C_2, C_3, \ldots$. Indican que existe un costo asociado por la ejecución de la operación de cada línea, este costo puede ser distinto entre las operaciones y puede depender del hardware y software; razón por la cual queda indicado de forma general. Lo que nos va a interesar es calcular las veces que cada operación de cada línea se repite.

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←1 to length[A]	c_1	
2 print A[j]	c_2	

Algoritmos en la computación

for j←1 to 3 ⟶ for (int j=1; j<=3; j++)

j=1 √

j=2 √

j=3 √

j=4 ×

Algoritmos en la computación

for j←1 to 3 ⟶ for (int j=1; j<=3; j++)

j=1 √

j=2 √

j=3 √

j=4 ×

La cantidad de comparaciones en un for es:

cantidad de números válidos + 1

Algoritmos en la computación

for j←1 to 3 ⟶ for (int j=1; j<=3; j++)

j=1 √

j=2 √

j=3 √

j=4 ×

Cantidad de comparaciones:

3 + 1

La cantidad de comparaciones en un for es:

cantidad de números válidos + 1

Algoritmos en la computación

for j←1 to n

¿Cuántas veces se repite?

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←1 to length[A]	c_1	n+1
2 print A[j]	c_2	n

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	?
2 print A[j]	c_2	?

Algoritmos en la computación

for j←2 to 4 ⟶ for (int j=2; j<=4; j++)

j=2 √

j=3 √

j=4 √

j=5 ×

La cantidad de comparaciones en un for es:

cantidad de números válidos + 1

Algoritmos en la computación

for j←2 to 4 ⟶ for (int j=2; j<=4; j++)

j=2 √

j=3 √

j=4 √

j=5 ×

La cantidad de comparaciones en un for es:

(4-2+1) + 1

Algoritmos en la computación

for j←2 to 6 ⟶ for (int j=2; j<=6; j++)

???

Algoritmos en la computación

for j←2 to n

La cantidad de comparaciones en el for es:

Algoritmos en la computación

for j←2 to n

La cantidad de comparaciones en el for es:

(n-2+1) + 1 = n

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	???
2 do key←A[j]	c_2	
3 i←j-1	c_3	
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	
3 i←j-1	c_3	
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	???
3 i←j-1	c_3	
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Depende de qué tan ordenados se encuentran los datos en A

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Para cada j, se puede repetir una cantidad diferente de veces

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Sea t_j, la cantidad de comparaciones que se hacen en el while para cada valor de j

Por ejemplo, t_2, es un número que indica cuántas veces se cumple la condición cuando j=2

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$t_2 + t_3 + t_4 + \ldots + t_n$
5 do A[i+1]←A[i]	c_5	
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Sea t_j, la cantidad de comparaciones que se hacen en el while para cada valor de j

Por ejemplo, t_2, es un número que indica cuántas veces se cumple la condición cuando j=2

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$t_2 + t_3 + t_4 + \ldots + t_n$
5 do A[i+1]←A[i]	c_5	$(t_2-1)+(t_3-1)+(t_4-1)+ \ldots + (t_n-1)$
6 i←i-1	c_6	
7 A[i+1]←key	c_7	

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$t_2 + t_3 + t_4 + \ldots + t_n$
5 do A[i+1]←A[i]	c_5	$(t_2-1)+(t_3-1)+(t_4-1)+ \ldots + (t_n-1)$
6 i←i-1	c_6	$(t_2-1)+(t_3-1)+(t_4-1)+ \ldots + (t_n-1)$
7 A[i+1]←key	c_7	

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$t_2 + t_3 + t_4 + \ldots + t_n$
5 do A[i+1]←A[i]	c_5	$(t_2-1)+(t_3-1)+(t_4-1)+ \ldots + (t_n-1)$
6 i←i-1	c_6	$(t_2-1)+(t_3-1)+(t_4-1)+ \ldots + (t_n-1)$
7 A[i+1]←key	c_7	n-1

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$\sum_{j=2}^{n} t_j$
5 do A[i+1]←A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6 i←i-1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 A[i+1]←key	c_7	n-1

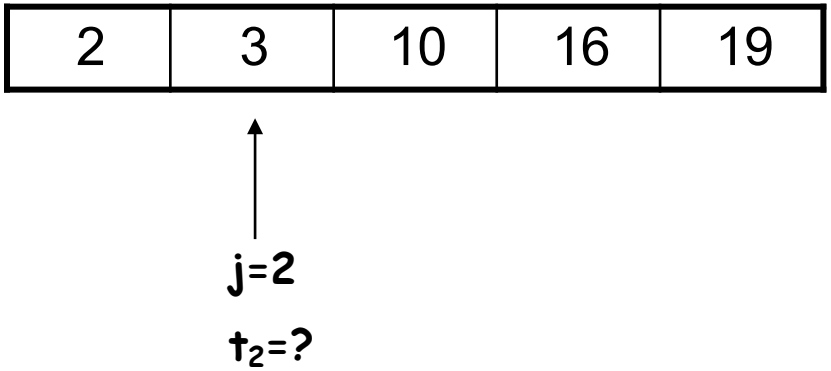
T(n)=???

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$\sum_{j=2}^{n} t_j$
5 do A[i+1]←A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6 i←i-1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 A[i+1]←key	c_7	n-1

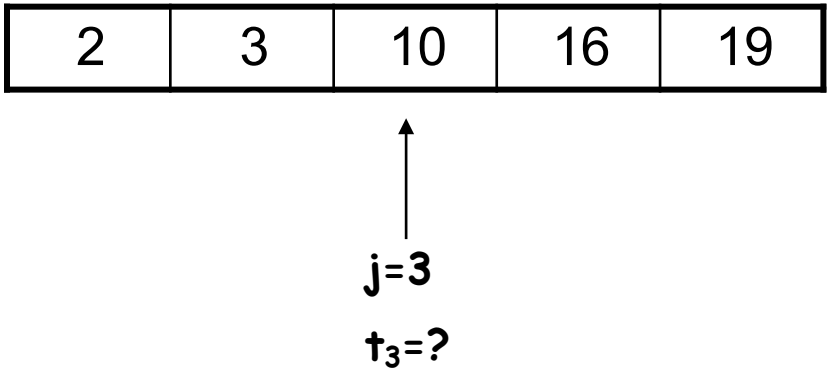
En el mejor de los casos, cuánto vale t_j?

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2do key←A[j]  3
              i←j-1
4                 while i >0 and A[i] > key
5                       do A[i+1]←A[i]
6                          i←i-1
7                 A[i+1]←key
```

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2do key←A[j]  3
            i←j-1
4               while i >0 and A[i] > key
5                       do A[i+1]←A[i]
6                          i←i-1
7               A[i+1]←key
```

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$\sum_{j=2}^{n} t_j$
5 do A[i+1]←A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6 i←i-1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 A[i+1]←key	c_7	n-1

En el mejor de los casos, t_j=1. Esto ocurre cuando el arreglo de entrada está ordenado ascendentemente. En cuyo caso la línea 4 se ejecutaría solo 1 vez por cada iteración en j.

T(n)=???

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$\sum_{j=2}^{n} 1$
5 do A[i+1]←A[i]	c_5	0
6 i←i-1	c_6	0
7 A[i+1]←key	c_7	n-1

En el mejor de los casos, t_j=1. Esto ocurre cuando el arreglo de entrada está ordenado ascendentemente. En cuyo caso la línea 4 se ejecutaría solo 1 vez por cada iteración en j. Podríamos identificar 3 formas de calcular T(n):

T(n) = n* c1 + (n-1)*c2 + … + (n-1)*c7 (forma exhaustiva, considerando máquina)

T(n) = n + (n-1) + (n -1) + …. + (n-1) (considerando todas las veces de todas las líneas)

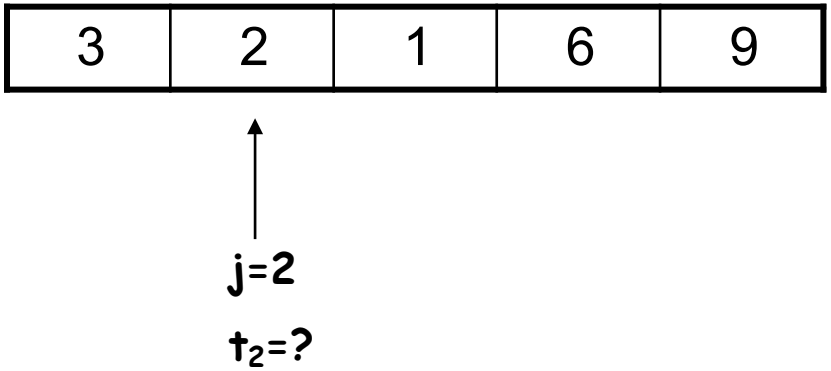
T(n) = $\sum_{j=2}^{n} 1$ (considerando la operación básica, es la que usaremos)

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$\sum_{j=2}^{n} t_j$
5 do A[i+1]←A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
6 i←i-1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 A[i+1]←key	c_7	n-1

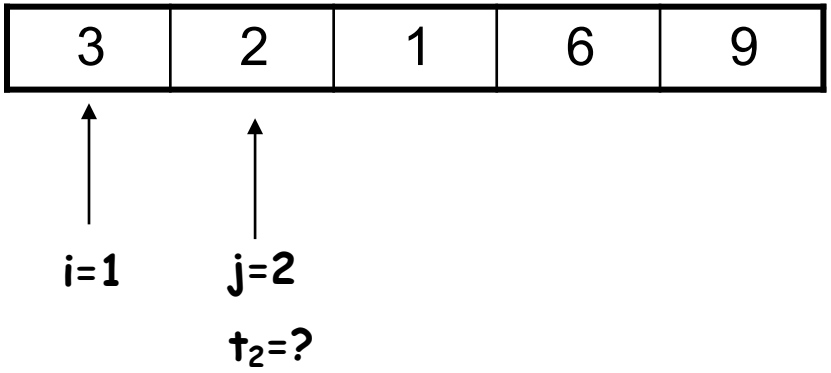
En el peor de los casos, cuánto vale t_j?

Algoritmos en la computación



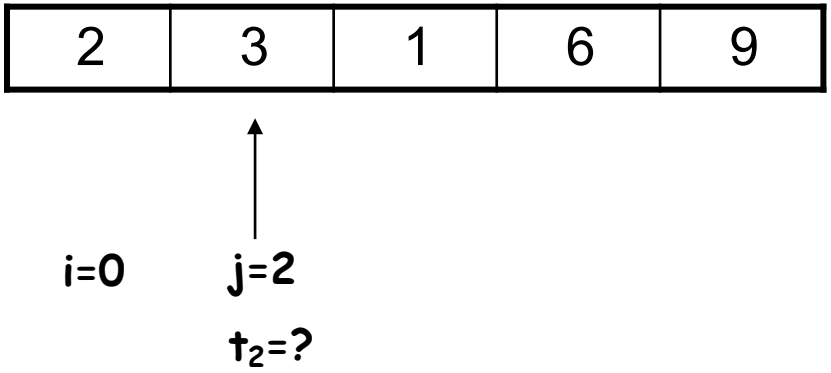

```
INSERTION-SORT(A)
1for j←2 to length[A]
2do key←A[j]  3
            i←j-1
4               while i >0 and A[i] > key
5                       do A[i+1]←A[i]
6                          i←i-1
7               A[i+1]←key
```

Algoritmos en la computación



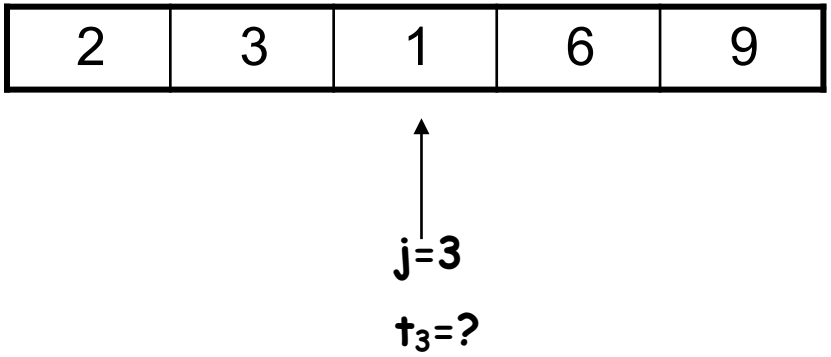

```
INSERTION-SORT(A)
1for j←2 to length[A]
2do key←A[j]  3
            i←j-1
4               while i >0 and A[i] > key
5                       do A[i+1]←A[i]
6                          i←i-1
7               A[i+1]←key
```

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2do key←A[j]  3
              i←j-1
4                 while i >0 and A[i] > key
5                       do A[i+1]←A[i]
6                          i←i-1
7                 A[i+1]←key
```

Algoritmos en la computación




```
INSERTION-SORT(A)
1for j←2 to length[A]
2do key←A[j]  3
          i←j-1
4             while i >0 and A[i] > key
5                   do A[i+1]←A[i]
6                      i←i-1
7             A[i+1]←key
```

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$\sum_{j=2}^{n} j$
5 do A[i+1]←A[i]	c_5	$\sum_{j=2}^{n} (j-1)$
6 i←i-1	c_6	$\sum_{j=2}^{n} (j-1)$
7 A[i+1]←key	c_7	n-1

En el mejor de los casos, t_j=j. Ocurre cuando el arreglo inicialmente está ordenado descendentemente, porque la línea 4 se ejecutará para todos los valores de i desde j-1 hasta 0. Podríamos identificar 3 formas de calcular T(n):

T(n) = n* c1 + (n-1)*c2 + … + (n-1)*c7 (forma exhaustiva, considerando máquina)

T(n) = n + (n-1) + (n -1) + …. + (n-1) (considerando todas las veces de todas las líneas)

T(n) = $\sum_{j=2}^{n} 1$ (considerando la operación básica, es la que usaremos)

Algoritmos en la computación

Instrucción	Costo	Veces que se repite
1 for j←2 to length[A]	c_1	n
2 do key←A[j]	c_2	n-1
3 i←j-1	c_3	n-1
4 while i >0 and A[i] > key	c_4	$\sum_{j=2}^{n} j/2$
5 do A[i+1]←A[i]	c_5	$\sum_{j=2}^{n} (j/2-1)$
6 i←i-1	c_6	$\sum_{j=2}^{n} (j/2-1)$
7 A[i+1]←key	c_7	n-1

En el caso promedio, se supone que se necesitan j/2 comparaciones aprox, esto es, t_j=j/2.

T(n)=???

Algoritmos en la computación

Calcule el tiempo de computo para el algoritmo

```
def programa1(mat1, mat2)        # suponga que len(mat1)=len(mat2)=n
```

Instrucción	Costo
1 i=1	c_1
2 while i<=len(mat1)	c_2
3 j←1	c_3
4 while j<=len(mat2)	c_4
5 mat3[i][j]←mat1[i][j]+mat2[i][j]	c_5
6 j←j+1	c_6
7 i←i+1	c_7

Algoritmos en la computación

Calcule el tiempo de computo para el algoritmo

def programa2(n)

Instrucción	Costo
1 s←0	c_1
2 i←1	c_2
3 while i<=n	c_3
4 t←0	c_4
5 j←1	c_5
6 while j<=i	c_6
7 t←t+1	c_7
8 j←j+1	c_8
9 s←s+t	c_9
10 i←i+1	c_{10}

Algoritmos en la computación

Calcule el tiempo de computo para el algoritmo

def programa3(n)

Instrucción	Costo
1 i←1	c_1
2 while i<=n	c_2
3 k←i	c_3
4 while k<=n	c_4
5 k← k+1	c_5
6 k←1	c_6
7 while k<=i	c_7
8 k←k+1	c_8
9 i←i+1	c_9

Algoritmos en la computación

Diseño de algoritmos

Otras alternativas para el diseño de algoritmos son:

- Aproximación incremental
- Dividir y conquistar
- Programación dinámica
- Técnicas voraces

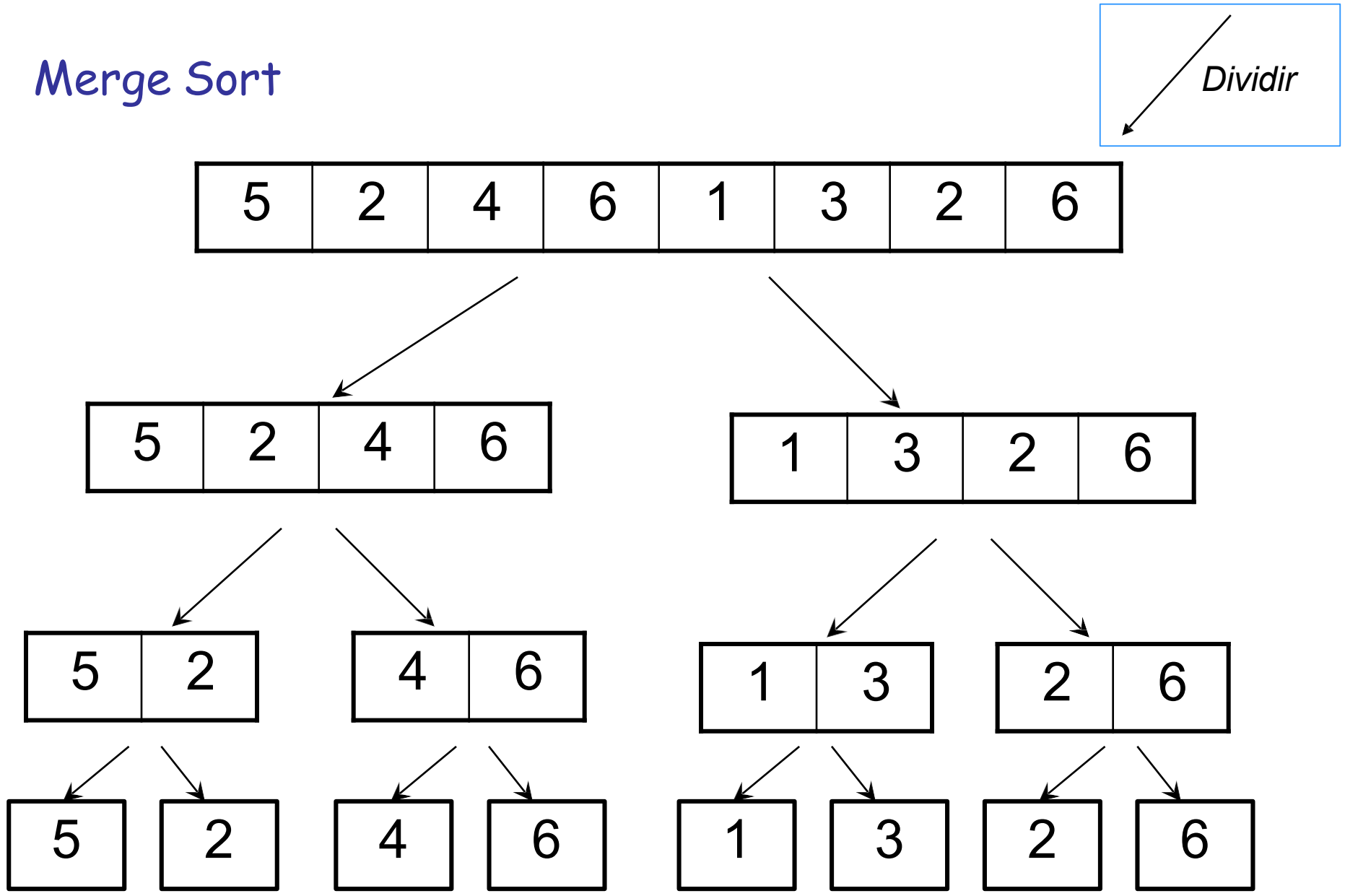
Algoritmos en la computación

Dividir y conquistar

- Considera recursividad
- Dividir el problema en subproblemas
- Conquistar los subproblemas (solucionarlos recursivamente)
- Combinar las soluciones de los subproblemas para crear la solución al problema original

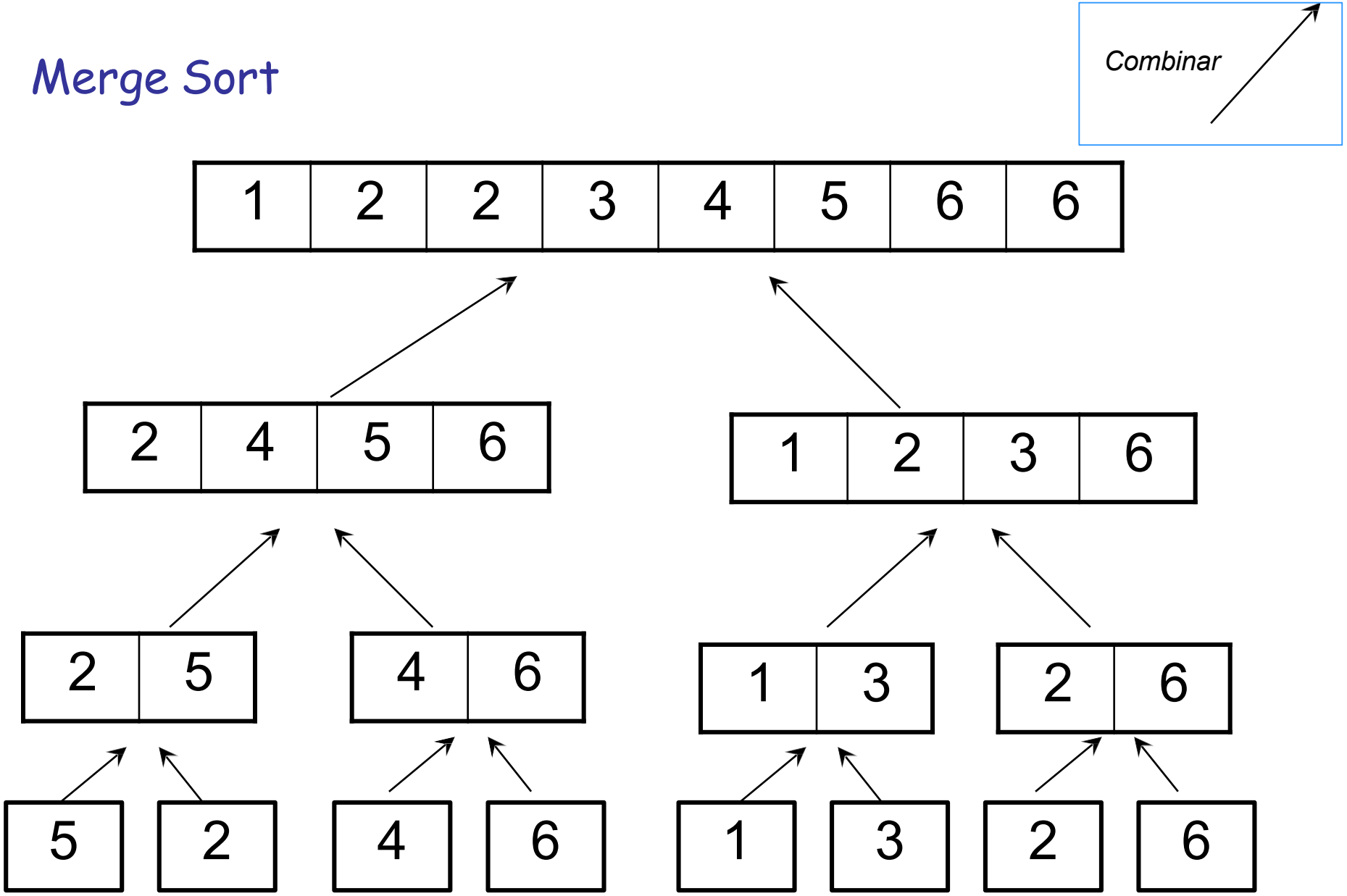
Algoritmos en la computación

Merge Sort



Algoritmos en la computación

Merge Sort



Algoritmos en la computación

Merge sort

Entrada: $\langle a_1, a_2, \ldots, a_n \rangle$

- Dividir: $\langle a_1, \ldots, a_q \rangle$ $\langle a_{q+1}, \ldots, a_n \rangle$
- Conquistar: $a'_1 \leq a'_2 \leq \ldots \leq a'_q$ y $a'_{q+1} \leq a'_{q+2} \leq \ldots \leq a'_n$
- Combinar: si $a'_1 \leq a'_{q+1}$ y $a'_2 > a'_{q+1}$ entonces $\langle a'_1, a'_{q+1}, \ldots \rangle$ sino ...

Algoritmos en la computación

Merge sort

$$T(n)=\begin{cases} 1 & n=1 \\ \text{Dividir}(n) + \text{Conquistar}(n) + \text{Combinar}(n) & \end{cases}$$

Algoritmos en la computación

Merge sort

$$T(n)=\begin{cases} 1 & n=1 \\ 1 + 2T(n/2) + n & n>1 \end{cases}$$

al resolver la recurrencia, $T(n) = n \lg n$

Algoritmos en la computación

Análisis de algoritmos Insertion sort y Merge sort

Computador 1	**Computador 2**
10^8 instrucciones/seg	10^6 instrucciones/seg

Implementación 1(Insertion)	**Implementación 2(Merge)**
$2n^2$	$50n*\lg n$

Algoritmos en la computación

Analisis de algoritmos Insertion sort y Merge sort

Computador 1	Computador 2
10^8 instrucciones/seg	10^6 instrucciones/seg

Implementación 1(Insertion)	Implementación 2(Merge)
$2n^2$	$50n*\lg n$

Ordenar un arreglo de 10^6 números

Tiempo 1(Insertion)	Tiempo 2(Merge)
$2(10^6)^2/10^8$=20.000segs=5.56horas	$50*10^6 \lg 10^6/10^6$=1.000segs=16.67 mins