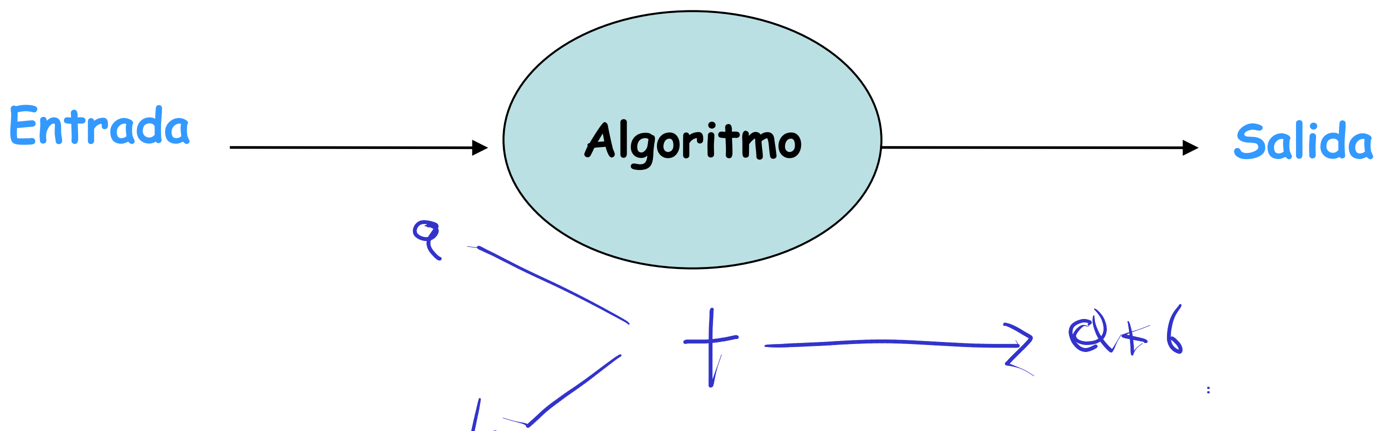
Fundamentos de análisis y diseño de algoritmos

Algoritmos en la computación

Algoritmos en la computación

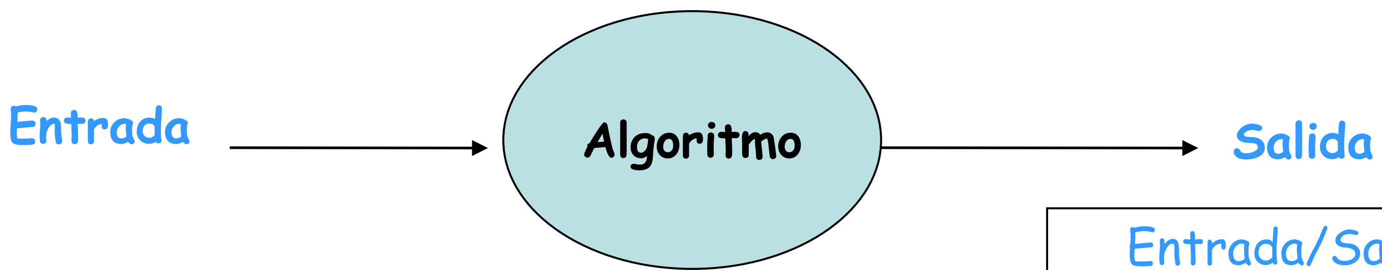
Un algoritmo es un procedimiento computacional que toma un valor o conjunto de valores como entrada y produce un valor o conjunto de valores como salida



"Un algoritmo se puede ver como una herramienta para resolver un problema computacional bien especificado"

Algoritmos en la computación

Un algoritmo es un procedimiento computacional que toma un valor o conjunto de valores como entrada y produce un valor o conjunto de valores como salida



Entrada/Salida especifican los problemas

"Un algoritmo se puede ver como una herramienta para resolver un problema computacional bien especificado"

Algoritmos en la computación

Problema 1: Cambio de monedas

200, 200, 200, 50, 50, 100

100

450

Entrada: $A=\{a_1, a_2, \ldots, a_n\}$ tal que ,$P \in N$

$a_i \in A$, $a_i \in N$

$\{\emptyset, |200|, |200|, |200|, |50|, |50|, |100|$

Salida: $A'=\langle a'_1, a'_2, \ldots, a'_n \rangle$ tal que $\sum_{m \in A'} m = P$ y $|A'|$

es mínimo

$O(2^n)$

Algoritmos en la computación

Problema 1: Cambio de monedas

El problema de cambio de monedas aborda la forma de encontrar el número mínimo de monedas (de ciertas denominaciones) de tal manera que las monedas seleccionadas sumen una cantidad dada.

Tomado de: https://www.istockphoto.com/es/fotos/us-coins

Algoritmos en la computación

Problema 2: ???

Primos

Entrada: $n \in Z^+$

Salida: 1, si $n=1$

1, si $n=2$

1, si $n>2$ y $n \bmod i \neq 0 \ \forall i \in \{2, ..., n-1\}$

0, si $n>2$ y $\exists x \mid n \bmod x = 0 \wedge x \in \{2, ..., n-1\}$

in		out
1		1
2		1
3	$3 \bmod 2 = 1$	1
4	$4 \bmod 2 = 0$ $4 \bmod 3 = 1$	0
5	$5 \bmod 2 = 1$ $5 \bmod 3 = 2$ $5 \bmod 4 = 1$	1

7 | $7 \bmod 2 = 1$, $7 \bmod 3 = 1$, $7 \bmod 4 = 3$, $7 \bmod 5 = 2$, $7 \bmod 6 = 1$

6 | $6 \bmod 2 = 0$, $6 \bmod 3 = 0$, $6 \bmod 4 = 2$, $6 \bmod 5 = 1$ | 0

Algoritmos en la computación

Problema 3: ???

Ordenar

menor a mayor

Entrada: $S=<a_1, a_2, \ldots, a_n>$

Salida: una permutación de S, $S'=<a_1', a_2'. \ldots, a_n'>$ tal que $a_1'<a_2'<\ldots,<a_n'$

$P(n,n) = n!$

Algoritmos en la computación

Instancia de un problema

$\langle 3,12,23,4,9 \rangle$ → **Algoritmo de ordenamiento** → $\langle 3,4,9,12,23 \rangle$

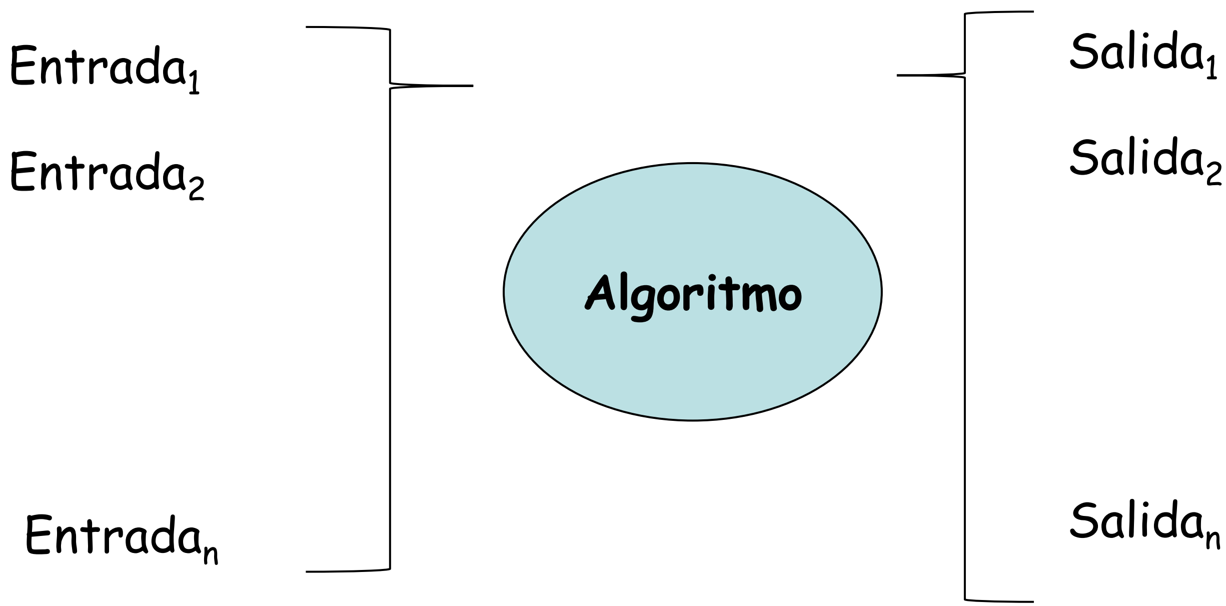
(instancia del problema)

Una instancia es una entrada válida para el algoritmo

Algoritmos en la computación

Correctitud

Se dice que un algoritmo es correcto, si para cada *instancia*, el algoritmo termina con la salida correcta



Algoritmos en la computación

```
primo(int n){
    if (n == 1)
        return 1;
    if (n % 2 == 0)
        return 0;
    else
        return 1;
    }
```

n	Sol
1	1
2	0 X

Para cada instancia, el algoritmo termina con la salida correcta

¿Es el algoritmo primo correcto?

R/ No

Algoritmos en la computación

```
primo(int n){
 if (n == 1)
       return 1;
    else {
       int c = 0;
       for (int i = 2; i < n; i++) {
          if (n % i == 0)
          c++;
       }
       if (c == 0)
          return 1;
       else
          return 0;
    }
}
```

Para cada instancia, el algoritmo termina con la salida correcta

¿Es el algoritmo primo correcto?

Sí

1	1
2	

Algoritmos en la computación

```
primo(int n){
 if (n == 1)
       return 1;
    else {
       int c = 0;
       for (int i = 3; i < n; i++) {
           if (n % i == 0)
               c++;
       }
       if (c == 0)
           return 1;
       else
           return 0;
    }
}
```

Para cada instancia, el algoritmo termina con la salida correcta

¿Es el algoritmo primo correcto?

n	sol
1	1
2	1
3	1
4	1

Algoritmos en la computación

Tipos de problemas solucionados utilizando algoritmos:

- Genoma humano: Identificar genes en secuencias de ADN que llegan hasta los 3200 millones de pares de bases nitrogenadas (A,T,C,G). ¿Que pasaría si lo hacemos manualmente?
- Búsquedas en Internet: Dada la cantidad de información indexada, responder de forma correcta la solicitud de una búsqueda en Internet. ¿Que pasaría si buscamos manualmente en cada página?

Algoritmos en la computación

Tipos de problemas solucionados utilizando algoritmos:

- Tratamiento de colisiones de objetos: Detectar una colisión entre dos cuerpos en un espacio 3D

- Búsquedas sobre videos: En un biblioteca, un usuario desea encontrar todos los videos donde aparezca la mascota de Univalle (La ardilla extraña)

Algoritmos en la computación

Análisis de algoritmos

Meta: Comparar algoritmos que resuelven un mismo problema

- Correctitud ← Ite
- Eficiencia
    - Tiempo $O(f(n))$ $\Omega(f(n))$ $\Theta(f(n))$
    - Espacio
- Estructuras de datos utilizadas
- Modelo computacional
- El tipo y número de datos con los cuales trabaja (escalabilidad)

Algoritmos en la computación

¿Cómo 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

¿Cómo 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

```
primo(int n) {
   if (n == 1)
      return 1;
   else {
      int c = 0;
      for (int i = 2; i < n; i++) {
         if (n % i == 0)
            c++;
      }
      if (c == 0)
         return 1;
      else
         return 0;
   }
}
```

¿De qué depende la cantidad de operaciones básicas que realizará el algoritmo para un llamado específico?

$n \geq 1$

$10 == 1$ } F

$100 == 1$ } F

$10^6 == 1$ } F

n	10	20	30	40	50
	9	19	29	39	49

$2<10$, $3<10$, $4<10$, …, $9<10$ } T

$\boxed{10<10}$ F

$2<10$ … $20<10$

7

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

$T_1(n)=3n^2$

1

$T_2(n)=6n^3$

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

Analicemos un ejemplo (Algoritmo de ordenamiento insertion-sort)

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		

¡Sin temor!, vamos a explorar este algoritmo.

Algoritmos en la computación

Este algoritmo recibe un arreglo de tamaño n y retorna el mismo arreglo ordenado de menor a mayor.

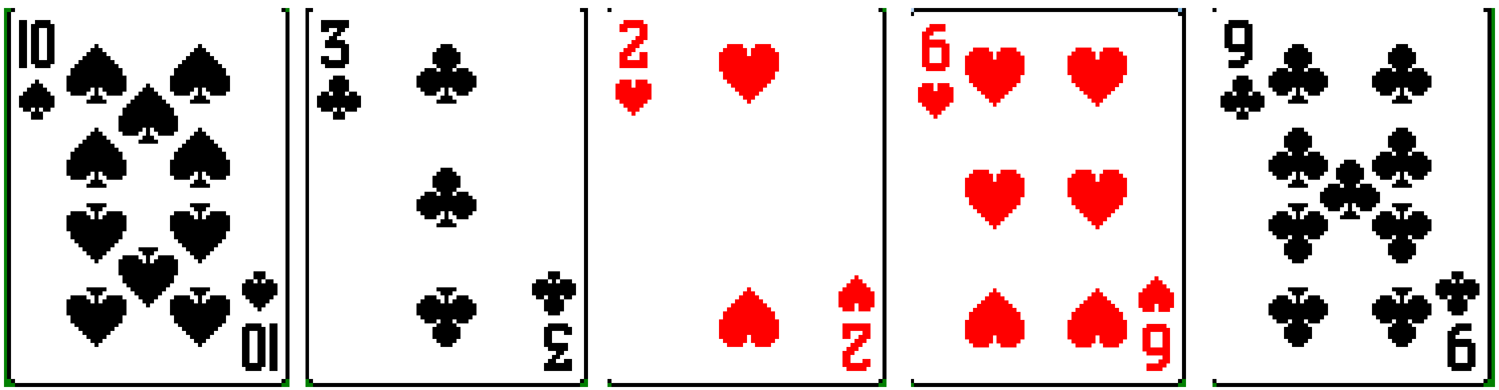
Ejemplo

Entrada = {10,3,2,6,9}

Salida = {2,3,6,9,10}

Algoritmos en la computación

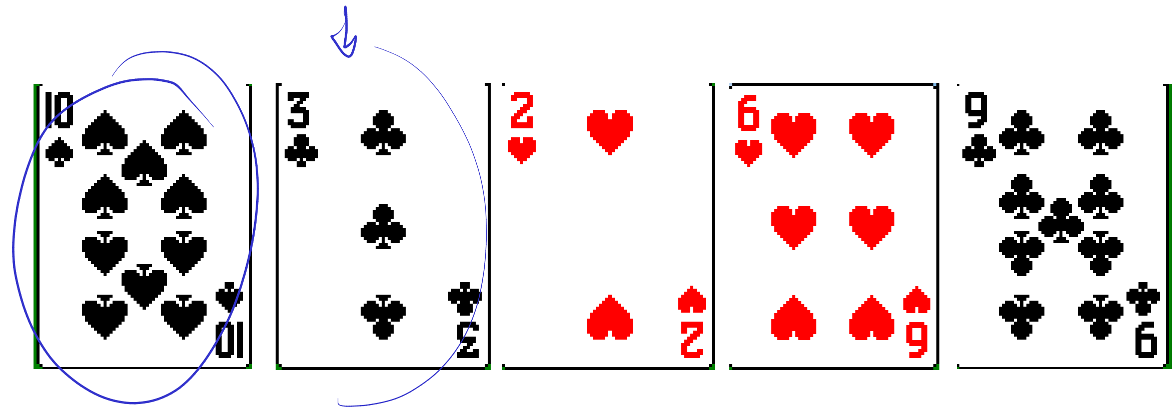
Insertion sort



Algoritmos en la computación

Insertion sort

i← 1 j← 2 Key=3



$\boxed{3\ 10}$ 2 6 9

3 10 2 6 9

3 2 10 6 9

$\boxed{2\ 3\ 10}$ 6 9

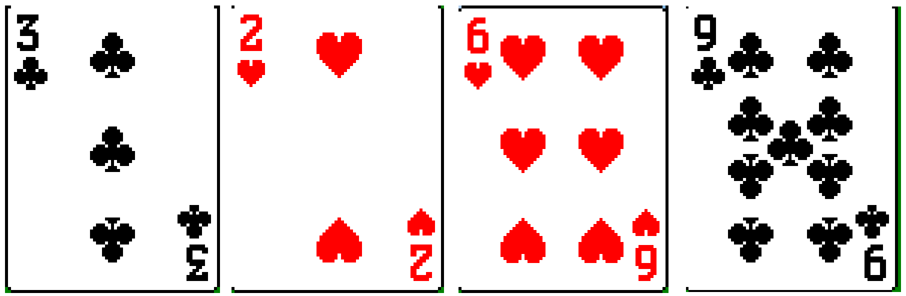
$\boxed{2\ 3\ 6\ 10}$ 9

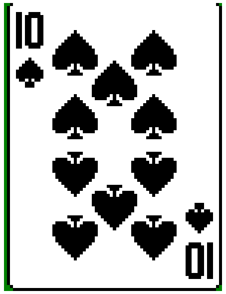
2 3 6 9 10

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

Insertion sort

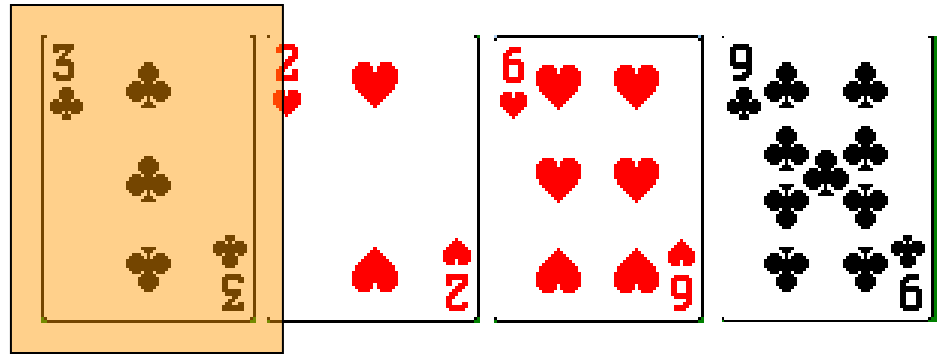


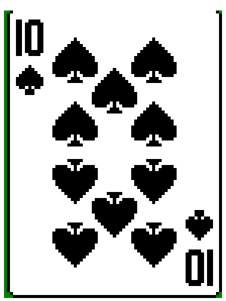


```
for j←2 to length[A]
    do key←A[j]
        i←j-1
        while i >0 and A[i] > key
            do A[i+1]←A[i]
                i←i-1
        A[i+1]←key
```

Algoritmos en la computación

Insertion sort

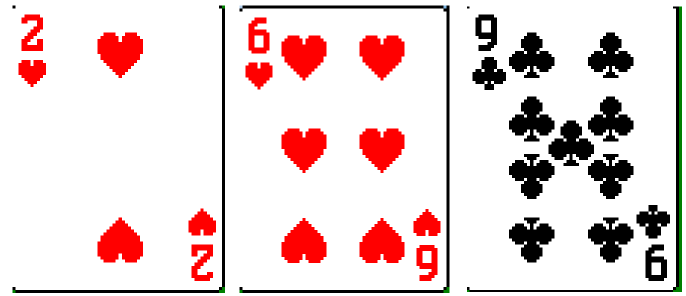


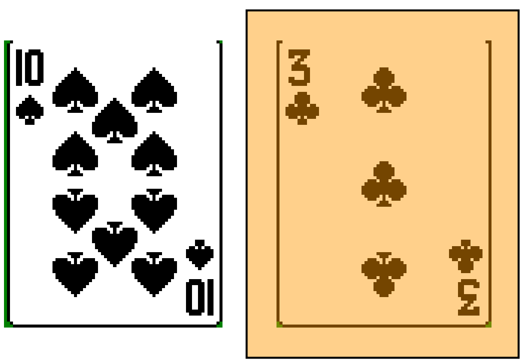


```
for j←2 to length[A]
    do key←A[j]
       i←j-1
       while i >0 and A[i] > key
           do A[i+1]←A[i]
              i←i-1
       A[i+1]←key
```

Algoritmos en la computación

Insertion sort



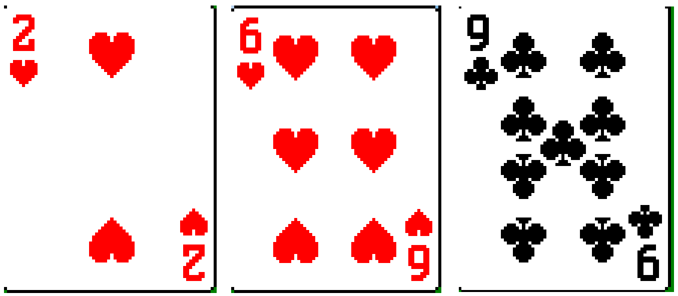


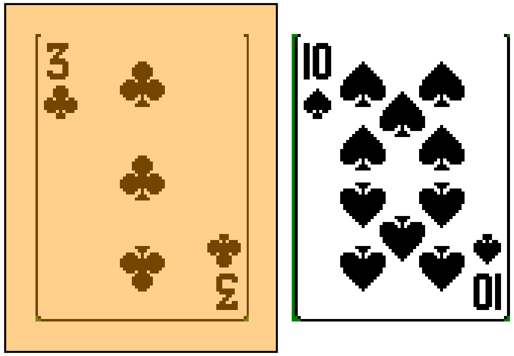
Se recorre de derecha a izquierda buscando <u>el lugar que debe ocupar</u>

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

Insertion sort

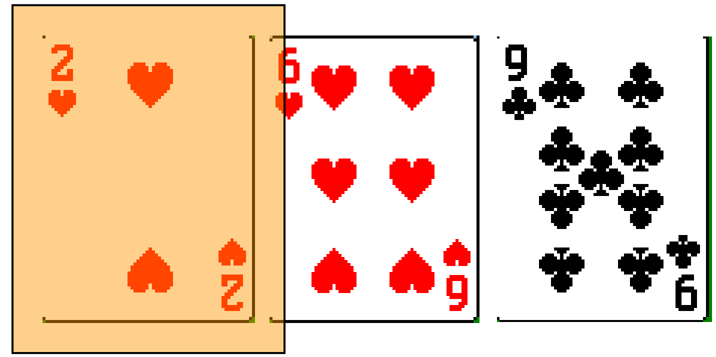


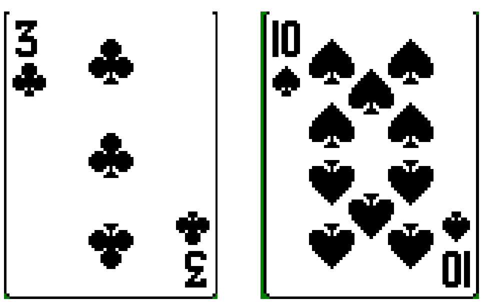


1	for j←2 to length[A]
2	do key←A[j]
3	i←j-1
4	while i >0 and A[i] > key
5	do A[i+1]←A[i]
6	i←i-1
7	A[i+1]←key

Algoritmos en la computación

Insertion sort

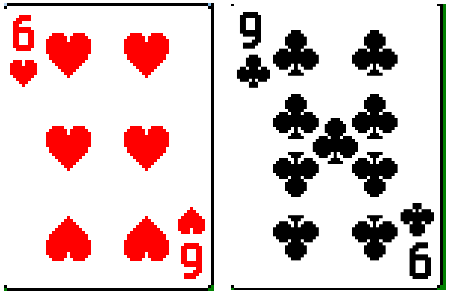


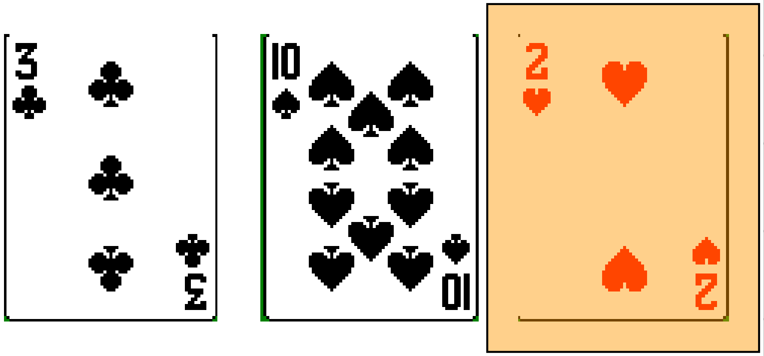


```
for j←2 to length[A]
    do key←A[j]
       i←j-1
       while i >0 and A[i] > key
           do A[i+1]←A[i]
              i←i-1
       A[i+1]←key
```

Algoritmos en la computación

Insertion sort

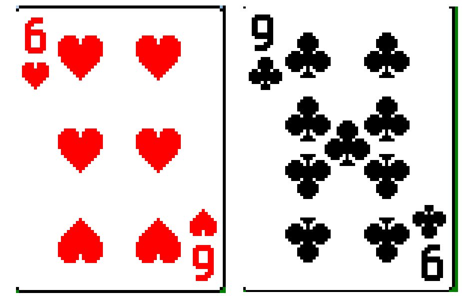


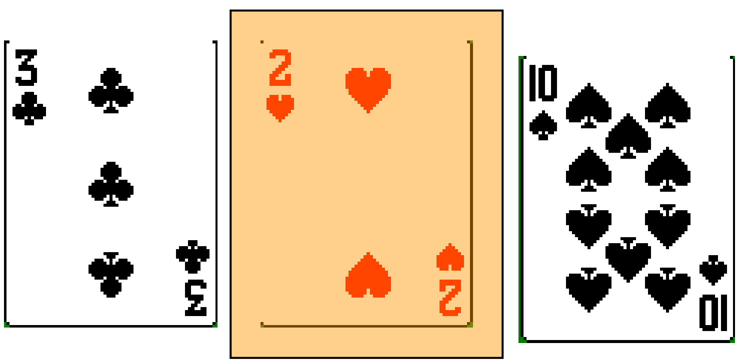



```
for j←2 to length[A]
      do key←A[j]
         i←j-1
         while i >0 and A[i] > key
               do A[i+1]←A[i]
                  i←i-1
         A[i+1]←key
```

Algoritmos en la computación

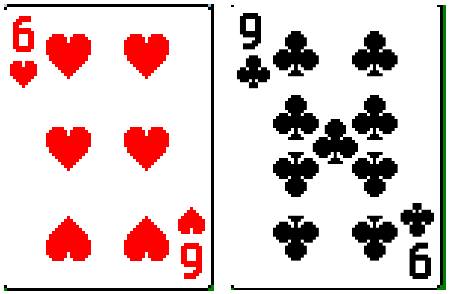
Insertion sort

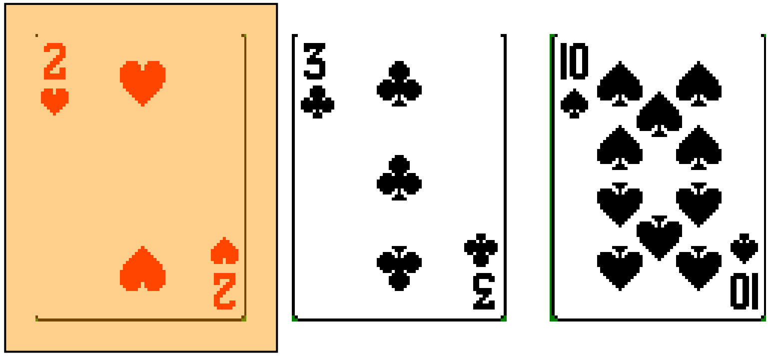




Algoritmos en la computación

Insertion sort

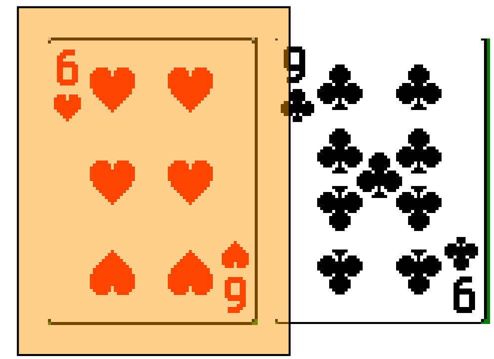


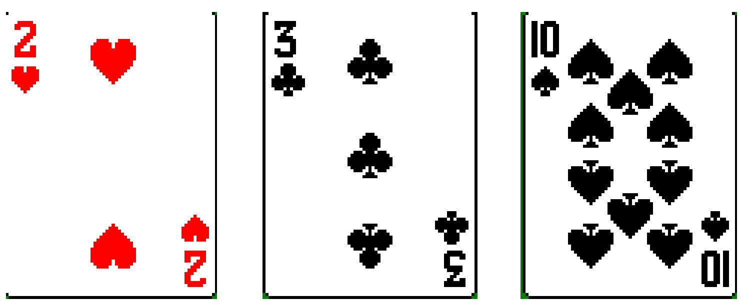


```
for j←2 to length[A]
    do key←A[j]
       i←j-1
       while i >0 and A[i] > key
           do A[i+1]←A[i]
              i←i-1
       A[i+1]←key
```

Algoritmos en la computación

Insertion sort

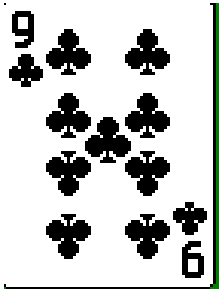


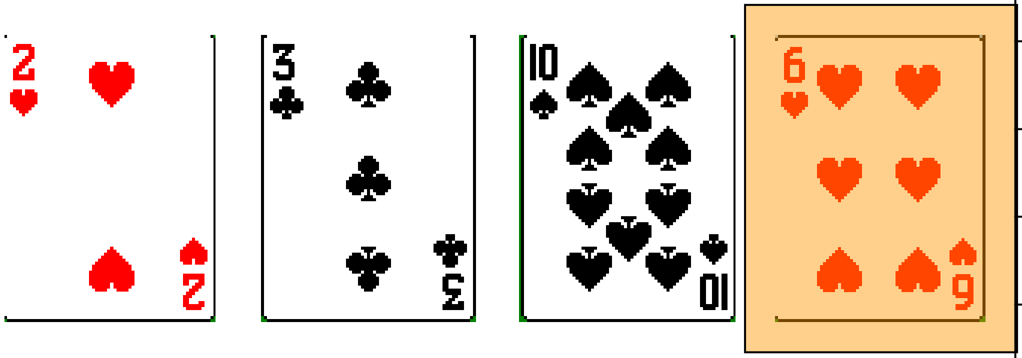


```
for j←2 to length[A]
    do key←A[j]
       i←j-1
       while i >0 and A[i] > key
           do A[i+1]←A[i]
              i←i-1
       A[i+1]←key
```

Algoritmos en la computación

Insertion sort

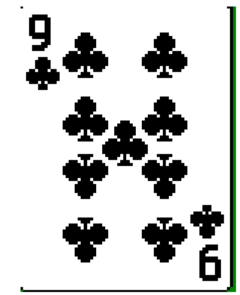


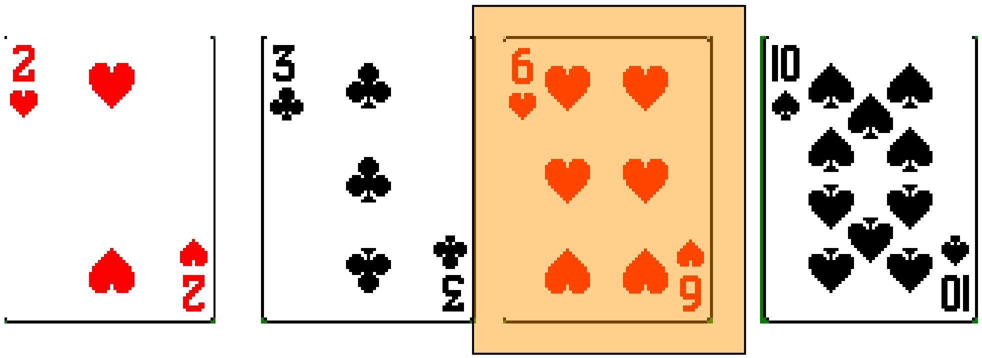


```
for j←2 to length[A]
        do key←A[j]
            i←j-1
            while i >0 and A[i] > key
                do A[i+1]←A[i]
                    i←i-1
            A[i+1]←key
```

Algoritmos en la computación

Insertion sort





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

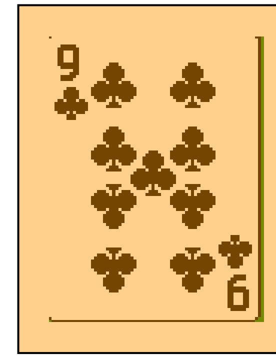
Insertion sort

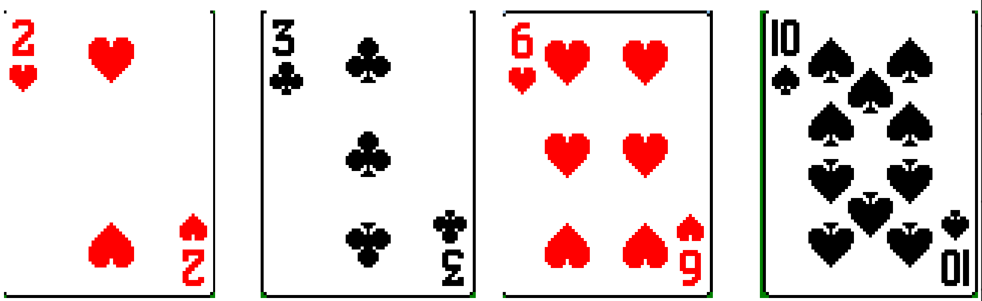
key=3

key

7 3 1 8

1 3 7 8

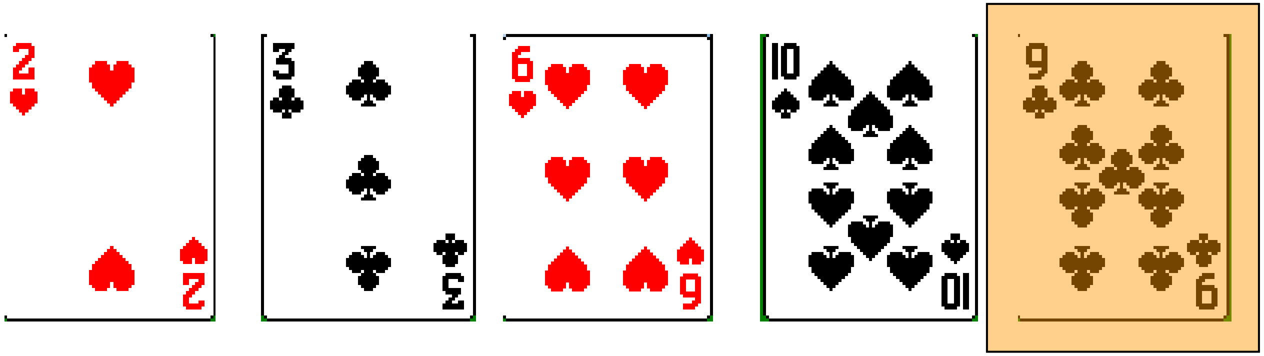




1	for j←2 to length[A]
2	do key←A[j]
3	i←j-1
4	while i >0 and A[i] > key
5	do A[i+1]←A[i]
6	i←i-1
7	A[i+1]←key

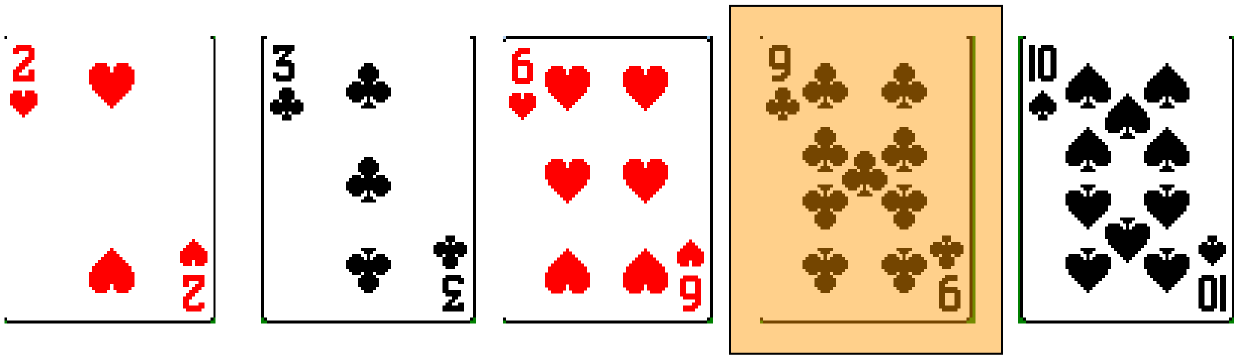
Algoritmos en la computación

Insertion sort



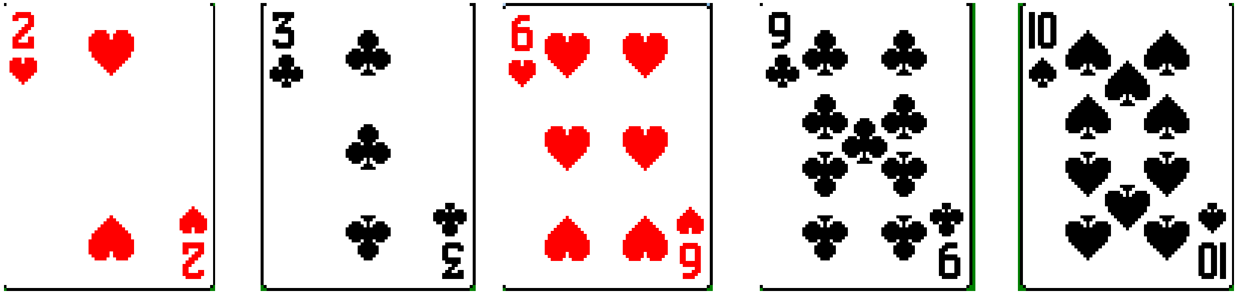
Algoritmos en la computación

Insertion sort



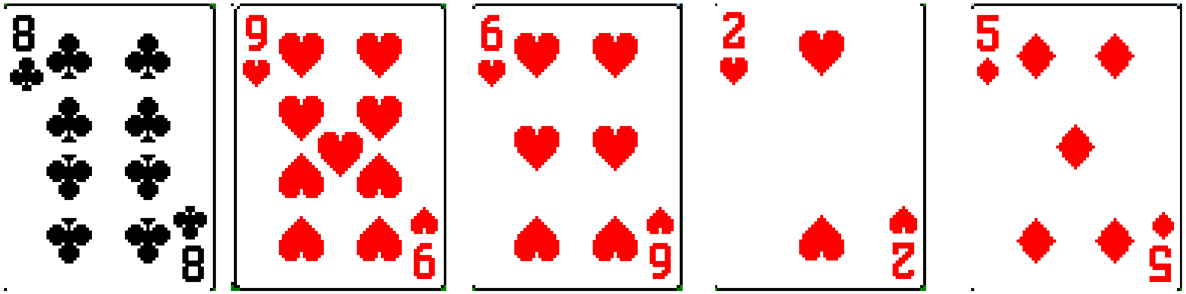
Algoritmos en la computación

Insertion sort



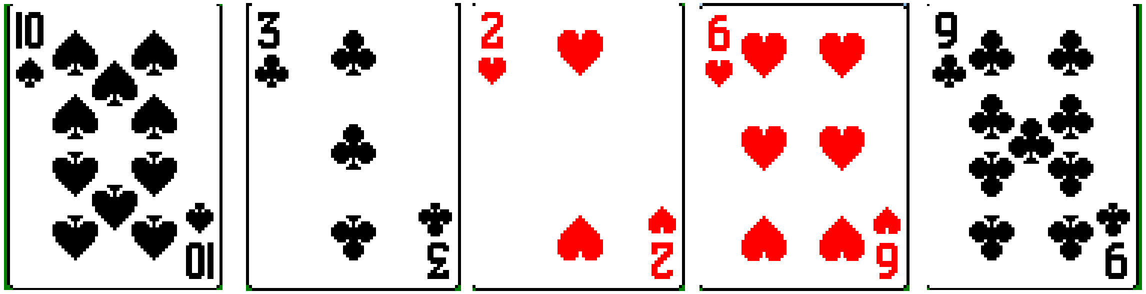
Algoritmos en la computación

Insertion sort



Algoritmos en la computación

Insertion sort



A:

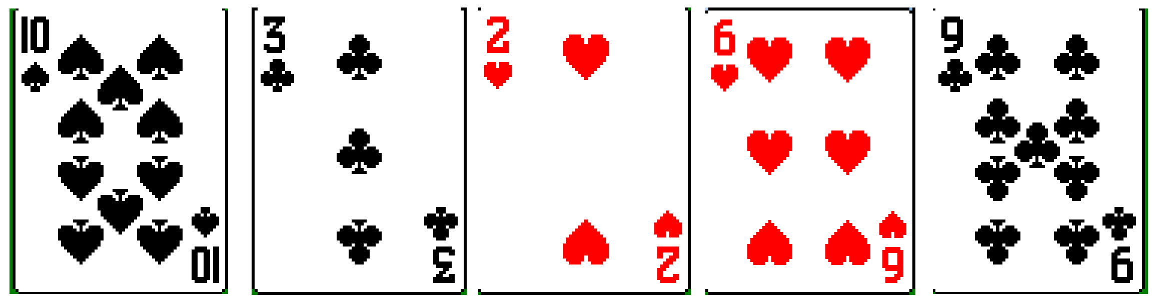
10	3	2	6	9

Algoritmos en la computación

Insertion sort

Desarrolle el algoritmo

INSERTION-SORT(A)



A:

10	3	2	6	9

Algoritmos en la computación

	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 $j=1$ $j \le n$ $j{+}{+}$	Costo	Veces que se repite
1 for j←1 to length[A]	c_1	
2 print A[j]	c_2	

Validos
$1 \le n$ ✓
$2 \le n$ ✓
$3 \le n$ ✓
⋮
$n-1 \le n$ ✓
$n \le n$ ✓

Salida
$n+1 \le n$ ✗

Algoritmos en la computación

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

$j = 1$

while ($j \leq 3$)

⋮

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 + validación condición falsa (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 + validación condición falsa (1)

Algoritmos en la computación

for j←1 to n

¿Cuántas veces se repite?

j

1
2
3
⋮
n
n+1 ← Salida

$\left.\begin{matrix}1\\2\\3\\\vdots\\n\end{matrix}\right\} n$

$n+1$

$5 \ldots n$ $\quad n+1$

$n-4+1$

$n-3$

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	n ?
2 print A[j]	c_2	n − 1 ?

n

j: 2, 3, 4, ⋮, n

$n-2+1 = n-1$

n+1

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:

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

Comparación inicial

Validación condición falsa

Algoritmos en la computación

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

???

$6 - 2 + 1 + 1 = 6$

incluye el inicial

Salida

j | 2, 3, 4, 5, 6 → 7

Algoritmos en la computación

for j←2 to n

$$\overset{\text{Validos}}{n-2+1} + \overset{\text{Salida}}{1}$$

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

$2, 3, 4, \ldots, n, n+1$

$n-1$ 1

	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	$n-1$

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	
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
for j←2 to length[A]	c_1	n
do key←A[j]	c_2	n-1
i←j-1	c_3	n-1
while i >0 and A[i] > key	c_4	
do A[i+1]←A[i]	c_5	
i←i-1	c_6	
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	$t_2 + t_3 + t_4 + \dots + t_n$
5 do A[i+1]←A[i]	c_5	$t_2 - 1 + t_3 - 1 + \dots + t_n - 1$
6 i←i-1	c_6	. . . .
7 A[i+1]←key	c_7	$n-1$

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

Los algoritmos debemos analizarlos, con respecto a:

- **Mejor caso:** Configuración de instancia(s) para las cuales el algoritmo realiza el menor número de pasos para dar la solución.
- **Peor caso:** Configuración de instancia(s) para las cuales el algoritmo realiza el mayor número de pasos para dar la solución.
- **Caso promedio:** Configuración típica o más frecuente de las instancias, este caso se puede analizar
    - Suponer configuraciones de instancias entre el mejor y peor caso, por ejemplo, si en el peor caso se hacen x comprobaciones y en el mejor 1 comprobación, suponer x/2 comprobaciones.
    - Con métodos estadísticos, para determinar la configuración esperada de las instancias

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] $\cancel{>}$ 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?

Key=3

2 3 6 10 19

Algoritmos en la computación

1	2	3	4	5
2	3	10	16	19

j=2

t_2=?

key = 3

i = 2

1) while 2>0 y 2>3 X

1 A[2] = 3 X

j = 3

key = 10

i = 2

1) while 2>0 y 3>10 X

j = 4 key = 16 i = 3 X

1) while 3>0 y 10>16

INSERTION-SORT(A)

```
for j←2 to length[A]
    do key←A[j]
       i←j-1
       while i >0 and A[i] > key
           do A[i+1]←A[i]
              i←i-1
       A[i+1]←key
```

j = 5 key = 19 i = 4

4>0 y 16>19 X

Algoritmos en la computación

2	3	10	16	19

j=3

t_3=?

Key = 10

i = 2

$t_3 = 1$

$2>0$ y $3>10$

j = 4

Key = 16

i = 3

$t_4 = 1$

$3>0$ y $10>16$

A[3] = 10

INSERTION-SORT(A)

```
for j←2 to length[A]
        do key←A[j]
            i←j-1
            while i >0 and A[i] > key
                  do A[i+1]←A[i]
                     i←i-1
            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.

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	$\sum_{j=2}^{n} 1$ 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.

T(n)=???

$$\sum_{j=1}^{n} c = c \times n$$

$$\sum_{j=2}^{n} 1 = \sum_{j=1}^{n-1} 1$$

$$1 \times (n-1)$$

Algoritmos en la computación

Para solucionar este caso, recordemos la forma cerrada de la sumatoria:

$$\sum_{i=1}^{n} C = C * n$$

Debido a que no la tenemos en la forma cerrada, debemos convertirla:

Entonces operando tenemos: $$\sum_{i=2}^{n} 1 = \boxed{\sum_{i=1}^{n} 1} - 1$$

$$\sum_{i=2}^{n} 1 = n - 1$$

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	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.

T(n) = n + 4(n – 1) = 5n - 4

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?

1 2 3 4 5 6

13 9 8 7 3 1

9 13 8 7 3 1

j = 2

Key = 9

i = 1

2

1 > 0 y 13 > 9

i = 0

0 > 0 y

j = 3

Key = 8

i = 2

13 > 8

i = 1

9 > 8

i = 0

3

1 2 3 4

8 9 13 7 3 1

j = 4 Key = 7

3 > 0 y 13 > 7

2 > 0 y 9 > 7

1 > 0 y 8 > 7

0 > 0 X

4

j = 5 i = 4

7 8 9 13 3 1

4 > 0 y 13 > 3

3 > 0 y 9 > 3

2 > 0 y 8 > 3

1 > 0 y 7 > 3

0 > 0 X

5

Algoritmos en la computación

3	2	1	6	9

j=2

t_2=?

j=2 key=2 i=1

1>0 y 3>2 ✓

3 3 1 6 9

i=0

0>0 ✗

4) 2

5,6) 1

2 3 1 6 9

```
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
```

j=3 key=1 i=2

• 2>0 y 2>1 ✓

2 3 3 6 9

i=1

1>0 y 2>1 ✓

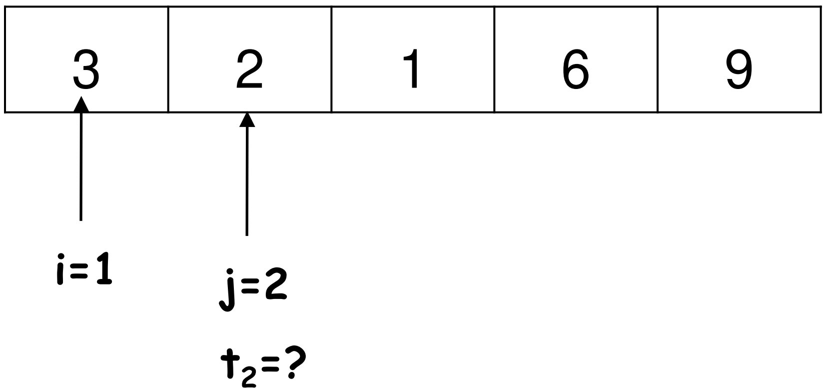
2 2 3 6 9

i=0 0>0 ✗

1 2 3 6 9

8) 3

Algoritmos en la computación



INSERTION-SORT(A)

```
for j←2 to length[A]
        do key←A[j]
           i←j-1
           while i >0 and A[i] > key
                  do A[i+1]←A[i]
                     i←i-1
           A[i+1]←key
```

Algoritmos en la computación

2	3	1	6	9

$i=0$

$j=2$

$t_2=?$

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

Algoritmos en la computación

2	3	1	6	9

j=3

t_3=?

```
INSERTION-SORT(A)
for j←2 to length[A]
    do key←A[j]
       i←j-1
       while i >0 and A[i] > key
           do A[i+1]←A[i]
              i←i-1
       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	$\sum_{j=2}^{n} j = 2+3+4+\dots+n$
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 peor de los casos, $t_j=j$.

T(n)=???

$$\sum_{j=2}^{n} j \longrightarrow \sum_{j=1}^{n-1} (j+1)$$

$$2+3+4+\dots+n \quad = \quad 2+3+4+5+\dots+n$$

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

$$\sum_{j=1}^{n-1} j + \sum_{j=1}^{n-1} 1$$

$$\downarrow$$

$$\frac{(n-1)(n)}{2} + \frac{2(n-1)}{2} = \frac{n(n-1)+2(n-1)}{2}$$

$$\frac{(n+2)(n-1)}{2}$$

$$\sum_{j=1}^{n} c = n \times c$$

$$\sum_{j=2}^{n} (j-1) = \sum_{j=1}^{n-1} (j+1-1) = \sum_{j=1}^{n-1} j$$

$$\frac{(n-1)(n)}{2}$$

Algoritmos en la computación

Para solucionar este caso, recordemos la forma cerrada de la sumatoria:,

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=1}^{n} C = C * n$$

Debido a que no la tenemos en la forma cerrada, debemos convertirla:

$$\sum_{j=2}^{n} 1 = \sum_{j=1}^{n} 1 - 1$$

Entonces operando tenemos:

$$\boxed{\sum_{j=2}^{n} (j-1) = \sum_{j=2}^{n} j - \sum_{j=2}^{n} 1}$$

$$\sum_{j=2}^{n} 1 = n - 1 \qquad \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

Dividimos la sumatoria Aprovechamos el anterior caso

$$\boxed{\sum_{j=2}^{n} (j-1) = \frac{n(n+1)}{2} - \cancel{1} - (n - \cancel{1})}$$

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	$\frac{(n+1)(n+2)}{2}$ $\frac{n(n+1)}{2}-1$
5 do A[i+1]←A[i]	c_5	$\frac{(n)(n-1)}{2}$ $\frac{n(n+1)}{2}-n$
6 i←i-1	c_6	$\frac{(n)(n-1)}{2}$ $\frac{n(n+1)}{2}-n$
7 A[i+1]←key	c_7	n-1

En el peor de los casos, $t_j=j$.

$T(n)= n + 3(n – 1) + 0,5*3(n (n + 1)) – 2n – 1$

$T(n) = 1,5n^2 + 3,5n - 4$

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, esto es, $t_j=j/2$.

T(n)=???

$$\sum_{j=2}^{n} j/2 = \sum_{j=1}^{n-1}\left(\frac{j}{2}+\frac{1}{2}\right) = \frac{1}{2}\sum_{j=1}^{n-1} j + \frac{1}{2}\sum_{j=1}^{n-1} 1$$

$$\frac{(n-1)n}{4} + \frac{2(n-1)}{4} = \frac{(n+2)(n-1)}{4}$$

$$\sum_{j=2}^{n} \frac{j}{2} - 1 = \sum_{j=1}^{n-1}\left(\frac{j}{2}+\frac{1}{2}-\frac{2}{2}\right) = \sum_{j=1}^{n-1}\left(\frac{j}{2}-\frac{1}{2}\right)$$

$$\frac{(n-1)n}{4} - \frac{2(n-1)}{4} = \frac{(n-2)(n-1)}{4}$$

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	$\frac{n(n+1)}{4}-\frac{1}{2}$
5 do A[i+1]←A[i]	c_5	$\frac{n(n+1)}{4}-n$
6 i←i-1	c_6	$\frac{n(n+1)}{4}-n$
7 A[i+1]←key	c_7	n-1

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

$T(n)= n + 3(n-1) + 0.25*3(n (n + 1)) - 0.5 - 2n$

$T(n) = 0{,}75n^2 + 2{,}75n - 3{,}5$

Algoritmos en la computación

Calcule el tiempo de cómputo para el algoritmo

def programa1(mat1, mat2)

suponga que len(mat1)=len(mat2)=n

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

Instrucción	Costo	
1 i=1	c_1	1
2 while i<=len(mat1) $1, 2, 3, \ldots, n$	c_2	$1, 2, 3, 4, 5, \ldots, n, n+1 \} \; n+1$
3 j←1	c_3	n
4 while j<=len(mat2) $1 \ldots n$ $(n+1)$	c_4	$\sum_{i=1}^{n} t_i = n(n+1)$
5 mat3[i][j]←mat1[i][j]+mat2[i][j]	c_5	$\sum_{i=1}^{n} (t_i - 1) = n^2$
6 j←j+1	c_6	$\sum_{i=1}^{n} (t_i - 1) = n^2$
7 i←i+1	c_7	n

$i=1 \quad 1, \ldots, n, n+1 \} \; (n+1)$

$i=2 \quad 1, \ldots, n, n+1 \} \; (n+1)$

$i=3 \quad 1, \ldots, n, n+1 \} \; (n+1)$

$i=n \quad 1, \ldots, n, n+1 \} \; (n+1)$

$$t_i = (n+1)$$

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

$$\sum_{i=1}^{n} n = n^2$$

Algoritmos en la computación

Calcule el tiempo de cómputo 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}

i =

1 2 3 4 5 ... n

t_i

2 + 3 + 4 + ... + n+1

1 2 3 ... n

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

Instrucción	Costo	
1 s←0	c_1	1
2 i←1	c_2	1
3 while i<=n	c_3	$1, 2, 3, \ldots, \boxed{n}, n+1 \Rightarrow n+1$
4 t←0	c_4	n
5 j←1	c_5	n
6 while j<=i	c_6	$t_i \Rightarrow t_1 + t_2 + t_3 \ldots + t_n$
7 t←t+1	c_7	$t_i - 1$
8 j←j+1	c_8	$t_i - 1$
9 s←s+t	c_9	n
10 i←i+1	c_{10}	n

j \ i	1	2	3	n
	1	1	1	1
	2	2	2	2
		3	3	3
			4	4
				⋮
				n+1

$\boxed{t_i = i+1}$

6 $\sum_{i=1}^{n} i+1 = \frac{n(n+1)}{2} + n$

7, 8 $\sum_{i=1}^{n} i = n\left(\frac{n+1}{2}\right)$

Algoritmos en la computación

Calcule el tiempo de cómputo 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

Calcule el tiempo de cómputo para el algoritmo

def programa4(n)

Suponga n impar

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+2	c_5
6 k←1	c_6
7 while k<=i	c_7
8 k←k+1	c_8
9 i←i+2	c_9

Algoritmos en la computación

Calcule el tiempo de cómputo para el algoritmo

def programa5(n)

Suponga n como potencia de 2, es decir tiene la forma 2^x con x >= 0

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+2	c_5
6 k←1	c_6
7 while k<=i	c_7
8 k←k+1	c_8
9 i←2i	c_9

programa(8)

```
i←1
while i<=n
    k←i
    while k<=n
        k← k+2
    k←1
    while k<=i
        k←k+1
    i←2i
```

$n = 2^x$

1 $\quad 2^0,\ 2^1,\ 2^2,\ 2^3,\ 2^4,\ 2^5,\ 2^6,\ \dots,\ 2^x,\ 2^{x+1}$

$1,\ 2,\ 4,\ 8,\ 16,\ 32,\ 64,\ \dots,\ n,\ 2n$

$1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ \dots,\ \log_2(n)+1,\ \log_2(n)+2$

$m = \log_2(i)+1$

$\boxed{\log_2(n)+2}$

$\log_2(n)+1$

$\log_2(n)+1$

$\log_2(n)+1$

```
i←1
while i<=n
    k←i
    while k<=n
        k← k+2
    k←1
    while k<=i
        k←k+1
    i←2i
```

$w = \log_2(i)+1$

$i = 1, 2, 4, 8, \ldots, n$

$w = 1, 2, 3, \ldots, \log_2(n)+1$

2^x

i / k	1	2	4	8	16 ... n
	1	2	4	8	16
	3	4	6	10	18
	5	6	8	12	20
	7	8	10	14	
	n−1	n	n	n	n
	n+1	n+2	n+2	n+2	n+2

g

2

4

8

n

n+2

$\frac{n}{2}+1-? = 2$

$2, 4, 6, 8, \boxed{10}, 12, \quad n=10, \quad n=12$

$2, 4, 6, 8, 10, 12, 14$

$1, 2, 3, 4, 5, 6$

$1, 2, 3, 4, 5, 6, 7$

$1, 2, 3, 4, \ldots \frac{n}{2}, \frac{n}{2}+1$

$\frac{n}{2}+1-0 \quad \frac{n}{2}+1-1 \qquad \frac{n}{2}+1-3$

$T_i = \boxed{\frac{n}{2}+1-\frac{i}{2}+1} \qquad \frac{n}{2}+1-7$

$$t_w = \frac{n}{2} + 1 - \frac{2^{w-1}}{2} + 1$$

$\rightarrow w = \log(i) + 1$

$w - 1 = \log_2(i)$

$$\boxed{2^{w-1} = i}$$

$t_1 + t_2 + t_3 + \dots + t_{\log_2(n)+1}$

$$\sum_{w=2}^{\log_2(n)+1} t_w = \sum_{w=2}^{\log(n)+1} \overbrace{\frac{n}{2} + 1}^{\text{Cte}} - \frac{2^w}{4} \overbrace{+1}^{\text{Cte}}$$

$$\sum_{w=2}^{\log(n)+1} \frac{n}{2} + 2 - \frac{1}{4} \sum_{w=2}^{\log(n)+1} 2^w = \sum_{w=1}^{\log_2(n)} \frac{n}{2} + 2 - \frac{1}{4} \sum_{w=0}^{\log_2(n)-1} 2^{w+2}$$

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

$$\left(\frac{n}{2} + 2\right) \log_2(n) - \cancel{\frac{1}{4}} \sum_{w=0}^{\log_2(n)-1} \cancel{4} \times 2^w$$

$$\left(\frac{n}{2} + 2\right) \log_2(n) - \frac{2^{\log_2(n)} - 1}{2 - 1}$$

$$\left(\frac{n}{2} + 2\right) \log_2(n) - n + 1$$

$$a^{\log(b)} = b^{\log(a)}$$

$$\boxed{\left(\frac{n}{2} + 2\right) \log_2(n) - n + 1}$$

1 3 5 7 $n-1$ $n+1$

n	2	4	8	16	32
	1	1	1	1	1
	3	3	3	3	3
		5	5	5	5
			7	7	.
			9	9	.
				11	.
				13	33
				15	
				17	
	2	3	5	9	17

$\frac{n}{2} + 1$

$i = 2, 4, 8, \dots n$

$$\left(\frac{n}{2} + 2\right) \log_2(n) - n + 1 + \overbrace{\frac{n}{2} + 1}^{\text{ok} \; i=1}$$

$t_w - 1$

5)

$$\sum_{w=2}^{\log(n)+1} \frac{n}{2} + 1 - \frac{2^w}{4}$$

$$\underbrace{\left(\frac{n}{2} + 1\right) \log_2(n)}_{} - n + 1 + \frac{n}{2} + 1$$

```
i←1
while i<=n
    k←i
    while k<=n
        k← k+2
    k←1
    while k<=i
        k←k+1
    i←2i
```

$w = \log_2(i)+1$

$w = 1, 2, 3, 4, 5, 6 \ldots n$

$i = 1, 2, 4, 8, 16, 32, \ldots, n$

k						
1	1	1	1	1	1	1
2	2	2	2	2	2	2
	3	3	3	3	3	⋮
		4	4	⋮	⋮	n
			⋮	16	32	n+1
			8	17	32	
			9			

$t_i = i+1$

$w = \log(i)+1 \qquad w-1 = \log(i)$

$i = \frac{2^w}{2}$

$t_w = \frac{2^w}{2}+1$

$$\sum_{w=1}^{\log_2(n)+1} \frac{2^w}{2}+1 = \sum_{w=1}^{\log_2(n)+1} 1 + \frac{1}{2}\sum_{w=1}^{\log_2(n)+1} 2^w$$

$$(\log_2(n)+1 + \frac{2}{2}\sum_{w=0}^{\log(n)} 2^w = \log_2(n)+\cancel{1}+\frac{2^{\log(n)+1}-\cancel{1}}{2-1}$$

$$\log_2(n)+2\times n^{\log_2(2)}-1 = \boxed{\log_2(n)+2n}$$

$t_{w-1} = \frac{2^w}{2}$

$$\frac{1}{2}\sum_{w=1}^{\log_2(n)+1} 2^w = \frac{1}{2}\sum_{w=0}^{\log_2(n)} 2\times 2^w = \sum_{w=0}^{\log_2(n)} 2^w = \frac{2^{\log_2(n)+1}-1}{2-1}$$

$$2\times 2^{\log_2(n)+1}-1 = 2\times n^{\log_2(2)}-1 = \boxed{2n-1}$$

Algoritmos en la computación

Diseño de algoritmos

Otras alternativas para el diseño de algoritmos son:

- Solución ingenua
- Dividir y conquistar
- Programación dinámica
- Técnicas voraces

Algoritmos en la computación

Análisis de algoritmos ordenamiento

Computador de la Abuela
10^9 instrucciones/seg (100MHz)

Implementación 1	Implementación 2
$2n^2$	$50n*\lg n$

Ordenar un arreglo de 10^8 números

Tiempo 1	Tiempo 2
$2(10^8)^2/10^9=2x10^7$segs=5555,6horas	$(50*10^8\lg 10^{8)}/10^9$=40segs=0,66 mins

Algoritmos en la computación

Análisis de algoritmos ordenamiento

Computador última generación
10^{11} instrucciones/seg (10GHz)

Implementación 1	**Implementación 2**
$2n^2$	$50n*\lg n$

Ordenar un arreglo de 10^8 números

Tiempo 1	**Tiempo 2**
$2(10^8)^2/10^{11}=2x10^5$segs=55,56horas	$(50*10^8\lg 10^{8)}/10^{11}$=0,4segs

Referencias

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press. Pages 5-29

Gracias

Próximo tema:

Computación iterativa:

- Concepto de estado
- Transición de estados
- Invariante de ciclo

- Creditos

Creador material: Oscar Bedoya, PhD.
Modificación: Carlos A Delgado, Msc Feb 2023