Fundamentos de análisis y diseño de algoritmos

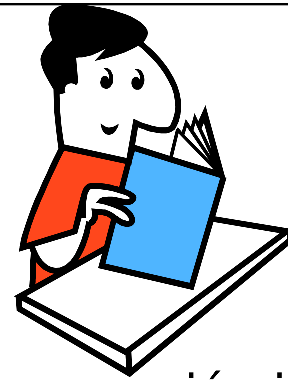
Universidad del Valle

Facultad de Ingeniería

Escuela de Ingeniería de sistemas y computación

Agosto 2017

Motivación



Programación ingenua

Programación dinámica

Programación voraz

Motivación

Estrategias de programación:

- **Programación ingenua:** Probar todas las posibles soluciones y mirar cual es la correcta
- **Divide y vencerás:** Partir el problema en problemas más pequeños e intentar solucionar estos
- **Programación dinámica:** Un divide y vencerás mejorado
- **Programación voraz:** Una buena estrategia.

Motivación

Pensemos este problema:

- Tus padres te han dejado al cuidado de la tienda familiar y te han pedido que al dar las vueltas trates de dar el menor número de monedas.
- Tienes las siguientes monedas {10,10,20,20,30,30,40,50,50}

Conjunto de tamaño 0, 1 conjuntos
Conjunto de tamaño 1, 9 conjuntos

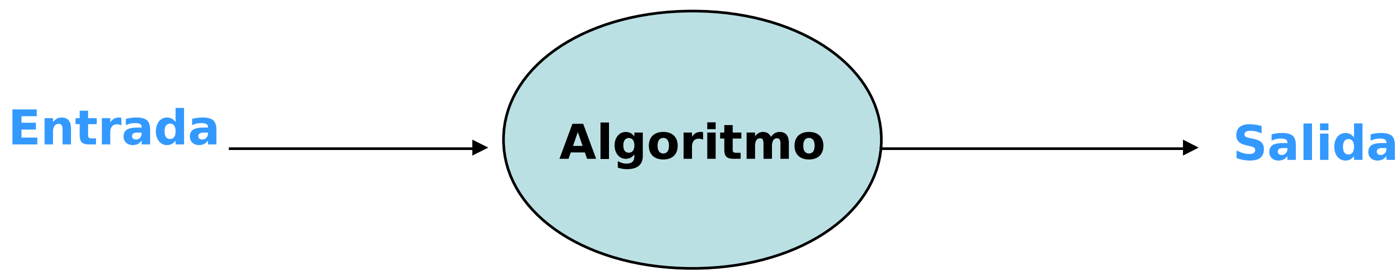
... Conjunto --> 2^9

- Doña Lupe, compra una papaya, esta vale 140 y te da 200, debes devolverle 60.

Bajo una estrategia ingenua ¿Cómo se resuelve este problema? ¿Podemos hacerlo sin tener que hacer tanto proceso?

Algoritmos en la computación

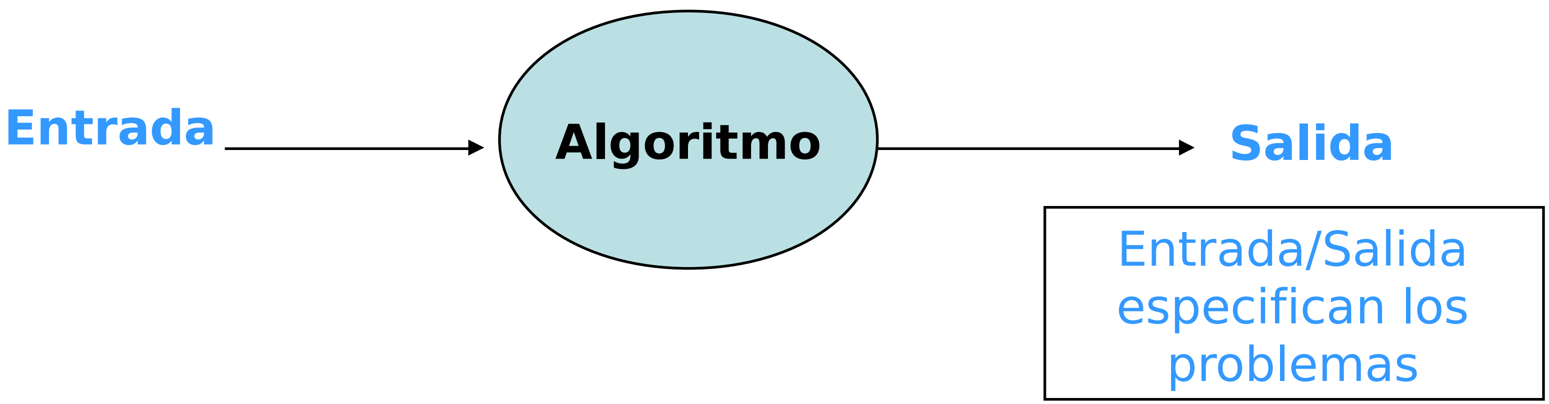
Un algoritmo es un procedimiento computacional que toma un valor o conjunto de valores como entrada y produce un valor o conjunto de valores como salida



“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



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

Algoritmos en la computación

Problema 1: Cambio de monedas

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

Salida: $<m_1, m_2, \ldots, m_n>$ tal que $\sum m_i * a_i = P$ y $\sum m_i$

es mínimo

Algoritmos en la computación

Problema 2: ???   Números primos

División exacta
Aquella su residuo es 0

Entrada: $n \in Z^+$

Salida: 1, si $n=1$

1, si $n=2$

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

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

Algoritmos en la computación

Problema 3: ???

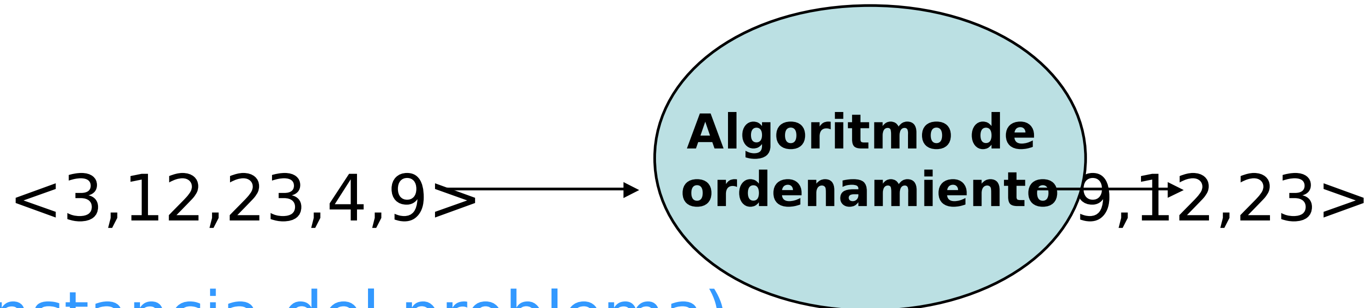
Problema de ordenamiento
(Menor a mayor)

Entrada: $S = \langle a_1, a_2, \ldots, a_n \rangle$, $a_i \in \mathbb{N}$, $1 \leq i \leq n$

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

Algoritmos en la computación

Instancia de un problema



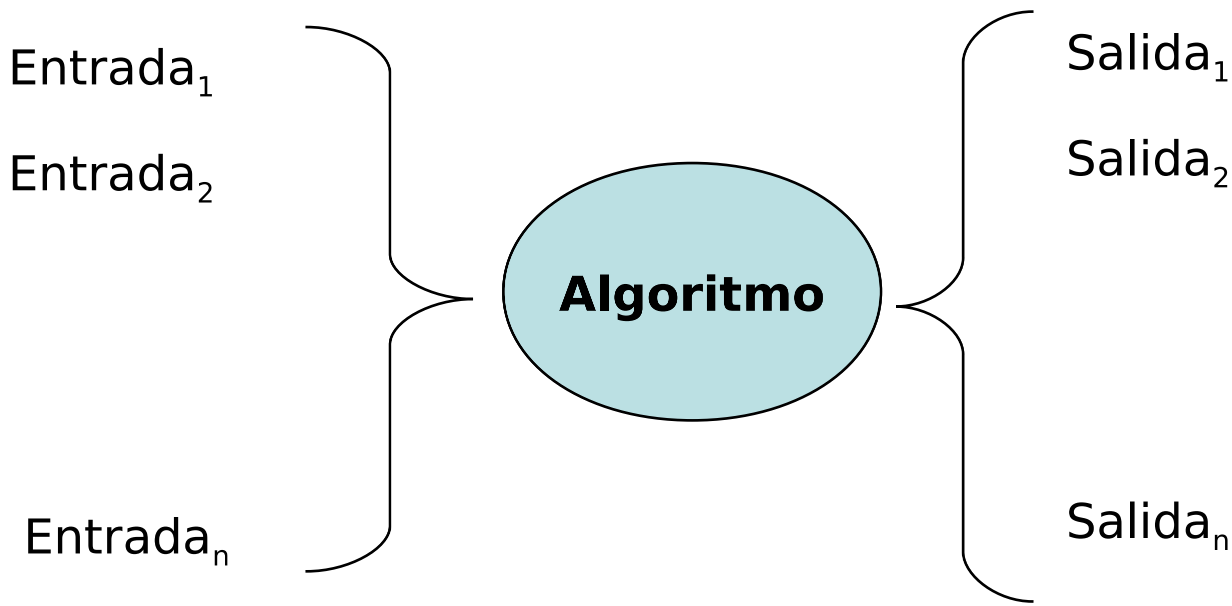
(instancia del problema)

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;
  }
```

$n=2$

Para cada instancia, el algoritmo termina con la salida correcta

¿Es el algoritmo <u>primo</u> correcto?

Instancias del algoritmo

$n \in Z^+$

Este algoritmo no es correcto,porque no tiene la respuesta correcta a todas las instancias

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;
 }
```

$n \in \mathbb{Z}^{+}$

Es el número de divisores entre 2 y n - 1

Para cada instancia, el algoritmo termina con la salida correcta

¿Es el algoritmo primo correcto?

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?

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
- Eficiencia
  - Tiempo Número de pasos / ejecuciones
  - Espacio Espacio en memoria requerido para la ejecución
- 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 $O(f(n))$
- 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

Complejidad de sus operaciones

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?

Depende de n

¿Porqué?

El número de ciclos en el for (iteraciones) depende del n

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

Analicemos un ejemplo (Algoritmo de ordenamiento insertion-sort)

Instrucción	Costo	Veces que se repite
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		

¡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.

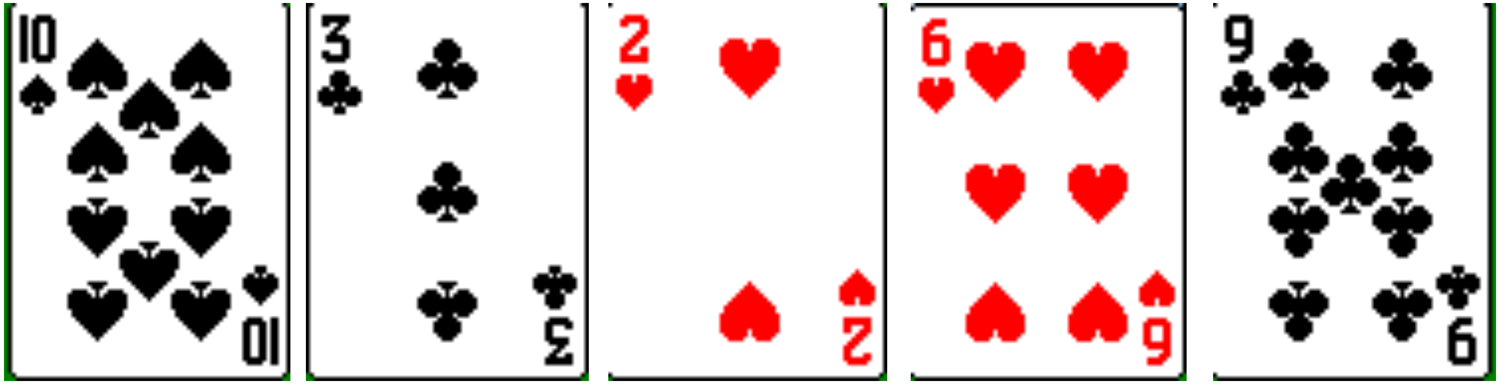
Ejemplo

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

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

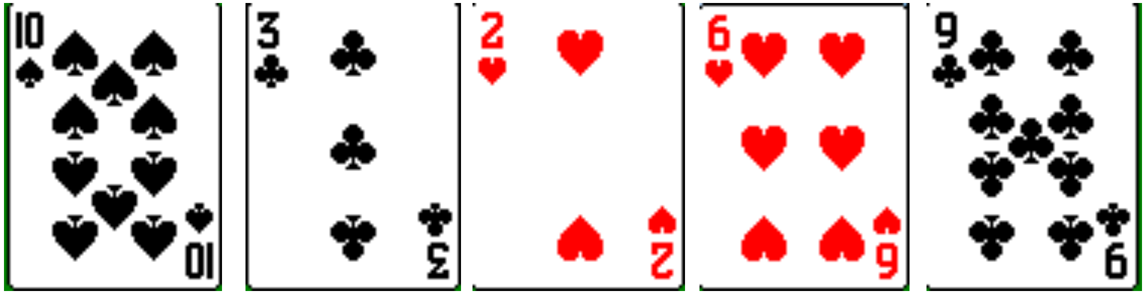
Algoritmos en la computación

Insertion sort



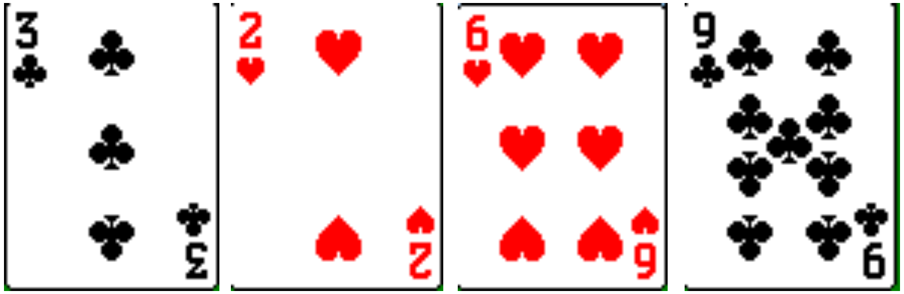
Algoritmos en la computación

Insertion sort



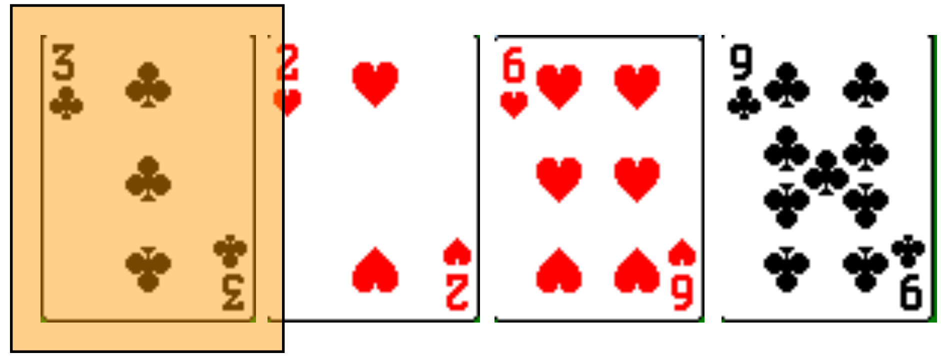
Algoritmos en la computación

Insertion sort



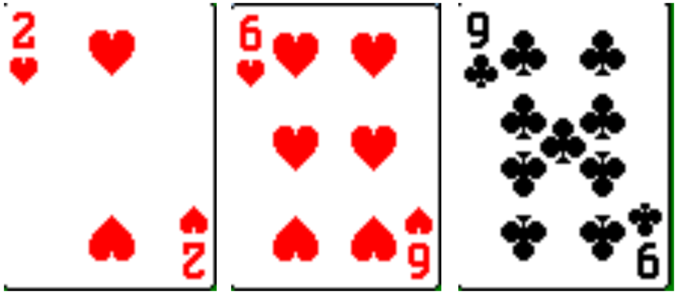
Algoritmos en la computación

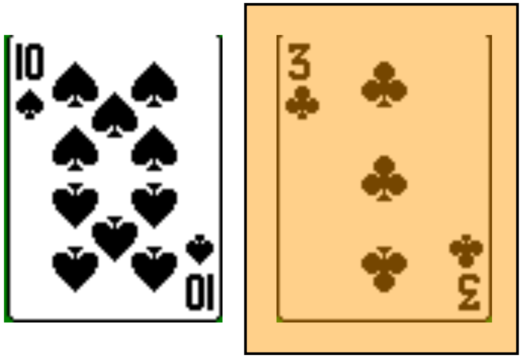
Insertion sort



Algoritmos en la computación

Insertion sort

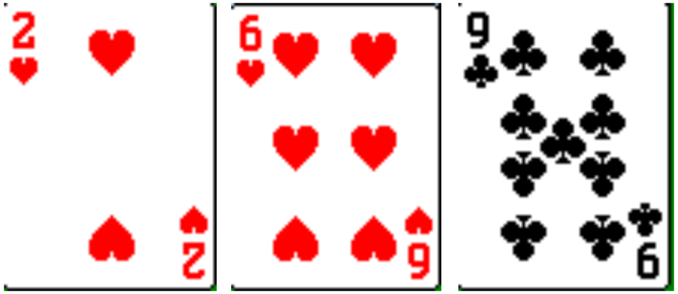


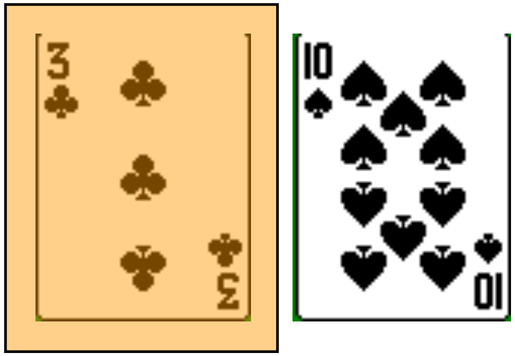


Se recorre de derecha a izquierda buscando el lugar que debe ocupar

Algoritmos en la computación

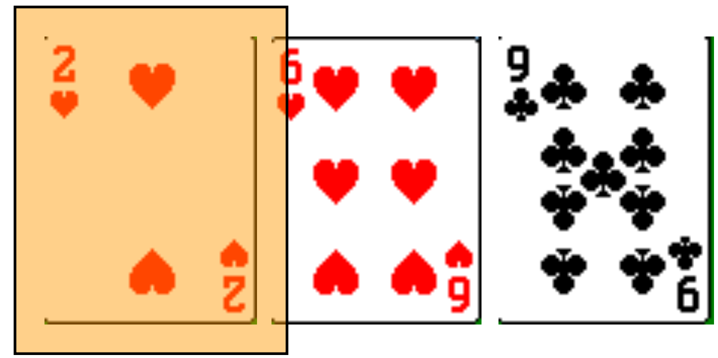
Insertion sort

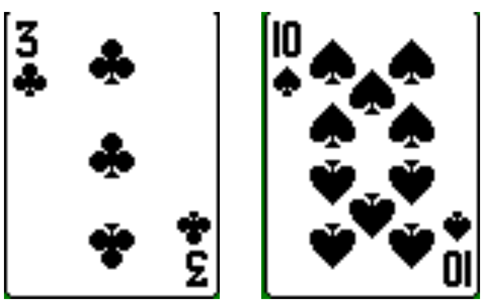




Algoritmos en la computación

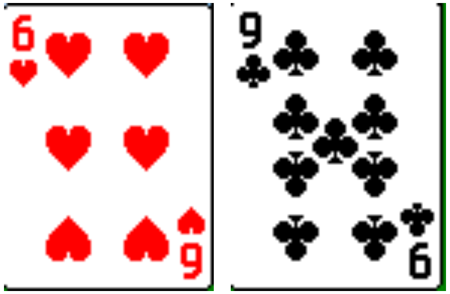
Insertion sort

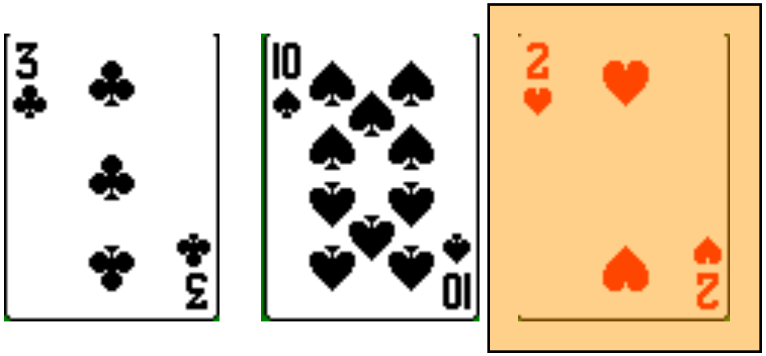




Algoritmos en la computación

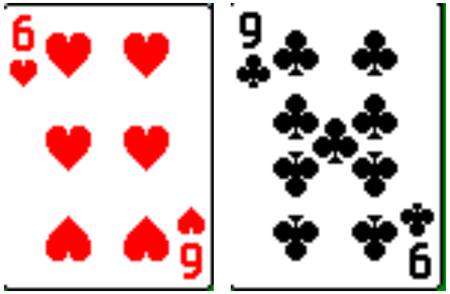
Insertion sort

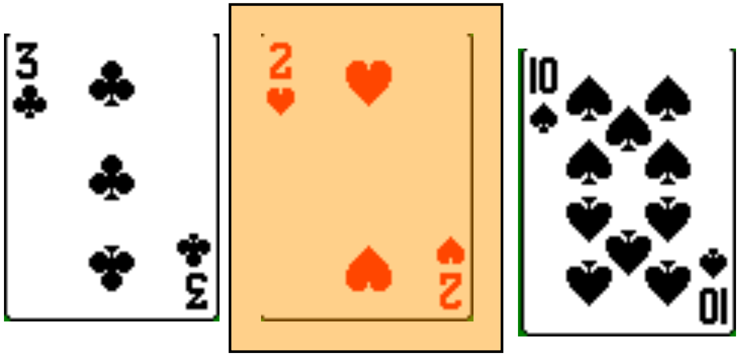




Algoritmos en la computación

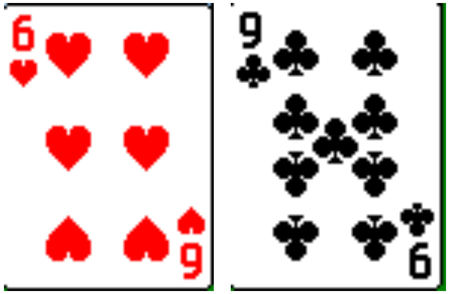
Insertion sort

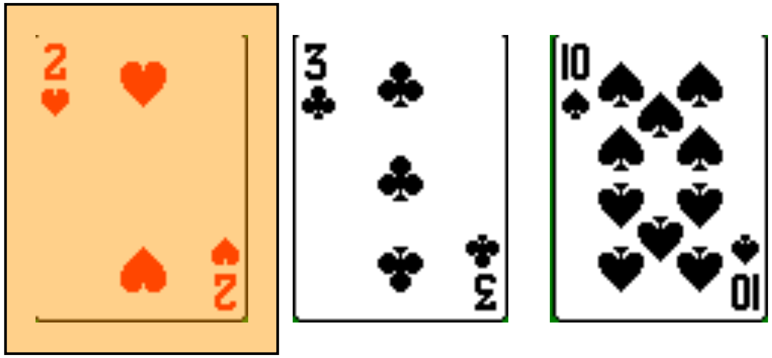




Algoritmos en la computación

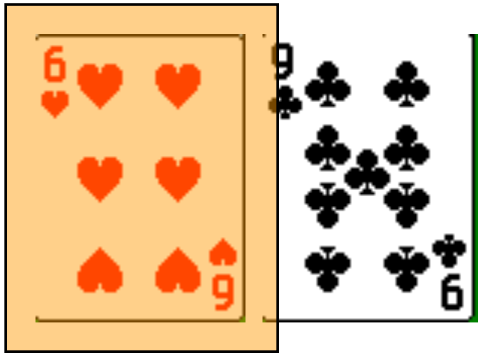
Insertion sort

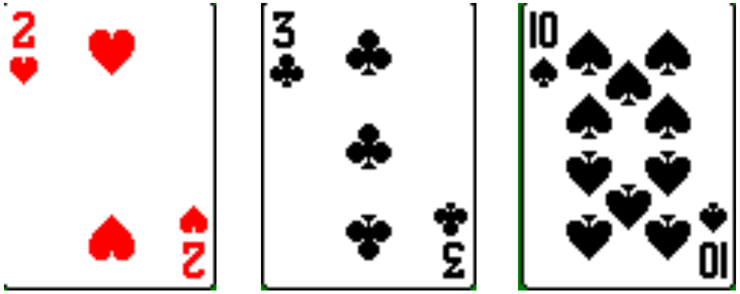




Algoritmos en la computación

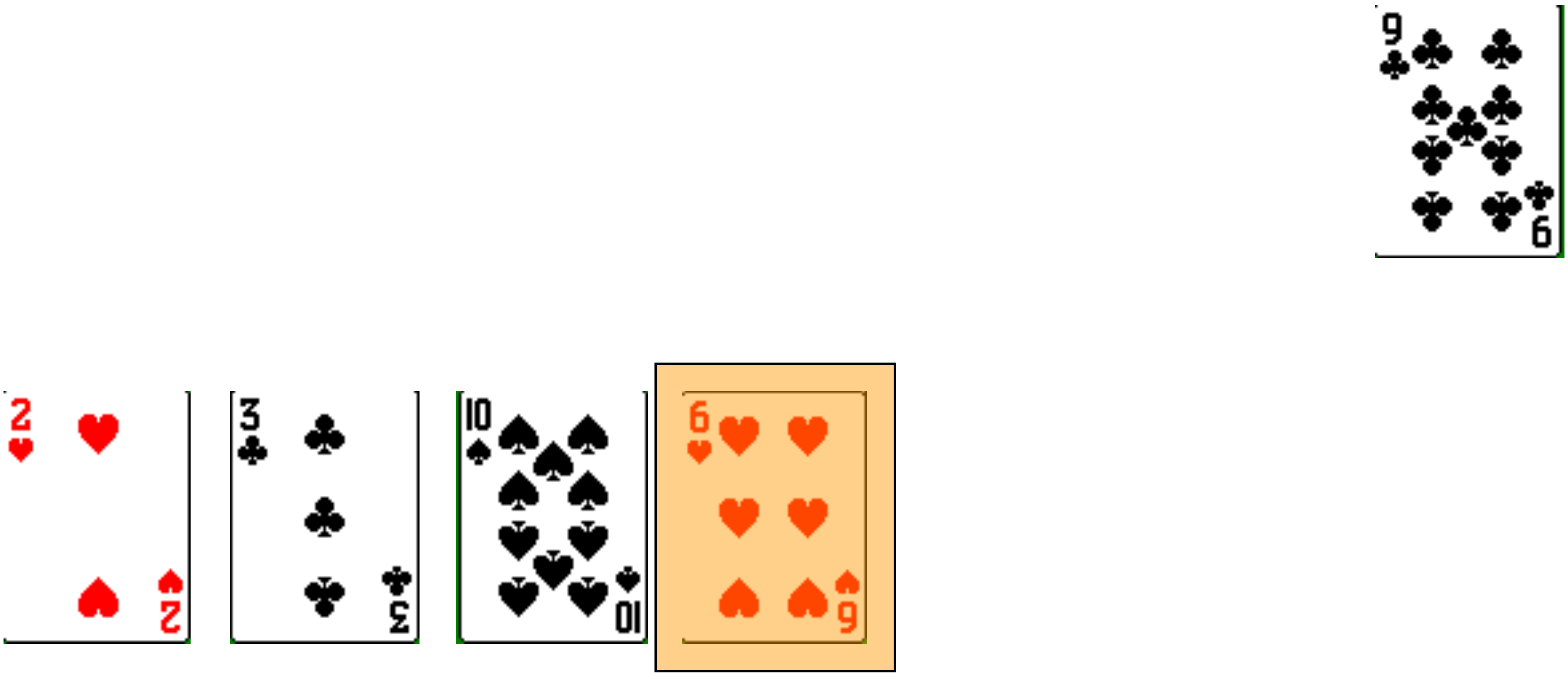
Insertion sort





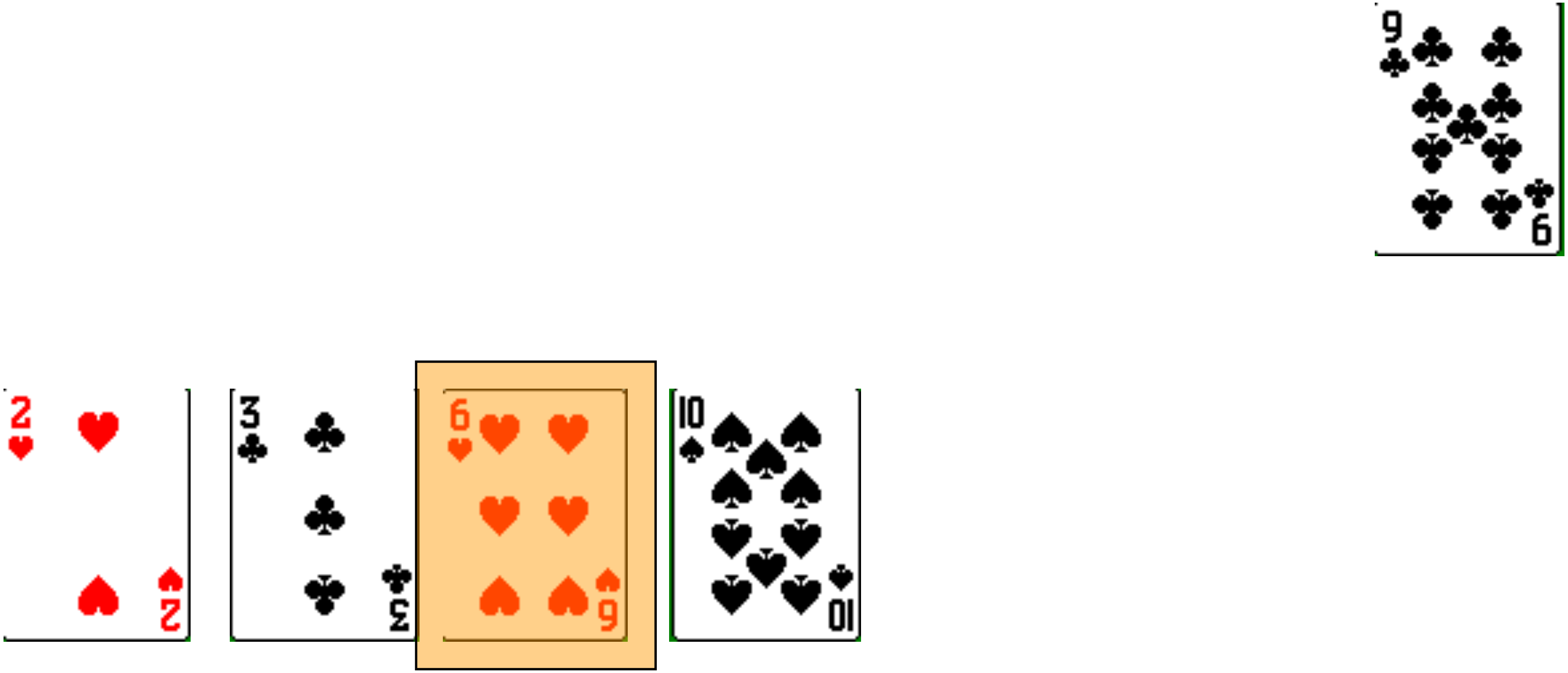
Algoritmos en la computación

Insertion sort



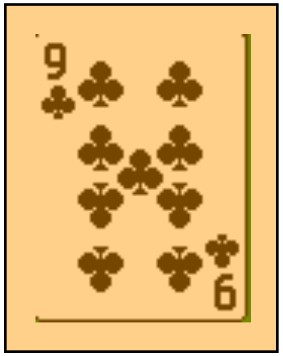
Algoritmos en la computación

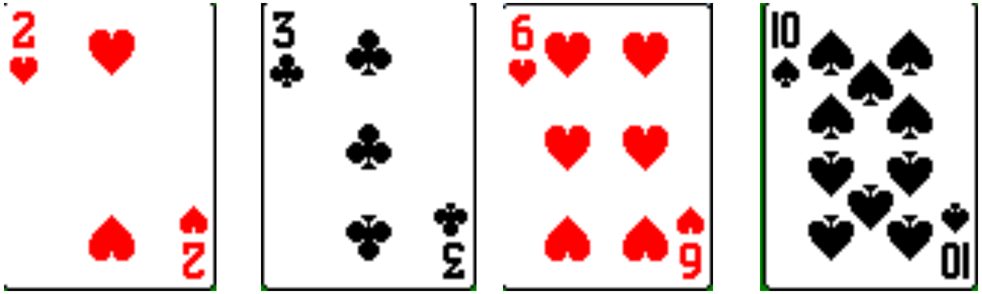
Insertion sort



Algoritmos en la computación

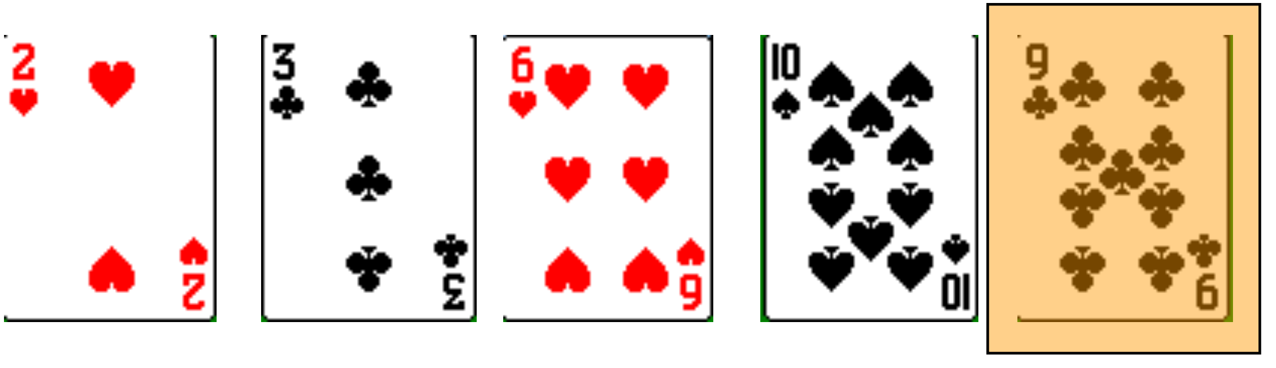
Insertion sort





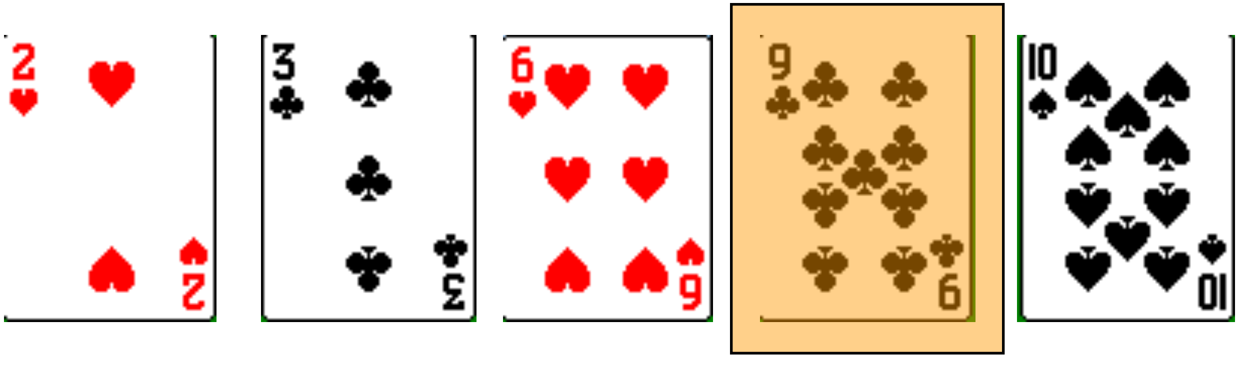
Algoritmos en la computación

Insertion sort



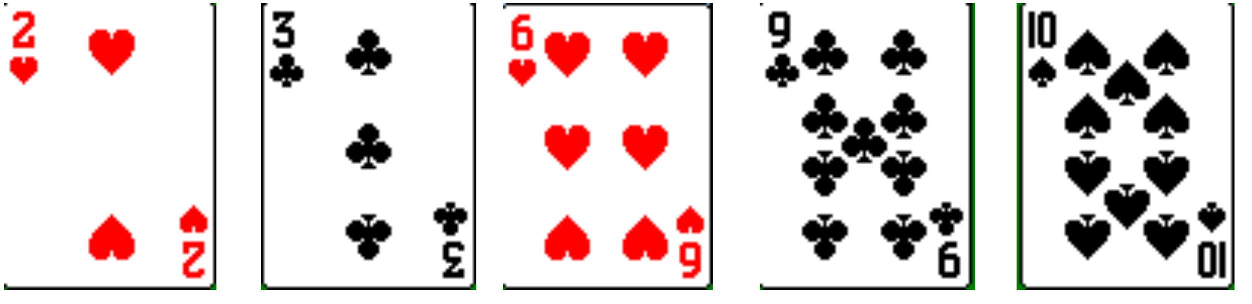
Algoritmos en la computación

Insertion sort



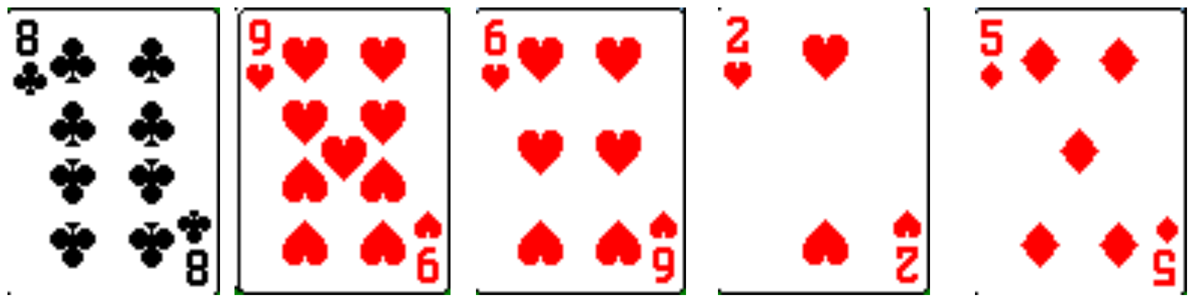
Algoritmos en la computación

Insertion sort



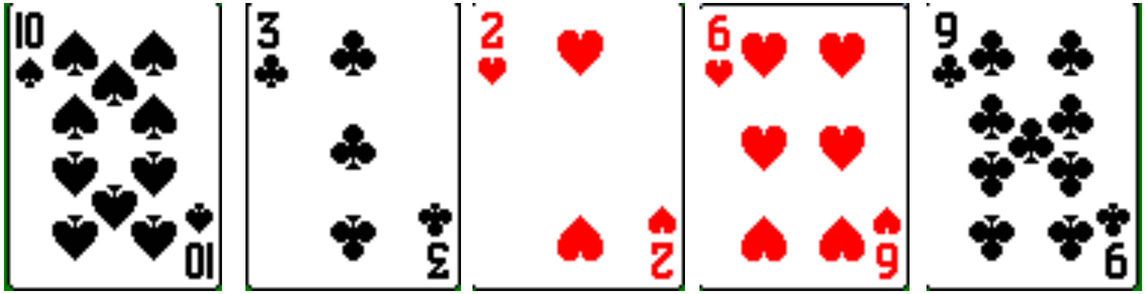
Algoritmos en la computación

Insertion sort



Algoritmos en la computación

Insertion sort

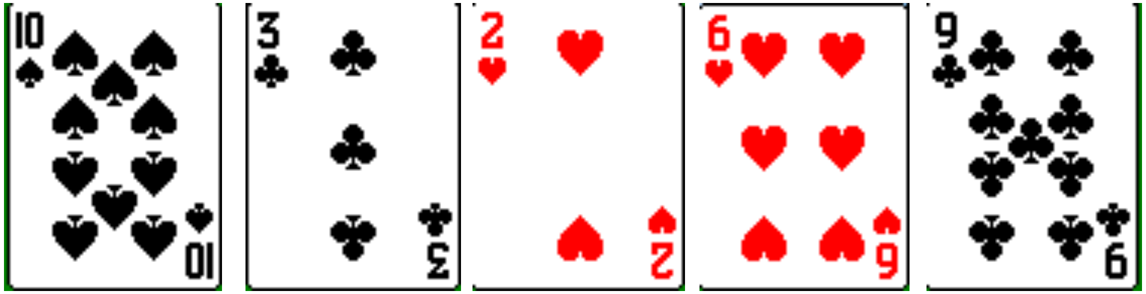


A:

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

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	

{ 4, 6, 1, 8, 2, 7 }

j = 2

key = 6

i = 1

A[i] = 4

A[i] > key   4 > 6 X

No entra el while

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

{ 4, 6, 1, 8, 2, 7 }

{4, 6, 1, 8, 2, 7}

```
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 = 3   i = 2

key = 1

i = 2

6 > 1

{4, 6, 6, 8, 2, 7}

i = 1 y 4 > 1 → {1, 4, 6, 8, 2, 7}

{4, 4, 6, 8, 2, 7}

i = 0

$\{1, 4, 6, 8, 2, 7\}$

$j = 4$

$key = 8$

$i = 3$ $6 > 8$

```
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
```

$\{1, 4, 6, 8, 2, 7\}$

$\{1, 4, 6, 8, 2, 7\}$

```
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
```

$j = 5$

$key = 2$

$i = 4$

$i > 0$ y $8 > 2$

$\{1, 4, 6, 8, 8, 7\}$

$i = 3 > 0$ y $6 > 2$

$\{1, 4, 6, 6, 8, 7\}$

$i = 2$, $4 > 2$

$\{1, 4, 4, 6, 8, 7\}$

$i = 1$ $1 > 2$ ✗

$\{1, 2, 4, 6, 8, 7\}$

{1, 2, 4, 6, 8, 7}

j=6 i=5

key=7

i=5>0 y 8>7 ✓

{1, 2, 4, 6, 8, 8}

i=4>0 y 6>7 ✗

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

{1, 2, 4, 6, 7, 8}

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	

Tarda el computador

Algoritmos en la computación

for j$\leftarrow$1 to 3 $\longrightarrow$ for (int j=1; j<=3; j++)

j=1 √

j=2 √

j=3 √

j=4 ×

Algoritmos en la computación

for j ← 1 to 3 $\longrightarrow$ for (int j=1; j<=3; j++)

j=1 √

j=2 √

j=3 √

j=4 ×

Comparación de salida

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?

$\left.\begin{array}{l} 1 \\ 2 \\ \vdots \\ n \end{array}\right\}$ n validos

$n+1$ } salida

$n+1$

Comparaciones

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

Lo de adentro del for (iterador)
se hace las veces que ENTRA

Los números válidos

Algoritmos en la computación

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

$$\underbrace{2 \quad 3 \quad 4 \; . \; . \; . \; . \; n}_{\text{Validas}} \qquad \underbrace{n+1}_{\text{Salida}}$$

n - 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 ×

$$\underbrace{4-2+1}_{\text{válidos}} \quad \underbrace{+1}_{\text{salida}}$$

4

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

Comparación de salida

Algoritmos en la computación

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

2
3
4
5
6
7

???

$$\underbrace{6 - 2 + 1}_{\text{saltos}} + \underbrace{1}_{\text{salida}}$$

6

Algoritmos en la computación

for j←2 to n

La cantidad de comparaciones en el for es:

$$\underbrace{n-2+1}_{\text{valido}} + \underbrace{1}_{\text{salida}} \rightarrow n$$

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

n

	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
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

$$\sum_{j=2}^{n} t_j$$

T(n)=**???**

Complejidad computacional

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] > 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

2	3	10	16	19

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	10	16	19

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

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

T(n)=???

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:

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

Entonces operando tenemos:

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

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 la condición SIEMPRE se cumple

j $\underbrace{j-1, j-2, \ldots, 1}_{\text{validos}}, 0$

En el peor de los casos, cuánto vale t_j?

j - 1 (número validos)
1 (salida)
Total comparaciones j

Algoritmos en la computación

3	2	1	6	9

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

3	2	1	6	9

i=1

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

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	$\sum_{j=2}^{n} j$
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)=???

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:

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

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

Entonces operando tenemos:

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

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

Dividimos la sumatoria Aprovechamos el anterior caso

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

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 computo 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)	c_2	$n+1$
3 j←1	c_3	n
4 while j<=len(mat2)	c_4	$n(m+1)$
5 mat3[i][j]←mat1[i][j]+mat2[i][j]	c_5	nm
6 j←j+1	c_6	nm
7 i←i+1	c_7	n

n

m

$O(nm)$

for(i = 1, i<=n, i++)

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}

1

1

$n+1$

n

n

t_i

$t_i - 1$

$t_i - 1$

n

n

$n=1+1+1$

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

$t_i = i+1$

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

$\underbrace{1, 2, \ldots, i, i+1}_{\text{valores}}$

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

$$\left.\frac{n(n+1)}{2} + n \right\} \quad 6$$

$$\frac{n(n+1)}{2} \quad \begin{matrix} 7 \\ 8 \end{matrix}$$

Investigar la forma cerrada de las sumatorias

Y cómo operar sumatorias.

Algoritmos en la computación

Calcule el tiempo de computo para el algoritmo

def programa3(n)

$\underbrace{i = 1,2,3, \ldots\ n}_{\text{validos}}, n+1$

k = i, i+1, ... , n, n+1

$t_1 = i = 1$

k : 1, 2, 3, n, n+1 + n+1

$t_2 = i = 2$

k = 2, 3, 4, n, n+1 n

t_3 n-1

$t_4 = n-2$

	Instrucción	Costo
1	1 i←1	c_1
n+1	2 while i<=n	c_2
n	3 k←i	c_3
t_i	4 $\boxed{\text{while k<=n}}$	c_4
$t_i - 1$	5 k← k+1	c_5
n	6 k←1	c_6
k_i	7 while k<=i	c_7
$k_i - 1$	8 k←k+1	c_8
n	9 i←i+1	c_9

i=1 i=2 i=3 i=4

$(n+1) + (n) + (n-1) + (n-2) + \ldots$

$$\sum_{i=1}^{n} (n-i+2) \Rightarrow (n+1)+n+\cdots$$

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

5)

$i=1$

$n + n-1 + n-2 \cdots$

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

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

$i = 1 \ldots\ldots n$

$k = 1$

while $k <= i$

$k = k+1$

$i = 1$

$k = \underbrace{1, 2}_{2}$

$i = 2$

$k: \underbrace{1, 2, 3}_{3}$

$i = 3$

$k: \underbrace{1, 2, 3, 4}_{4}$

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

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

Algoritmos en la computación

Calcule el tiempo de computo para el algoritmo

def programa4(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+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

1

$\lceil n/2 \rceil + 1$

$\lceil \frac{n}{2} \rceil$

1, 3, 5, 7, ..., n

par

impar

1, 3, 5, 7, ..., n-1 n+1

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

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

1, 3, 5, 7, n, n+2

4 6

$i = 1, 3, 5, \ldots$

ti
ti -1

```
k <-- i
while(k <= n)
    k = k + 2
```

$t_1 = 1, 3, 5, \ldots n, n-1 \quad \lceil n/2 \rceil + 1 - 0$

$t_3 = 3, 5, 7, \ldots n, n-1 \quad \lceil n/2 \rceil - 1 + 1$

$t_5 = 5, 7, 9, \ldots n / n-1 \quad \lceil n/2 \rceil - 2 + 1$

$\left\lfloor \frac{i}{2} \right\rfloor$

$k : 1, 2, 3, 4$

$i : 1, 3, 5$

$2i - 1 \to k$

$i = \frac{k+1}{2}$

$\lceil n/2 \rceil + 1 + \lceil n/2 \rceil - 1 + 2$

$$\sum_{k=1}^{n} n/2 + 1 + \left\lfloor \frac{k+1}{4} \right\rfloor = \frac{n^2}{2} + n + \frac{1}{4} n + \left\lfloor \frac{n(n+1)}{8} \right\rfloor$$

Algoritmos en la computación

Calcule el tiempo de computo para el algoritmo

def programa5(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+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

1

$\underbrace{1, 2, 4, 8, 16 \ldots n}_{\text{Valdos}}$

$\underbrace{\lfloor \log_2(n) \rfloor + 1 + 1}$

$\lfloor \log_2(n) \rfloor + 2$

i = i 2

i = 2

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 Ultima 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	$(50*10^8 \lg 10^{8)}/10^{11}=0,4$segs

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