Análisis del Algoritmo de Ordenamiento de inserción

José Ortiz Bejar

Facultad de Ingeniería Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo

Febrero 23, 2022

Contenido

- **Entrada:** una secuencia $A = \langle a_1, a_2, \ldots, a_n \rangle$

- **Entrada:** una secuencia $A = \langle a_1, a_2, \ldots, a_n \rangle$ **Salida:** una secuencia $\langle b_1, b_2, \ldots, b_n \rangle$ tal que
  - $\langle b_1, b_2, \ldots, b_n \rangle$ es una *permutación* de $\langle a_1, a_2, \ldots, a_n \rangle$

- **Entrada:** una secuencia $A = \langle a_1, a_2, \ldots, a_n \rangle$ **Salida:** una secuencia $\langle b_1, b_2, \ldots, b_n \rangle$ tal que
  - $\langle b_1, b_2, \ldots, b_n \rangle$ es una *permutación* de $\langle a_1, a_2, \ldots, a_n \rangle$
  - $\langle b_1, b_2, \ldots, b_n \rangle$ está *ordenada*

$$b_1 \leq b_2 \leq \cdots \leq b_n$$

- **Entrada:** una secuencia $A = \langle a_1, a_2, \ldots, a_n \rangle$ **Salida:** una secuencia $\langle b_1, b_2, \ldots, b_n \rangle$ tal que
  - $\langle b_1, b_2, \ldots, b_n \rangle$ es una *permutación* de $\langle a_1, a_2, \ldots, a_n \rangle$
  - $\langle b_1, b_2, \ldots, b_n \rangle$ está *ordenada*

$$b_1 \leq b_2 \leq \cdots \leq b_n$$

- Por lo general, A es implementada como un arreglo

- **Entrada:** una secuencia $A = \langle a_1, a_2, \ldots, a_n \rangle$ **Salida:** una secuencia $\langle b_1, b_2, \ldots, b_n \rangle$ tal que
  - $\langle b_1, b_2, \ldots, b_n \rangle$ es una *permutación* de $\langle a_1, a_2, \ldots, a_n \rangle$
  - $\langle b_1, b_2, \ldots, b_n \rangle$ está *ordenada*

$$b_1 \leq b_2 \leq \cdots \leq b_n$$

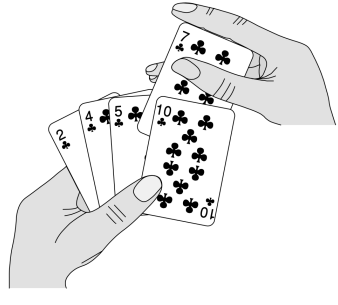
- Por lo general, A es implementada como un arreglo

$A =$

6	8	3	2	7	6	11	5	9	4

Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas



Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas

Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

Ordenamiento de Inserción

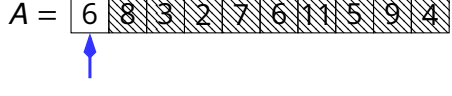
- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

$A =$

6	8	3	2	7	6	11	5	9	4

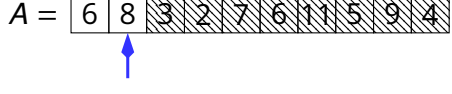
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

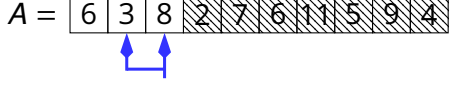


Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

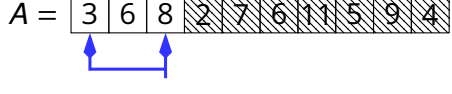
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



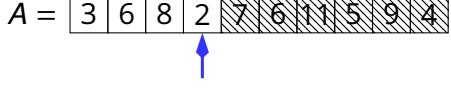
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



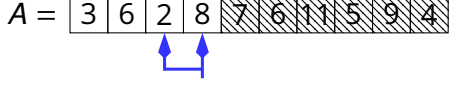
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



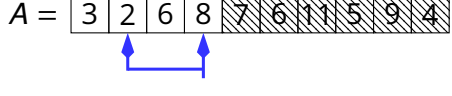
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



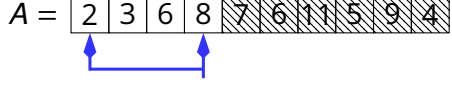
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



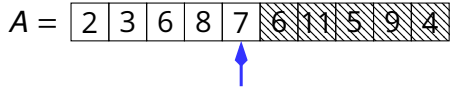
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



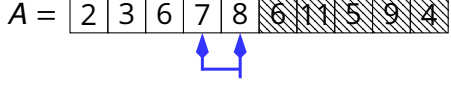
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



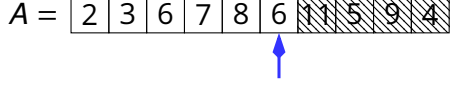
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



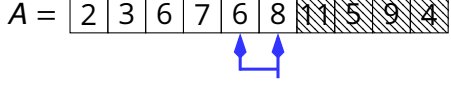
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



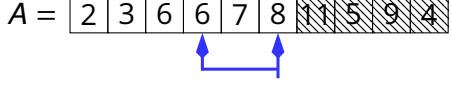
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



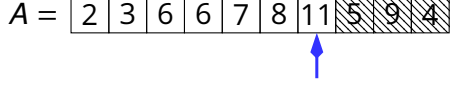
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



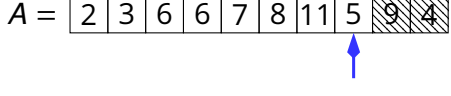
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



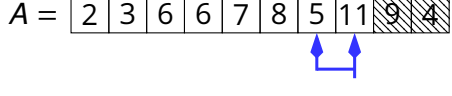
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



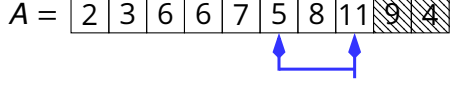
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



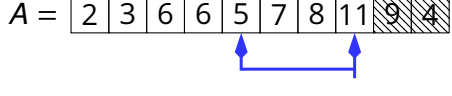
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



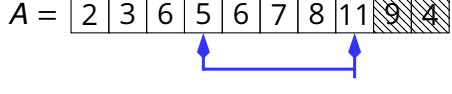
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



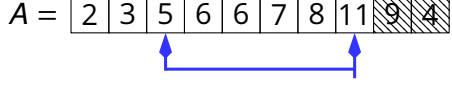
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



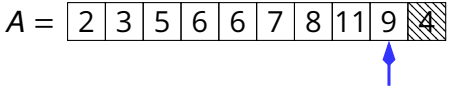
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



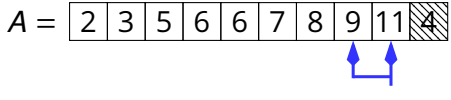
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

$A =$

2	3	5	6	6	7	8	9	11	4

Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

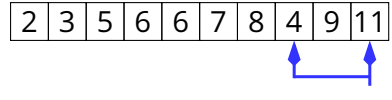
$A =$

2	3	5	6	6	7	8	9	4	11

Ordenamiento de Inserción

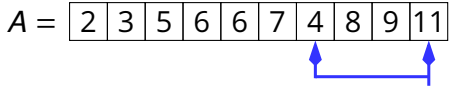
- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

$A =$



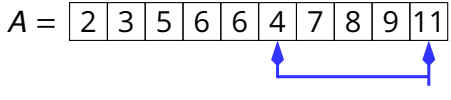
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



Ordenamiento de Inserción

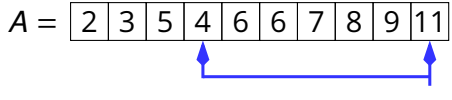
- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada

$A =$

2	3	5	6	4	6	7	8	9	11

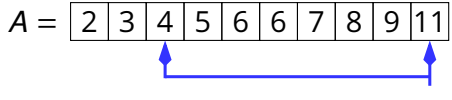
Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



Ordenamiento de Inserción

- **Idea:** igual como ordenamos una mano de cartas
  - procesar la secuencia de izquierda a derecha
  - tomar el valor en la posición actual a_j
  - insertarlo en la posición que le corresponde dentro de la secuencia $\langle a_1, a_2, \ldots a_{j-1} \rangle$ de tal forma que la subsequencia $\langle a_1, a_2, \ldots a_j \rangle$ se mantenga ordenada



Ordenamiento de Inserción (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

Ordenamiento de Inserción (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- ¿El procedimiento **Insertion-Sort** es *correcto*?
- ¿Cuál es la complejidad **Insertion-Sort**?
- ¿Podemos hacerlo mejor?

Complejidad de Insertion-Sort

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

Complejidad de Insertion-Sort

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- Las lineas del ciclo mas externo (lineas 1–2) se ejecutan exactamente $n - 1$ veces (con $n = \mathit{length}(A)$)
- ¿Qué ocurre el ciclo interno (lineas 3–5)?
  - ¿Cuál es la complejidad del mejor, peor y caso promedio?

Complejidad de Insertion-Sort (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Mejor caso:**

Complejidad de Insertion-Sort (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Mejor caso:** el ciclo interno *nunca* se ejecuta
  - ¿Qué caso es este último?

Complejidad de Insertion-Sort (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Mejor caso:** el ciclo interno *nunca* se ejecuta
  - ¿Qué caso es este último?
- **Peor caso:**

Complejidad de Insertion-Sort (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Mejor caso:** el ciclo interno *nunca* se ejecuta
  - ¿Qué caso es este último?
- **Peor caso:** el ciclo interno se ejecuta exactamente $j - 1$ veces para cada iteración del ciclo externo.
  - ¿Qué caso es este?

Complejidad de Insertion-Sort (3)

- La complejidad del peor caso es cuando el ciclo interno se ejecuta exactamente $j - 1$ veces, entonces

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

Complejidad de Insertion-Sort (3)

- La complejidad del peor caso es cuando el ciclo interno se ejecuta exactamente $j - 1$ veces, entonces

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

$T(n)$ es la *serie aritmética* $\sum_{k=1}^{n-1} k$, entonces

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

$$T(n) = \Theta(n^2)$$

Complejidad de Insertion-Sort (3)

- La complejidad del peor caso es cuando el ciclo interno se ejecuta exactamente $j - 1$ veces, entonces

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

$T(n)$ es la *serie aritmética* $\sum_{k=1}^{n-1} k$, entonces

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

$$T(n) = \Theta(n^2)$$

- El mejor caso es $T(n) = \Theta(n)$

Complejidad de Insertion-Sort (3)

- La complejidad del peor caso es cuando el ciclo interno se ejecuta exactamente $j - 1$ veces, entonces

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

$T(n)$ es la *serie aritmética* $\sum_{k=1}^{n-1} k$, entonces

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

$$T(n) = \Theta(n^2)$$

- El mejor caso es $T(n) = \Theta(n)$
- El caso promedio es $T(n) = \Theta(n^2)$

Corrección

Corrección

- ¿El metódo **Insertion-Sort** termina para todas las entradas válidas?

Corrección

- ¿El metódo **Insertion-Sort** termina para todas las entradas válidas?
- Si es así, ¿Satisface las condiciones del problema de ordenamiento?
  - A contiene una *permutación* de los valores iniciales de A
  - A está *ordenada:* $A[1] \leq A[2] \leq \cdots \leq A[length(A)]$

Corrección

- ¿El metódo **Insertion-Sort** termina para todas las entradas válidas?
- Si es así, ¿Satisface las condiciones del problema de ordenamiento?
  - A contiene una *permutación* de los valores iniciales de A
  - A está *ordenada:* $A[1] \leq A[2] \leq \cdots \leq A[length(A)]$
- Deseamos tener ***una prueba formal de la corrección***
  - no parece ser algo trivial...

La lógica en los pasos algorítmicos

Ejemplo 1: (programa simple)

Bigger(n)

```
1  // debe regresar un valor mayor que n
2  m = n * n + 1
3  return m
```

Ejemplo 1: (programa simple)

```
Bigger(n)
    1  // debe regresar un valor mayor que n
    2  m = n * n + 1
    3  return m
```

Ejemplo 2: (branching)

```
SortTwo(A)
    1  // debe ordenar (en-sitio) un arreglo de 2 elementos
    2  if A[1] > A[2]
    3       t = A[1]
    4       A[1] = A[2]
    5       A[2] = t
```

Invariantes de ciclo

Invariantes de ciclo

- Formulamos una condición C para el *invariante de ciclo*
  - C debe ser verdadera *durante* cada iteración

Invariantes de ciclo

- Formulamos una condición *C* para el *invariante de ciclo*
  - *C* debe ser verdadera *durante* cada iteración
  - *C* debe ser relevante para la definición del problema : utilizamos *C* al final de de la iteración para probar la corrección del resultado

Invariantes de ciclo

- Formulamos una condición *C* para el *invariante de ciclo*
  - *C* debe ser verdadera *durante* cada iteración
  - *C* debe ser relevante para la definición del problema : utilizamos *C* al final de de la iteración para probar la corrección del resultado
- Entonces, ya solo es necesario probar que el algoritmo termina

Invariantes de ciclo (2)

Invariantes de ciclo (2)

- Formulación: este es el paso que requiere de mayor ingenio
  - *el invariante debe reflejar la estructura del algoritmo*
  - debe ser la base para probar la corrección de la solución

Invariantes de ciclo (2)

- Formulación: este es el paso que requiere de mayor ingenio
  - *el invariante debe reflejar la estructura del algoritmo*
  - debe ser la base para probar la corrección de la solución

- Prueda de validez (e.i., que *C* sea de hecho una invariante de ciclo): usalmente una *prueba por inducción*
  - ***inicialización:*** debemos probar que
    el invariante C es verdadero antes de entrar al ciclo
  - ***mantenimiento:*** debemos probar que
    Si *C es cierto al inicio de cada iteración* ***entonces*** *se mantiene verdadero después de cada iteración*

Loop Invariant for Insertion-Sort

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

Loop Invariant for Insertion-Sort

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- La idea principal es insertar $A[i]$ en $A[1 \,..\, i-1]$ de tal forma que se mantenga una *sub-secuencia ordenada* $A[1 \,..\, i]$

Loop Invariant for Insertion-Sort

```
Insertion-Sort(A)
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- La idea principal es insertar $A[i]$ en $A[1 \,..\, i-1]$ de tal forma que se mantenga una *sub-secuencia ordenada* $A[1 \,..\, i]$

- ***Invariant:*** (ciclo externo) *el sub-arreglo* $A[1 \,..\, i-1]$ *contiene los elementos originales en* $A[1 \,..\, i-1]$ *pero ordenados*

Invariante de ciclo para Insertion-Sort (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

Invariante de ciclo para Insertion-Sort (2)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Inicialización:** $j = 2$, entonces $A[1\,..\,j-1]$ es un arreglo con un único elemento $A[1]$
  - $A[1]$ contiene el elemento original en $A[1]$
  - $A[1]$ está ordenado (trivial)

Invariante de ciclo para Insertion-Sort (3)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

Invariante de ciclo para Insertion-Sort (3)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Mantenimiento:** informalmente, si $A[1\,..\,i-1]$ es una permutación de $A[1\,..\,i-1]$ y $A[1\,..\,i-1]$ está ordenada (invariante), entonces *si* se entra en el ciclo interno :
  - se desplaza los elementos en $A[k\,..\,i-1]$ hacia la derecha de uno en uno
  - se inserta el valor *clave*, el cual estaba originalmente en la posición $A[i]$ en la posición que le corresponde dentro de la sub-secuencia en el rango $1 \leq k \leq i-1$

Invariante de ciclo para Insertion-Sort (4)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

Invariante de ciclo para Insertion-Sort (4)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Terminación: Insertion-Sort** termina con $i = \mathit{length}(A) + 1$; el invariante define que

Invariante de ciclo para **Insertion-Sort** (4)

Insertion-Sort(A)

```
1  for i = 2 to length(A)
2      j = i
3      while j > 1 and A[j − 1] > A[j]
4          swap A[j] and A[j − 1]
5          j = j − 1
```

- **Terminación: Insertion-Sort** termina con $i = \mathit{length}(A) + 1$; el invariante define que
  - $A[1\,..\,i-1]$ es una permutación de la secuencia $A[1\,...\,i-1]$ original
  - $A[1\,..\,i-1]$ está ordenada

  Dada la condición de terminación, $A[1\,..\,i-1]$ es el arreglo A completo

  Entonces **Insertion-Sort** es *correcto*

Sumario

- Dado un problema P y un algoritmo $\mathbb{A}$
  - P define formalmente una condición *correción*
  - por simplicidad se asume que, $\mathbb{A}$ consiste de un único ciclo

Sumario

- Dado un problema P y un algoritmo $\mathbb{A}$
  - P define formalmente una condición *correción*
  - por simplicidad se asume que, $\mathbb{A}$ consiste de un único ciclo

---

1. Formular un invariante C

- Dado un problema P y un algoritmo $\mathbb{A}$
  - P define formalmente una condición *correción*
  - por simplicidad se asume que, $\mathbb{A}$ consiste de un único ciclo

---

1. Formular un invariante C
2. **Inicialización** (para todas la entradas validas)
   - probar que C se mantiene antes de la ejecución de la primera instrucción de la primera iteración

Sumario

- Dado un problema P y un algoritmo $\mathbb{A}$
  - P define formalmente una condición *correción*
  - por simplicidad se asume que, $\mathbb{A}$ consiste de un único ciclo

---

1. Formular un invariante C
2. **Inicialización** (para todas la entradas validas)
   - probar que C se mantiene antes de la ejecución de la primera instrucción de la primera iteración
3. **Mantenimiento** (para todas las entradas validas)
   - probar que si C se mantiene antes de la primera instrucción de la primera iteración, entonces se mantiene al final de la iteración.

Sumario

- Dado un problema P y un algoritmo $\mathbb{A}$
  - P define formalmente una condición *correción*
  - por simplicidad se asume que, $\mathbb{A}$ consiste de un único ciclo

---

1. Formular un invariante C
2. **Inicialización** (para todas la entradas validas)
   - probar que C se mantiene antes de la ejecución de la primera instrucción de la primera iteración
3. **Mantenimiento** (para todas las entradas validas)
   - probar que si C se mantiene antes de la primera instrucción de la primera iteración, entonces se mantiene al final de la iteración.
4. **Terminación** (para todas las entradas validas)
   - probar que el ciclo termina, con alguna condición de salida X

Sumario

- Dado un problema P y un algoritmo $\mathbb{A}$
  - P define formalmente una condición *correción*
  - por simplicidad se asume que, $\mathbb{A}$ consiste de un único ciclo

---

1. Formular un invariante C
2. **Inicialización** (para todas la entradas validas)
   - probar que C se mantiene antes de la ejecución de la primera instrucción de la primera iteración
3. **Mantenimiento** (para todas las entradas validas)
   - probar que si C se mantiene antes de la primera instrucción de la primera iteración, entonces se mantiene al final de la iteración.
4. **Terminación** (para todas las entradas validas)
   - probar que el ciclo termina, con alguna condición de salida X
5. Probar que $X \wedge C \Longrightarrow P$, lo cual implica que $\mathbb{A}$ es correcto

Ejercicio: Analizar Selection-Sort

```
Selection-Sort(A)
1  n = length(A)
2  for i = 1 to n − 1
3      smallest = i
4      for j = i + 1 to n
5          if A[j] < A[smallest]
6              smallest = j
7      swap A[i] and A[smallest]
```

Ejercicio: Analizar Selection-Sort

Selection-Sort(A)

```
1  n = length(A)
2  for i = 1 to n − 1
3        smallest = i
4        for j = i + 1 to n
5              if A[j] < A[smallest]
6                    smallest = j
7        swap A[i] and A[smallest]
```

- ¿Es correcto?
  - ¿Cuál es el invariante del ciclo?
- ¿Cuál es su complejidad?
  - ¿Cuales son los casos promedio, peor y mejor?

Ejercicio: Analiza Bubblesort

Bubblesort(A)

```
1  for i = 1 to length(A)
2      for j = length(A) downto i + 1
3          if A[j] < A[j − 1]
4              swap A[j] and A[j − 1]
```

Ejercicio: Analiza Bubblesort

Bubblesort(A)

```
1  for i = 1 to length(A)
2      for j = length(A) downto i + 1
3          if A[j] < A[j − 1]
4              swap A[j] and A[j − 1]
```

- ¿Es correcto?
  - ¿Cuál es el invariante del ciclo?
- ¿Cuál es su complejidad?
  - ¿Cuales son los casos promedio, peor y mejor?