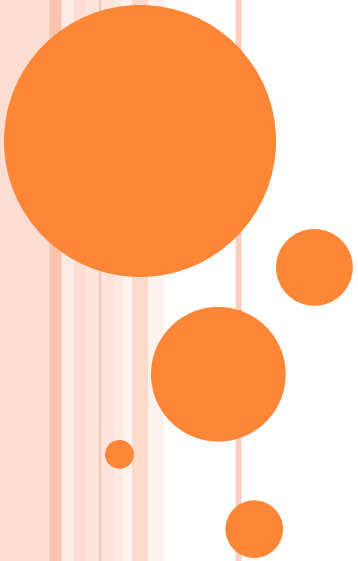
Análisis del algoritmo de ordenamiento por inserción

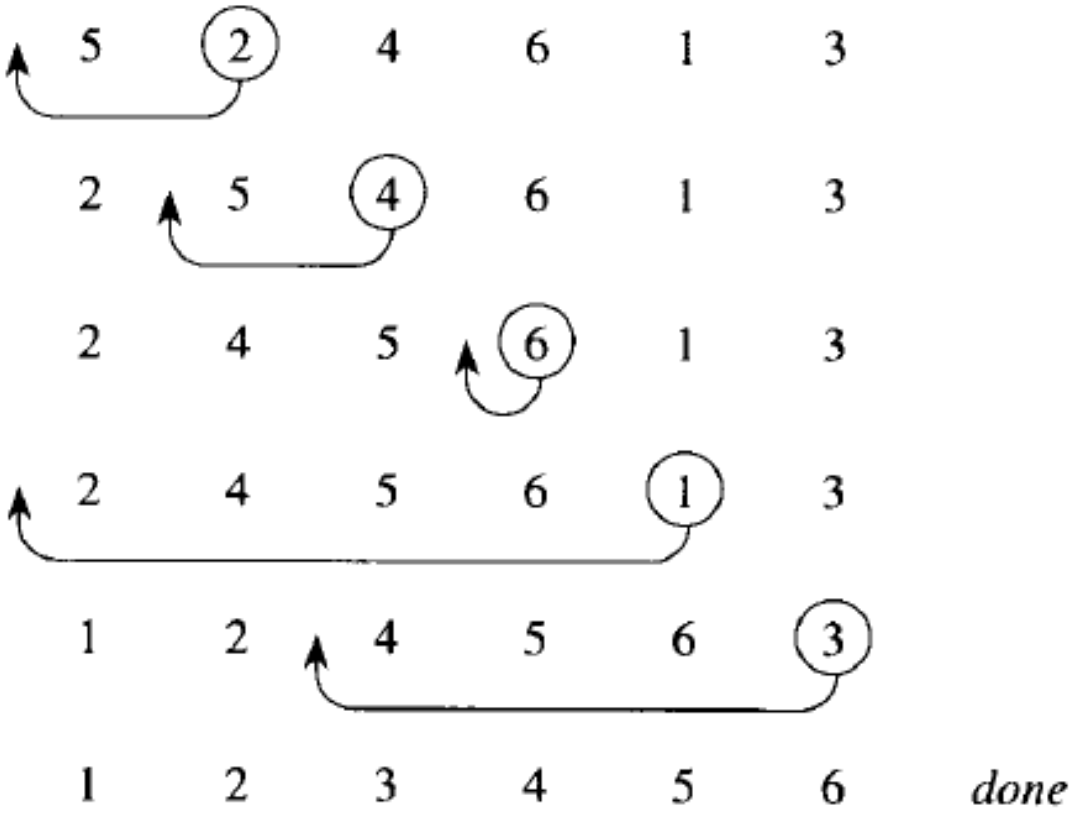


Ordenamiento por inserción

- Algoritmo eficiente para ordenar una pequeña cantidad de elementos.

- La ordenación por inserción funciona de la misma manera en que muchas personas ordenan una mano de cartas.

Ordenamiento por inserción



Ordenamiento por inserción

```
INSERTION-SORT(A)
1  for j ← 2 to length[A]
2       do key ← A[j]
3          ▷ Insert A[j] into the sorted sequence A[1 .. j − 1].
4          i ← j − 1
5          while i > 0 and A[i] > key
6               do A[i + 1] ← A[i]
7                  i ← i − 1
8          A[i + 1] ← key
```

Prueba de escritorio

- Una **prueba de escritorio** es un tipo de prueba algorítmica, que consiste en la validación y verificación del algoritmo a través de la ejecución de las sentencias que lo componen (proceso) para determinar sus resultados (salida) a partir de un conjunto determinado de elementos (entrada).

Análisis del algoritmo

- Analizar un algoritmo significa predecir los recursos que el algoritmo requiere.

- Generalmente, analizando varios algoritmos candidatos para un problema, el más eficiente puede ser fácilmente identificado. Dicho análisis puede indicar más de un candidato viable, pero varios algoritmos inferiores generalmente se descartan en el proceso.

Análisis del algoritmo

- Analizar incluso un algoritmo simple puede ser un desafío. Las herramientas matemáticas necesarias pueden incluir combinatoria discreta, teoría de probabilidad elemental, destreza algebraica y la capacidad de identificar los términos más significativos en una fórmula.

- Porque el comportamiento de un algoritmo puede ser diferente para cada entrada posible, necesitamos un medio para resumir ese comportamiento en fórmulas sencillas y fáciles de entender.

Análisis del algoritmo

- El tiempo que tarda el procedimiento de ordenamiento por inserción depende de la entrada:
  - Ordenar mil números lleva más tiempo que ordenar tres números.
  - Es más, INSERTION-SORT puede tomar diferentes cantidades de tiempo para ordenar dos secuencias de entrada del mismo tamaño dependiendo de qué tan ordenadas ya están.

- La mejor noción de tamaño de entrada depende del problema que se esté estudiando. Para muchos problemas, como clasificar o calcular transformadas discretas de Fourier, la medida más natural es la cantidad de elementos en la entrada, por ejemplo, el tamaño de matriz n para ordenar.

Análisis del algoritmo

- Para muchos otros problemas, como multiplicar dos enteros, la mejor medida del tamaño de entrada es el número total de bits necesario para representar la entrada en notación binaria ordinaria.

- Por ejemplo, si la entrada a un algoritmo es un grafo, el tamaño de entrada se puede describir por el número de vértices y aristas en el grafo.

Análisis del algoritmo

- El tiempo de ejecución de un algoritmo en una entrada particular es el número de operaciones primitivas o "pasos" ejecutados.

- Una cantidad constante de tiempo es necesaria para ejecutar cada línea de nuestro pseudocódigo.

- Una línea puede tomar una cantidad de tiempo diferente a otra línea, pero asumiremos que cada ejecución de la i-ésima línea toma el tiempo C_i, donde C_i es una constante.

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** *length*[A]	c_1	n
2 **do** $key \leftarrow A[j]$	c_2	$n-1$
3 ▷ Insert $A[j]$ into the sorted		
▷ sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

- El tiempo de ejecución del algoritmo es la suma de los tiempos de ejecución de cada declaración ejecutada.
- Una declaración que toma *Ci* pasos para ejecutarse y se ejecuta *n* veces contribuirán *Cin* al tiempo total de ejecución.
- Para calcular *T(n)*, el tiempo de ejecución del ordenamiento por inserción, sumamos los productos de los costos y los tiempos obteniendo:

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

- Incluso para entradas de un tamaño determinado, el tiempo de ejecución de un algoritmo puede depender de qué entrada es dada.

- Por ejemplo, el mejor de los casos ocurre si el arreglo ya está ordenado.

- Para cada $j=2,3,...,n$ encontramos que $A[i] \leq key$ en la línea 5 cuando i tiene su valor inicial de $j-1$.
- Por tanto, $t_j = 1$ para $j=2, 3,...,n$ y el mejor tiempo de ejecución es:

```
INSERTION-SORT(A)                                   cost    times
1  for j ← 2 to length[A]                           c1      n
2      do key ← A[j]                                c2      n − 1
3         ▷ Insert A[j] into the sorted
          ▷       sequence A[1 . . j − 1].         0       n − 1
4         i ← j − 1                                 c4      n − 1
5         while i > 0 and A[i] > key                c5      ∑_{j=2}^{n} t_j
6             do A[i + 1] ← A[i]                    c6      ∑_{j=2}^{n} (t_j − 1)
7                i ← i − 1                          c7      ∑_{j=2}^{n} (t_j − 1)
8         A[i + 1] ← key                            c8      n − 1
```

- **Mejor de los casos**

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \ . \end{aligned}$$

- Este tiempo de ejecución se puede expresar como :

  an + b para las *constantes a* y *b* que dependen de los costos *Ci*, por lo tanto, es una **función lineal** de *n*.

- Si el arreglo está en orden inverso, es decir, en orden decreciente, el resultados es el **peor de los casos.**
- Debemos comparar cada elemento *A[j]* con cada elemento en todo el subarreglo ordenado *A[1...j-1]*, por lo que $t_j = j$ para *j=2, 3, ... ,n.*
- Señalando que : $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$ y $\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$

Identidad en sumatorias $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Este tiempo de ejecución en el peor de los casos se puede expresar como:

$an^2 + bn + c$

para constantes a, b y c que dependen de los costos Ci.

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

Por lo tanto es una **función cuadrática de n**

Análisis del peor de los casos y del caso promedio

- Normalmente, nos concentraremos en el tiempo de ejecución del peor de los casos porque, es el tiempo de ejecución más grande para cualquier entrada de tamaño n.
  - El tiempo de ejecución del peor caso de un algoritmo es un límite superior en el tiempo de ejecución para cualquier entrada. Conocerlo nos da la garantía de que el algoritmo nunca tardará más.
  - Para algunos algoritmos, el peor de los casos ocurre con bastante frecuencia.
  - El "caso medio" suele ser tan malo como el peor de los casos.

Orden de crecimiento

- Primero, ignoramos el costo real de cada declaración, usando las constantes c_i para representar estos costos.
- En el peor de los casos, el tiempo de ejecución es:

$$an^2 + bn + c$$

- para algunas constantes a, b y c que dependen de los costos de cada sentencia c_i.
- Por lo tanto, ignoramos no solo los costos de cada sentencia, sino también los costos abstractos c_i.

- Es la tasa de crecimiento, u orden de crecimiento, del tiempo de ejecución lo que realmente nos interesa.
- Por lo tanto, consideramos solo el término principal de una fórmula (por ejemplo, an^2),dado que los términos de orden inferior son relativamente insignificantes para **n** grandes.

Orden de crecimiento

- También se ignora el coeficiente constante del término principal, ya que los factores constantes son menos significativo que la tasa de crecimiento para determinar la eficiencia computacional para grandes entradas.

- Por lo tanto, escribimos que la **ordenación por inserción**, tiene un tiempo de ejecución en el peor de los casos de $\boldsymbol{\Theta(n^2)}$ (pronunciado "theta de n-cuadrado").