Diseño de Algoritmos

Sesión 05

Profesores:

Tomás Lara Valdovinos – t.lara@uandresbello.edu

Jessica Meza-Jaque – je.meza@uandresbellu.edu

OBJETIVOS DE LA SESIÓN

- Conocer el ritmo de crecimiento de algoritmos a través de notaciones asintóticas
- Conocer las medidas de eficiencia
- Conocer algoritmos comunes y su comparación de eficiencia

CONTENIDOS DE LA SESIÓN

- Concepto de notación asintótica
- Medidas de eficiencia
- Algoritmos de ordenamiento burbuja e inserción

Ritmo de crecimiento de un algoritmo

- Se entiende por ritmo de crecimiento a la eficiencia variable de la ejecución de un algoritmo dependiendo de un factor de éste.
- Estamos interesados en la **cantidad de recursos** de cómputo que requiere un algoritmo para ser ejecutado.
- Un algoritmo debe ser, en primera instancia, **correcto**, esto quiere decir que produzca un resultado esperado en un tiempo finito.
- El ritmo de crecimiento es calculado bajo una plataforma que agrega un **factor constante** despreciable.

Notación asintótica

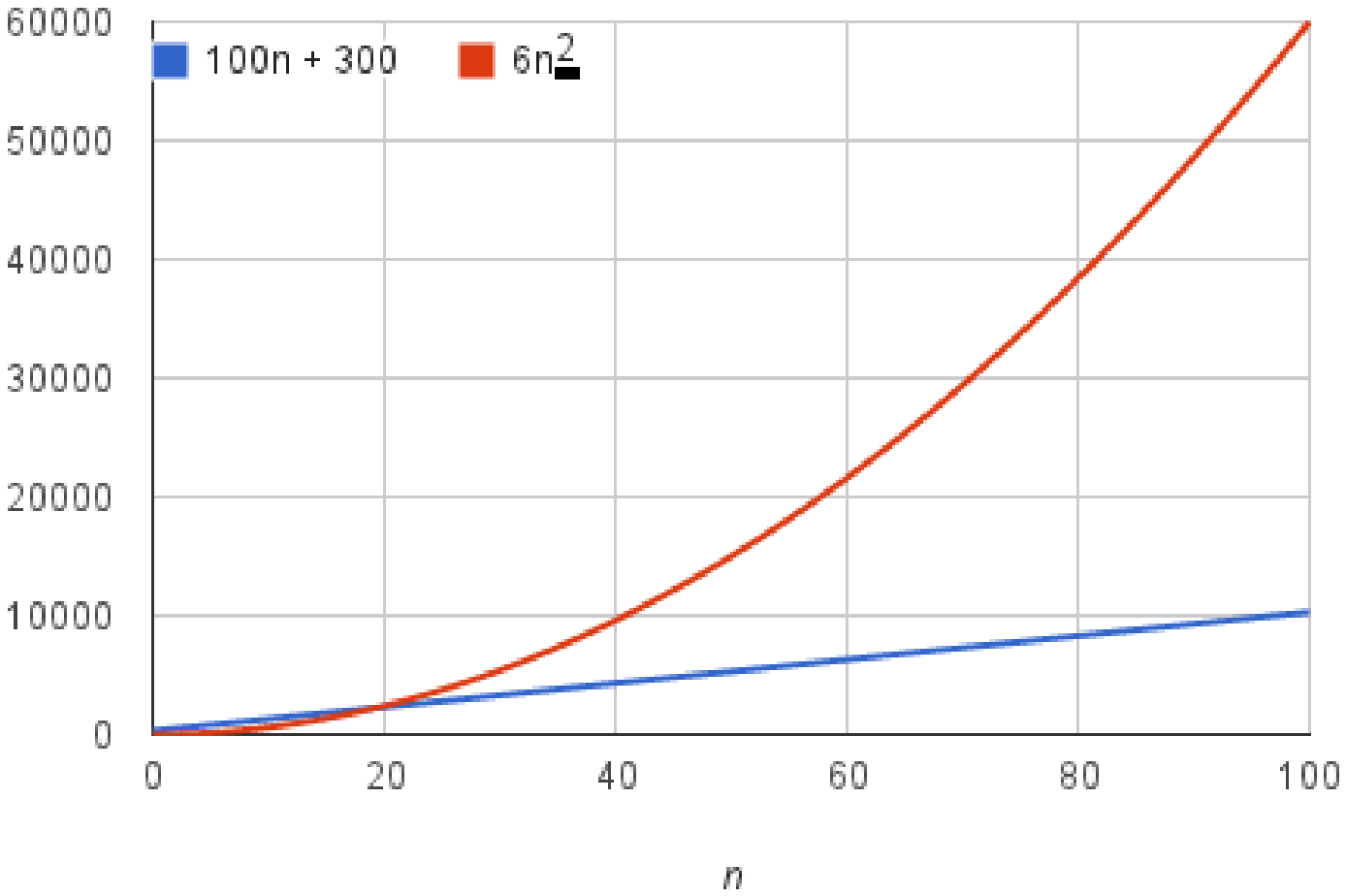
- Pensamos acerca del tiempo de ejecución del algoritmo como una función del **tamaño de la entrada.**
- **Ritmo de crecimiento** de la ejecución: Nos concentramos en qué tan rápido aumenta ésta en función del tamaño de entrada.
- Utilizamos los términos **más significativos** para determinar la tasa de crecimiento.

Ejemplo

- Tenemos: $6n^2+100n+300$
  Como la función que define el tiempo necesario para ejecutar un algoritmo **A** en función de su entrada de tamaño **n**.

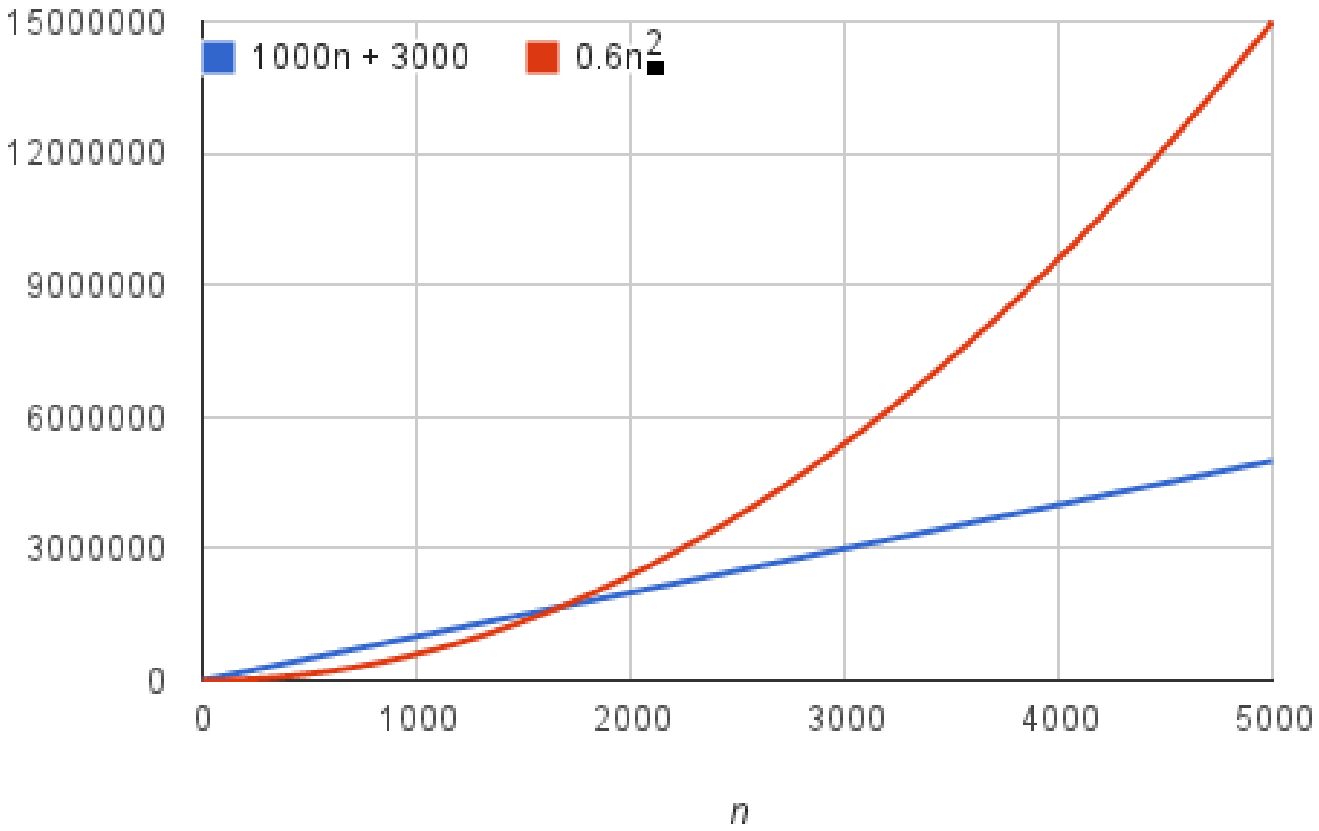
- El término $6n^2$ se vuelve más significativo que el resto de los términos, $100n+300$.

Gráficamente



- La tasa de crecimiento dada por 100n + 300 no es significativa en comparación a $6n^2$
- Si llevamos **n** a valores muy grandes los factores restantes pierden significado en el comportamiento de la curva
- Por lo tanto decimos que el algoritmo **A** es de orden $\boldsymbol{n^2}$

Otra comparación



- Si cambiamos los coeficientes, por ejemplo reducimos el de la potencia cuadrada y aumentamos los coeficientes lineales, tenemos el mismo comportamiento de crecimiento para **A.**
- Por lo tanto el orden es n^2

Resumen

- Al descartar los términos menos significativos y los coeficientes constantes, podemos enfocarnos en la parte importante del tiempo de ejecución de un algoritmo —su tasa de crecimiento— sin involucrarnos en detalles que complican nuestro entendimiento.
- Cuando descartamos los coeficientes constantes y los términos menos significativos, usamos **notación asintótica**.

Notación O grande (Big-O)

- Veamos una implementación sencilla de una búsqueda lineal:

```
var doLinearSearch = function(array) {
        for (var guess = 0; guess < array.length; guess++) {
                if (array[guess] === targetValue) {
                        return guess; // ¡lo encontraste!
                }
        }
        return -1; // no lo encontraste
};
```

Notación O grande (Big-O)

- Denotemos el **tamaño** del arreglo (array.length) con **n**.
- El número máximo de veces que el ciclo **for** puede ejecutarse es n.
- Este **peor caso** ocurre cuando el valor que se está buscando no está presente en el arreglo, o es el último elemento cuando sabemos que éste pertenece al arreglo.

Notación O grande (Big-O)

- Detonamos con O(f(x)) la función que define el mayor tiempo para ejecutar un algoritmo cuando n tiende al infinito.
- Para la búsqueda secuencial usamos:
  O(n), para referirnos al tiempo de ejecución de este algoritmo.

Popularidad de big-O

- Además de Big-O también tenemos:
  - Big-Ω : como el tiempo mínimo requerido para ejecutar el algoritmo (Cota inferior)
  - Big-Θ : Como el tiempo comprendido entre Big-O y Big- Ω

- Big-O es popularmente utilizado debido a que representa mejor el peor de los casos en la ejecución de un algoritmo.
- También es conocido como la cota superior de la notación Big-Θ.

Familias de rendimiento

- En orden decremental podemos definir la eficiencia de un algoritmo en los siguientes órdenes:
  - Constante
  - Logarítmica
  - Sub-linear
  - Linear
  - N log n
  - Cuadrática
  - Cúbica
  - Exponencial
  - Factorial

Análisis del mejor, promedio y peor de los casos.

- Peor Caso:
  - Define una instancia de entrada para la cual un algoritmo muestra el peor comportamiento de ejecución.
  - Podemos identificarlo como el input que previene que un algoritmo se ejecute de manera eficiente.
  - Ejemplo: En la búsqueda secuencial el peor de los casos es cuando el valor buscado se encuentra en el final de la lista (o no se encuentra en la lista).

Análisis del mejor, promedio y peor de los casos.

- Caso Promedio:
  - Define el comportamiento esperado cuando ejecutamos el algoritmo en instancia del problema aleatorios.
  - Informalmente, mientras algunos input podrían requerir mucho tiempo de ejecución por lo general la mayoría no lo requiere.
  - Esta medida describe la expectativa promedio que un usuario debería tener.
  - Ejemplo: En la búsqueda secuencial es más probable que el elemento buscado esté entremedio de la lista a que esté al final.

Análisis del mejor, promedio y peor de los casos.

- Mejor caso:
  - Define una clase de instancia de entrada para la cual un algoritmo muestra el mejor comportamiento de ejecución.
  - Para esos input, el algoritmo requiere muy poco trabajo para llegar a un resultado.
  - Ejemplo: En la búsqueda secuencial el mejor de los casos es que el elemento buscado esté al principio de la lista.

Análisis del mejor, promedio y peor de los casos.

- Consejos:
  - Por lo general, diseñamos algoritmos que estén adaptados para funcionar mejor en el peor de los casos y en el promedio, ya que el mejor de los casos raramente suele ocurrir.
  - Medir la eficiencia de un algoritmo basado en el caso promedio sugiere una visibilidad general de la ejecución.
  - Medir la eficiencia de un algoritmo basado en el peor de los casos nos indica los límites de la ejecución.

Medidas de eficiencia

- A la cantidad de **tiempo** que requiere la ejecución de un cierto algoritmo se le suele llamar **coste en tiempo.**
- mientras que a la cantidad de **memoria** que requiere se le suele llamar **coste en espacio**.

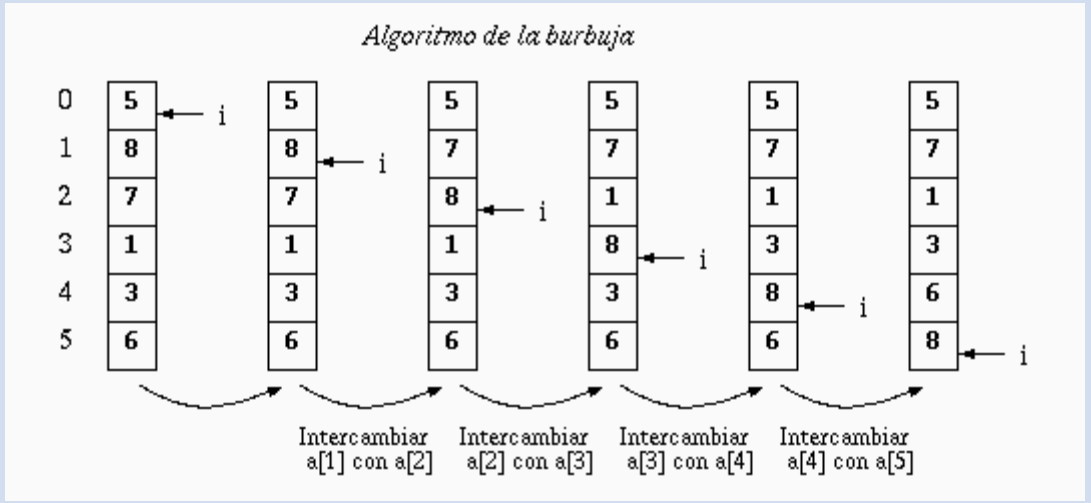
Comparación de eficiencia

- Una manera de comparar algoritmos que resuelven un **mismo problema** es utilizando las notaciones asintóticas en sus casos mejor, peor y promedio.
- Existen otros factores de comparación que dependen de las características propias del algoritmo. Es importante evaluar estas características dependiendo de la definición de nuestro problema.

Ordenamiento de burbuja

Mejor	Promedio	Peor
$O(n^2)$	$O(n^2)$	$O(n^2)$

```
Ordenar(A)
    For i = 1 to n do
        For j=0 to n-i do
            If A[j] > A[j+1] do
                Aux = A[j]
                A[j] = A[j+1]
                A[j+1] = Aux
end
```



Ordenamiento de burbuja

- Compara todos con todos.
- Es de fácil implementación.
- Debido a su alto orden de tiempo de ejecución no es recomendable para valores de **n** muy altos.

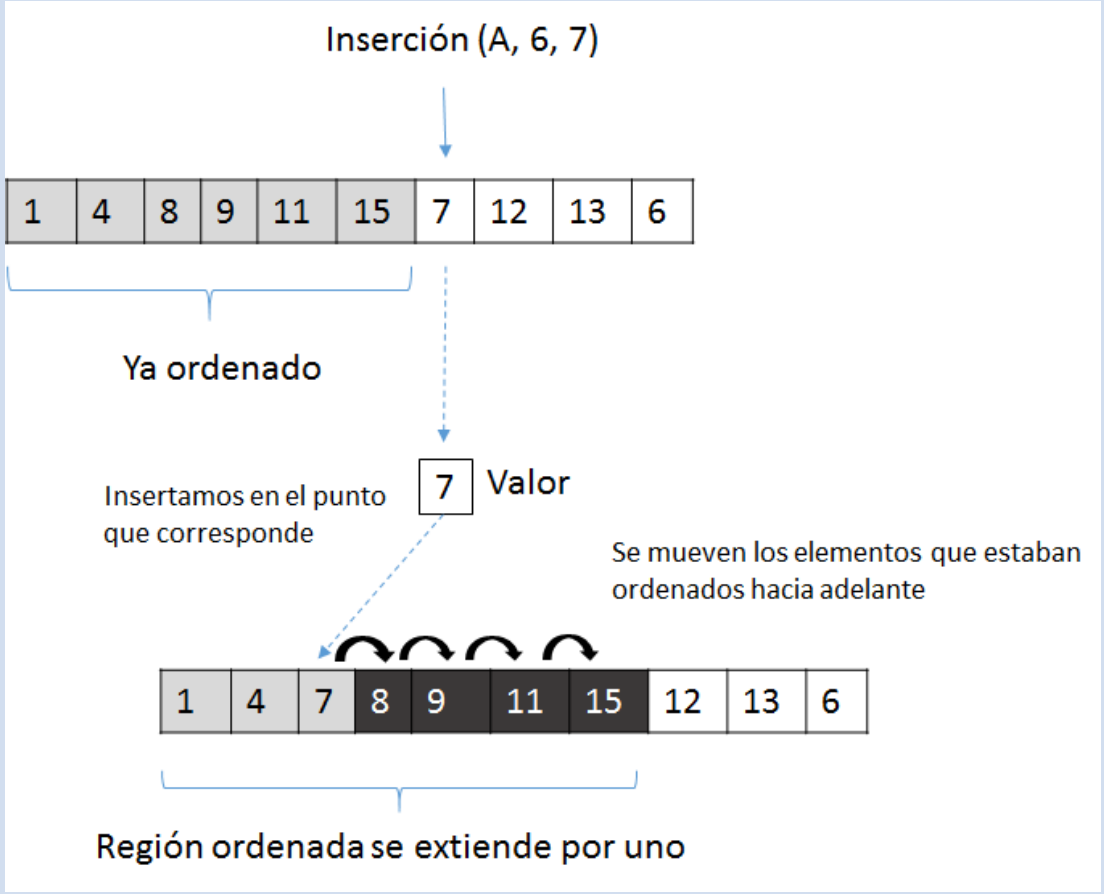
Ordenamiento por inserción

- Iterativo
- Listas

Mejor	Promedio	Peor
O(n)	$O(n^2)$	$O(n^2)$

```
Ordenar(A)
    For i = 1 to n-1 do
        Insercion(A, i, A[i])
End

Insercion(A, pos, Valor)
    i = pos – 1
    While i >= 0 and A[i] > Valor do
        A[i + 1] = A[i]
        i = i – 1
    A[i + 1] = Valor
End
```



Ordenamiento por inserción

- Fácil implementación
- Simula el ordenar una mano de cartas
- Funciona perfectamente sobre grupos de datos parcialmente ordenados.
- No es recomendable su uso en grupos de datos muy grandes.

CHECK - OBJETIVOS DE LA SESIÓN

- Conocer el ritmo de crecimiento de algoritmos a través de notaciones asintóticas
- Conocer las medidas de eficiencia
- Conocer algoritmos comunes y su comparación de eficiencia

CHECK

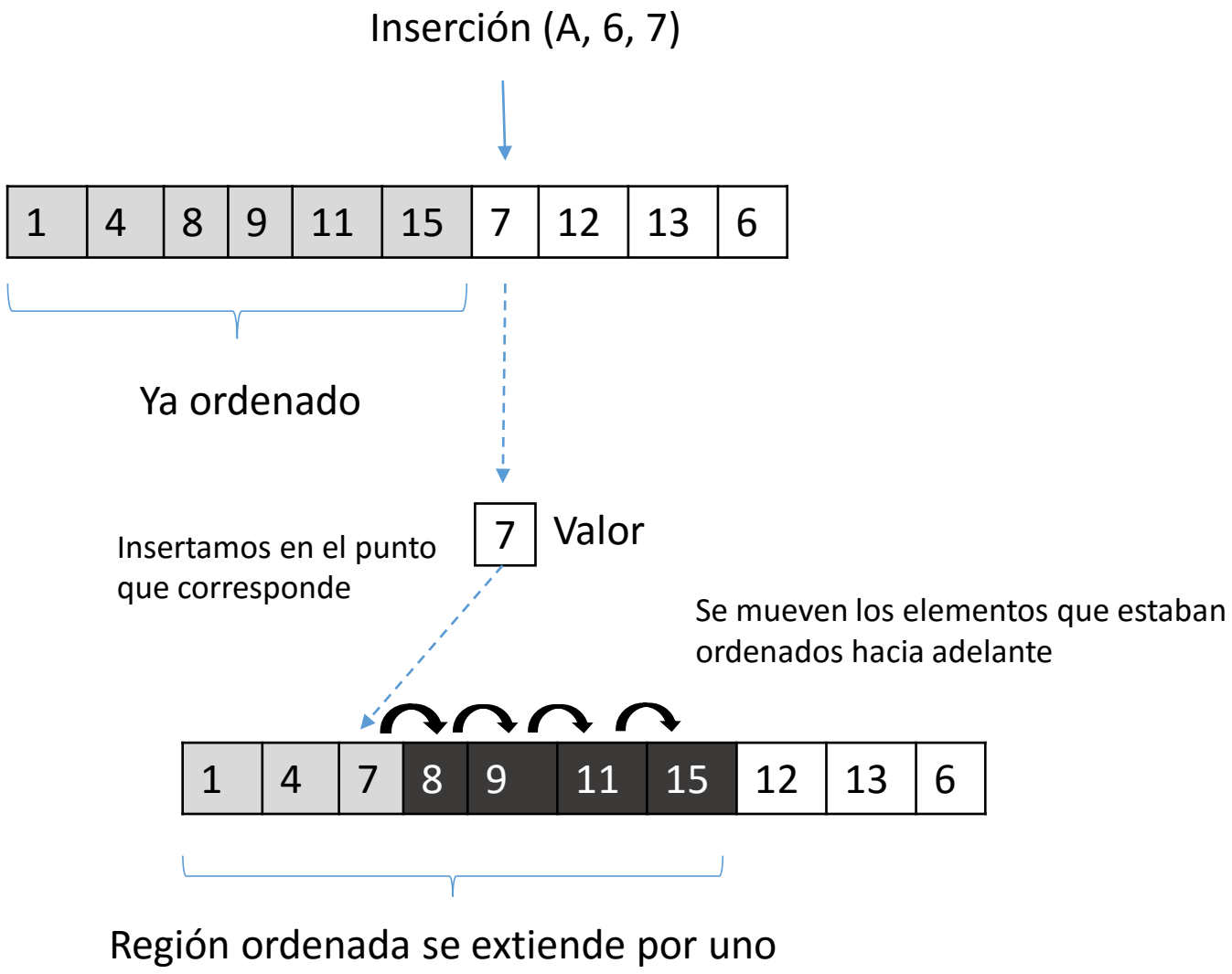
Diseño de Algoritmos

Sesión 05

Profesores:

Tomás Lara Valdovinos – t.lara@uandresbello.edu

Jessica Meza-Jaque – je.meza@uandresbellu.edu


Inserción (A, 6, 7)
1
4
8
9
11
15
7
12
13
6
Ya ordenado
7
Valor
Insertamos en el punto que corresponde
Se mueven los elementos que estaban ordenados hacia adelante
1
4
7
8
9
11
15
12
13
6
Región ordenada se extiende por uno