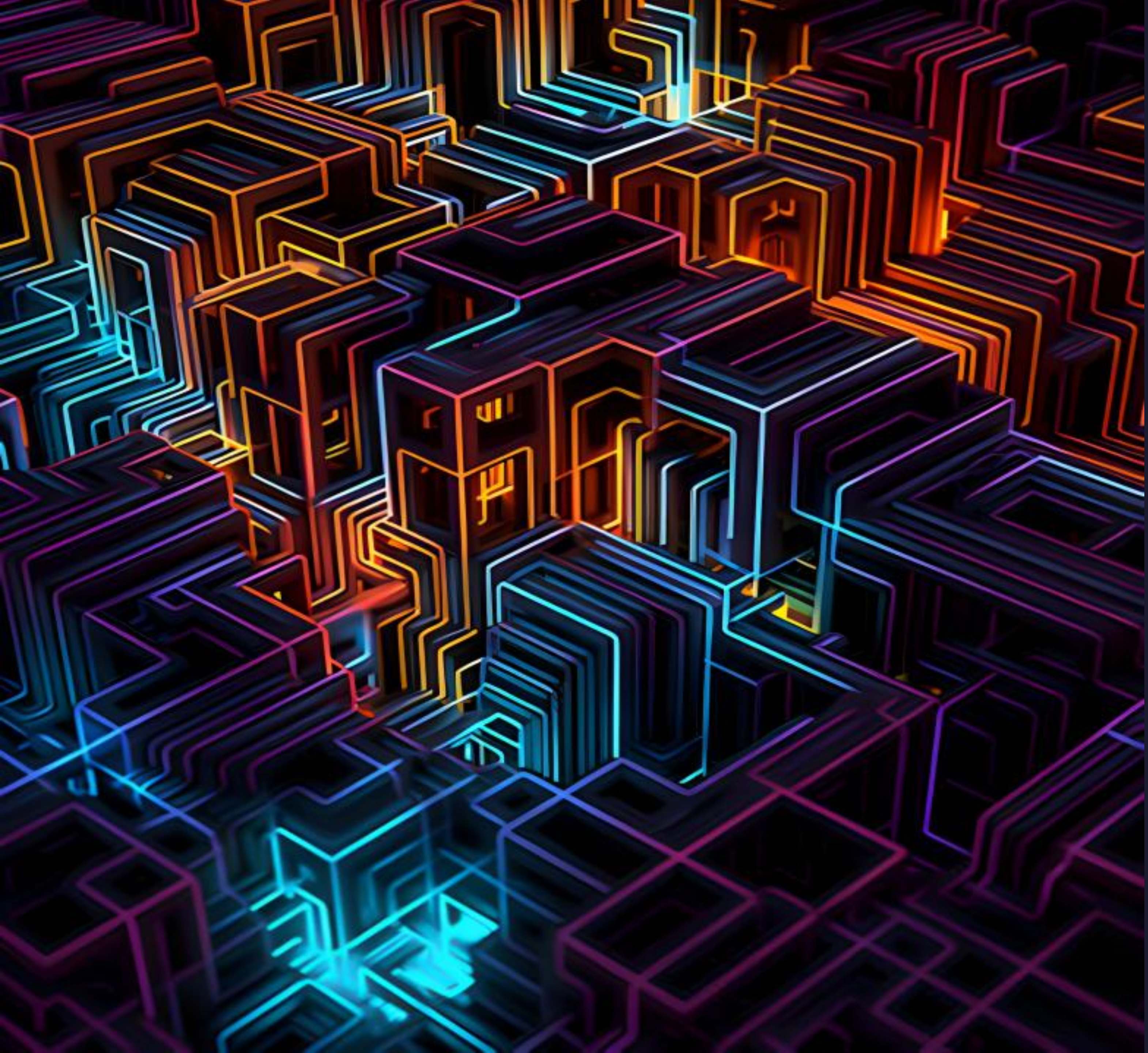


Técnicas de diseño de algoritmos: decrementa y vencerás

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

Tecnológico de Monterrey

Decrementa y vencerás

Enfoque: explotar la relación entre un problema de tamaño n y versiones del mismo problema, con un tamaño menor

Produce algoritmos incrementales, que pueden ser:

Top-down	Bottom-up
Usualmente recursivo	Usualmente iterativo
Resuelve para n con llamadas recursivas para subproblemas de tamaño $m < n$	Resuelve desde $i < n$, incrementando i en ciclos $for/while$

Decrementa y vencerás

Top-down

Usualmente recursivo
Resuelve para n con llamadas recursivas para subproblemas de tamaño $m < n$

Bottom-up

Usualmente iterativo
Resuelve desde $i < n$, incrementando i en ciclos $for/while$

Ejemplo
Algoritmo de Euclides para el máximo común divisor, basado en la relación:

$$\gcd(a, b) = \begin{cases} a & \text{si } b = 0 \\ \gcd(b, a \bmod b) & \text{si } b \geq 1 \end{cases}$$

Programa ambas versiones

$$\gcd(55, 34) = ?$$

Tipos de decremento

Algoritmos con decremento constante

Típicamente, la constante es 1, y se decrementa como $n-1$

Algoritmos con decremento por un factor constante

El factor suele ser 2, decrementando como $n/2$

Algorimos con decremento variable

El decremento no es fijo

Hay 3 tipos de decrementos en los algoritmos de tipo decrementa y vencerás

Tipos de decremento

Decremento constante

Típicamente, la constante es 1, y se decrementa como $n - 1$

Factoriales, Fibonacci y exponenciación de a^n básico

Insertion-sort

Decremento por un factor constante

El factor suele ser 2, decrementando como $n/2$

Búsqueda binaria, y variantes

Decremento variable

El decremento no es fijo

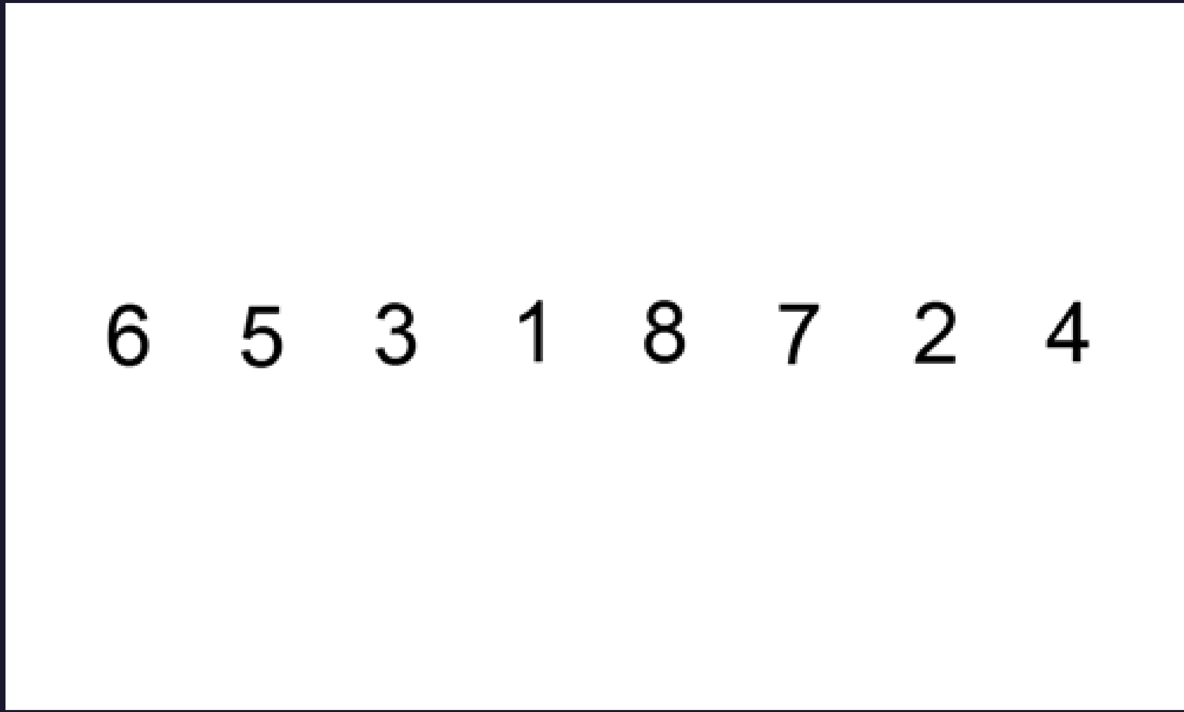
Inserción y búsqueda en arboles binarios (BST)

Decremento constante: insertion-sort

Inserción (Insertion sort)

$O(??)$

Inserta cada elemento en la posición correcta, desplazando al resto



Decremento constante: insertion-sort

Inserción (Insertion sort)

$O(n^2)$

Para ordenar los n elementos, ordenamos primero hasta $n - 1$

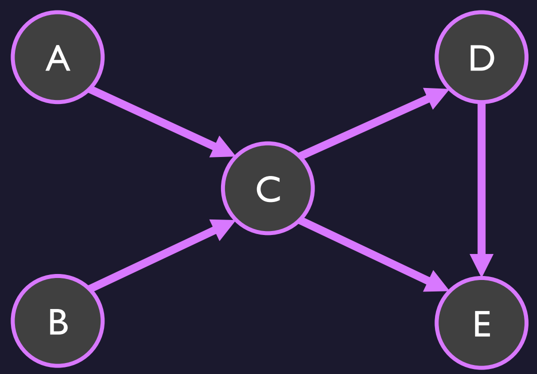
```
ALGORITHM insertionSort(A, n)
// Input: Un arreglo A con n elementos, su tamaño n
// Output: A ordenado
for i = 0 to n − 1
    c = A[i]
    j = i − 1
    // Desplaza los elementos mayores que c
    while j ≥ 0 and A[j] > c do
        A[j + 1] = A[j]
        j = j − 1
    end while
    A[j + 1] = c
end for
```

Decremento constante: ordenamiento topológico

Se utiliza en grafos dirigidos

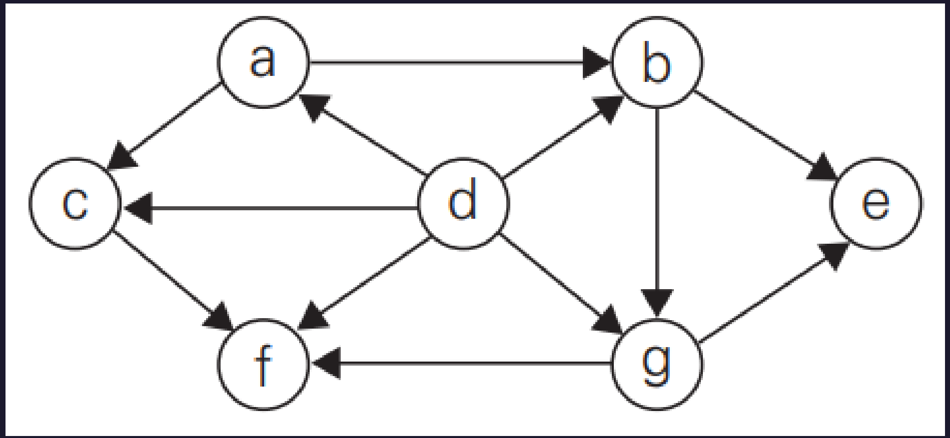
El ordenamiento topológico ayuda a encontrar un orden para realizar tareas/procesos que dependen unos de otros

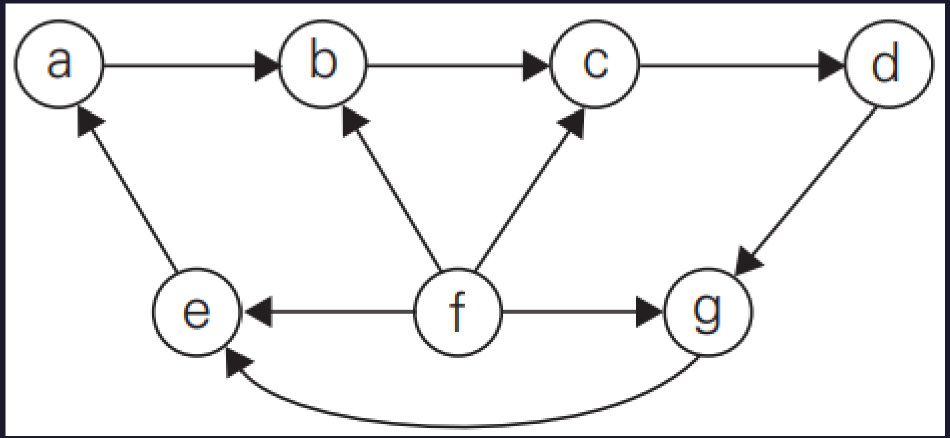
```
ALGORITHM topoSort(G)
// Computa el ordenamiento topológico del grafo G
// Input: Un grafo dirigido G
// Output: orden de visita de los n nodos

L = lista vacía que guardara el orden de visita
while G tenga un nodo v sin aristas de entrada do
        Agrega v a la lista L
        Remueve v y sus aristas de G
end while
// Si L termina teniendo n nodos, G tiene un ordenamiento topológico y sin
tiene ciclos. De lo contrario, G no se puede resolver por ser un grafo cíclico.
return L
```



Decremento constante: ordenamiento topológico

1) Practica con estos dos grafos





1) Encuentra un nodo v sin aristas de entrada
2) Agrégalo a la lista L
3) Remuévelo de G, junto con sus aristas
4) Repite hasta que no queden nodos, o no encuentres un nodo v sin aristas de entrada

2) Programa este algoritmo y analiza su complejidad

Decremento por factor constante: búsqueda binaria

```
ALGORITHM   binarySerch(A, n, x)
// Computa el índice index donde x esta en A, devuelve −1, si x no se encuentra
// Input: Un arreglo ordenado A, su tamaño n, elemento buscado x
// Output: El índice de x en A, o −1 si no esta
l = 0
r = n − 1
while l ≤ r do
    i = l + (r − l)/2
    if A[i] == x then
        return i
    else if x < A[i] then
        r = i − 1
    else
        l = i + 1
    end if
end while
return -1
```

Search for 47

0	4	7	10	14	23	45	47	53

Decremento por factor constante: fake-coin

Reglas:

- Tienes n monedas, pero 1 es falsa
- La moneda falsa es idéntica al resto, excepto que tiene un peso diferente
- Tienes una balanza y puedes comparar los pesos de grupos de una o varias monedas

¿Cómo sería un algoritmo eficiente para esto?

Pista: No es revisar una por una

Decremento variable: arboles BST

Los arboles BST son estructuras para almacenamiento eficiente de información y búsquedas rápidas

Propiedad de los arboles binarios de búsqueda:

El valor en un nodo (llave) es:

- Mayor a todos los valores en su subárbol izquierdo
- Menor que todos los valores en su subárbol derecho

¿Por qué son de decremento variable? Veamos un ejemplo

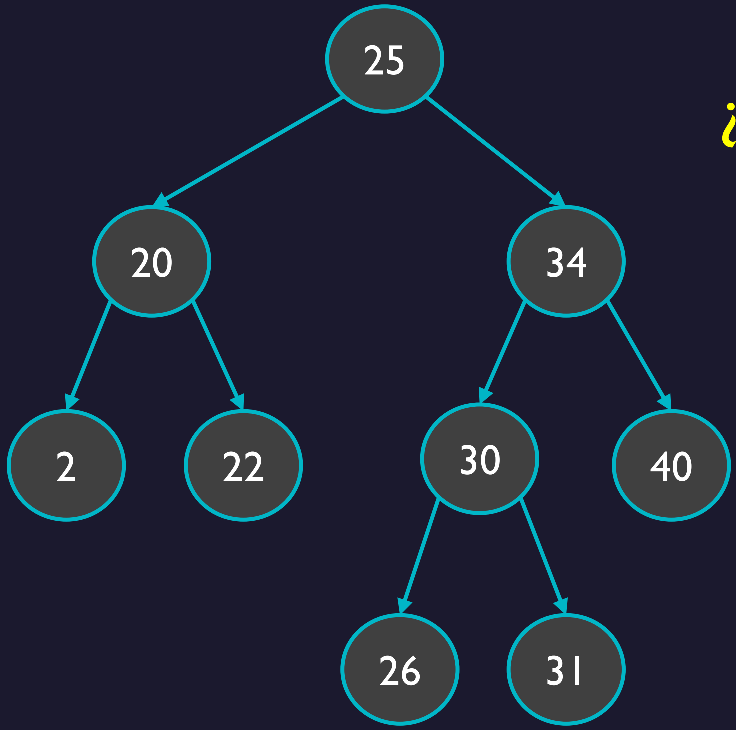
Crea dos BST para los sig. valores:

a) 25, 20, 34, 2, 22, 30, 40, 26, 31

b) 2, 20, 22, 25, 26, 30, 31, 34, 40

Decremento variable: arboles BST

a) 25, 20, 34, 2, 22, 30, 40, 26, 31

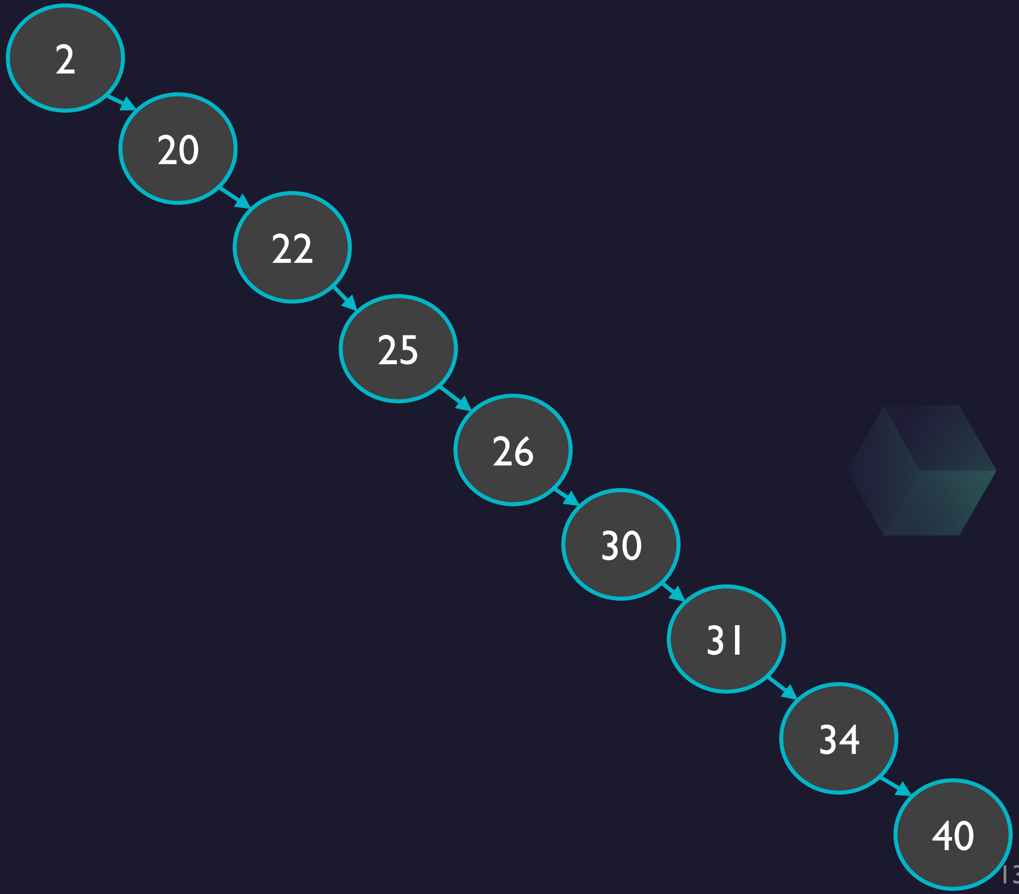


¿Cuál es la complejidad de insertar y buscar?

Mejor caso:
$O($? $)$

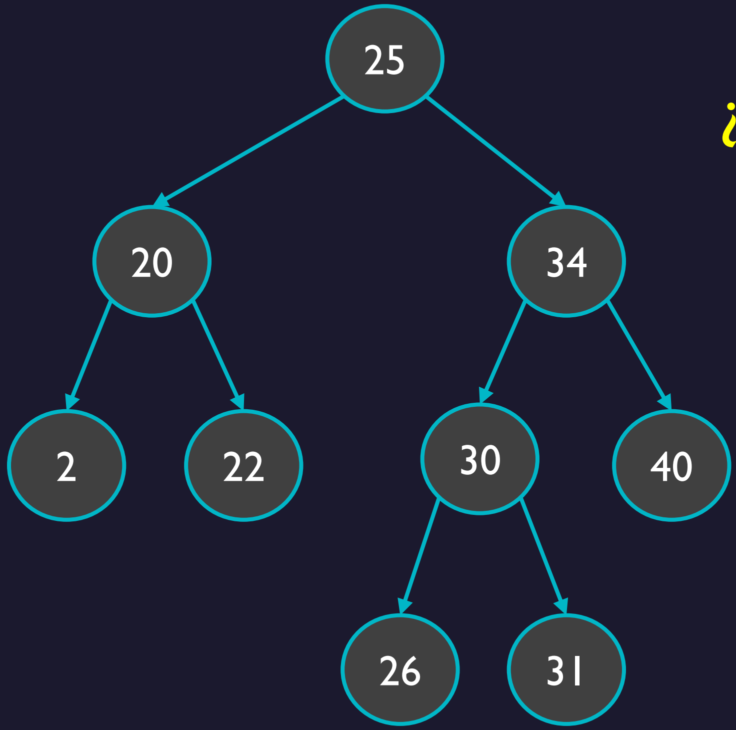
Peor caso:
$O($? $)$

b) 2, 20, 22, 25, 26, 30, 31, 34, 40

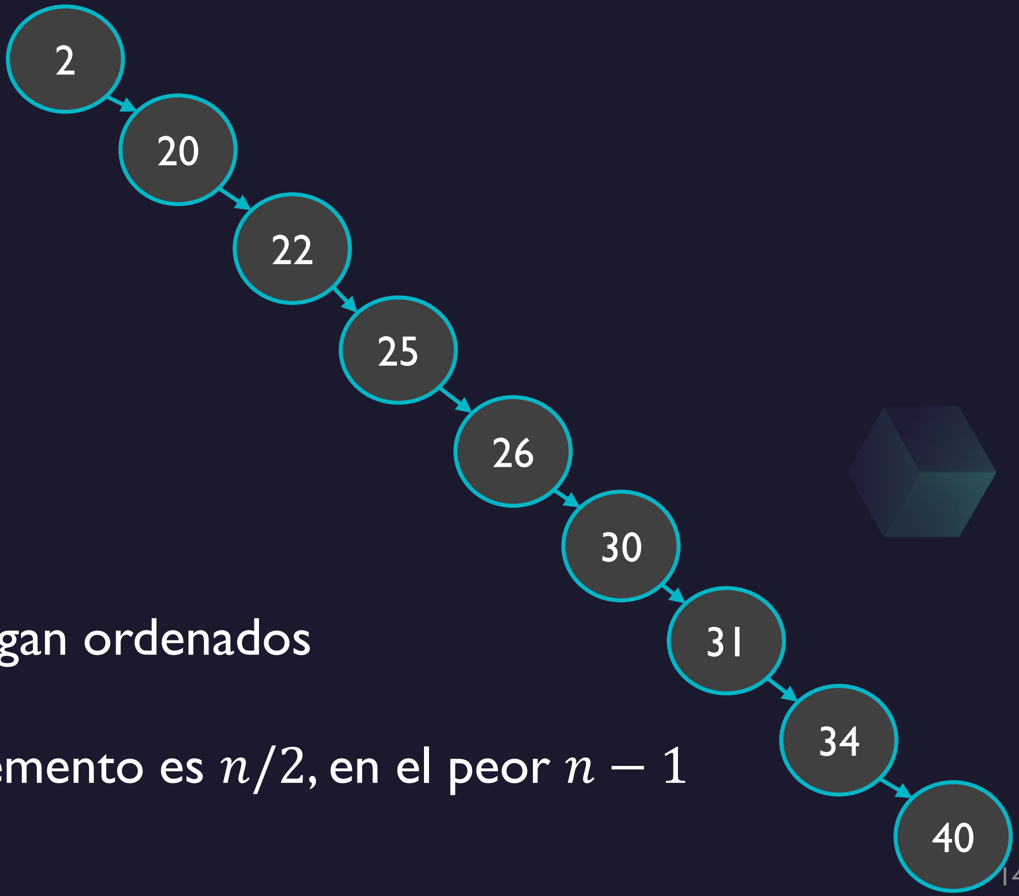


Decremento variable: arboles BST

a) 25, 20, 34, 2, 22, 30, 40, 26, 31



b) 2, 20, 22, 25, 26, 30, 31, 34, 40



¿Cuál es la complejidad de insertar y buscar?

Mejor caso:
$O(\log n)$

Peor caso:
$O(n)$, si los elementos llegan ordenados

En el mejor caso, el decremento es $n/2$, en el peor $n-1$

Decremento variable: GDC

Algoritmo de Euclides para el máximo común divisor

Basado en la relación:

$$\gcd(a,b) = \begin{cases} a & \text{si } b = 0 \\ \gcd(b, a \bmod b\,) & \text{si } b \geq 1 \end{cases}$$

Ejemplo:

$$\gcd(60,24) =$$
$$\gcd(24,12) =$$
$$\gcd(12,0) = \ 12$$

Muy fácil como algoritmo recursivo

En C/C++ y Python el modulo residuo es: a $\%\ b$

Decremento variable: selection problem

El problema: encontrar el k-esimo elemento en orden de magnitud en un arreglo desordenado

La estrategia decrementa-y-vencerás para este problema usa el particionamiento con un elemento pivote

Ejemplo: el pivote es $A[0] = 6$

Movemos los elementos menores que el pivote a su derecha, los mayores a la izquierda

6	1	24	4	5	7	2	8	9
0	1	2	3	4	5	6	7	8

El pivote queda en el índice s (que le correspondería si A estuviera ordenado)

2	1	4	5	6	7	24	8	9
0	1	2	3	4	5	6	7	8
				s				

¿Cómo podríamos encontrar el elemento correcto para el índice $k = 2$?

Decremento variable: selection problem

Buscamos el valor del elemento que debería ocupar el índice $k = 2$

Con el primer particionamiento, averiguamos que 6 va en el índice $s = 4$

Además, colocamos a todos los elementos menores que 6 a su izquierda

Por lo tanto, el valor que buscamos debe existir en los índices 0 a 3

2	**1**	**4**	**5**	**6**	**7**	**24**	**8**	**9**
0	1	2	3	4	5	6	7	8
				s				

Uno de estos debe ser el tercer elemento mas pequeño, al que le correspondría el índice 2

Estrategia: repetir el proceso en una sección reducida del arreglo, hasta que $s = k$

Decremento variable: selection problem

El particionamiento: Algoritmo de Lomuto

```
ALGORITHM lomuto(A, L, R)
// Input:  Arreglo A, inicio L y fin R de la sección
// Output:  Arreglo particionado e índice del pivote s

s = L
pivote = A[0]
for i = L + 1 to R
        if A[i] < pivote then
                s = s + 1
                Intercambia A[s] con A[i]
        end if
end for
Intercambia A[s] con A[L]
return A, índice s
```

Si llamamos $lomuto(A, 0, n)$ con $n = 9$, ocurre esto:

6	1	24	4	5	7	2	8	9
0	1	2	3	4	5	6	7	8

2	1	4	5	6	7	24	8	9
0	1	2	3	4	5	6	7	8
				s				

Decremento variable: selection problem

¿Cómo usarías el algoritmo de Lomuto para encontrar el elemento del índice k?

Recuerda el ejemplo:

2	1	4	5	6	7	24	8	9
0	1	2	3	4	5	6	7	8
				s				

Uno de estos debe ser el tercer elemento mas pequeño, al que le corresponde el índice 2

En cada iteración, el problema se reduce de manera variable porque no sabemos donde quedara el pivote

Por lo tanto:

Si $k = s$, ya tienes la respuesta, es $A[s]$

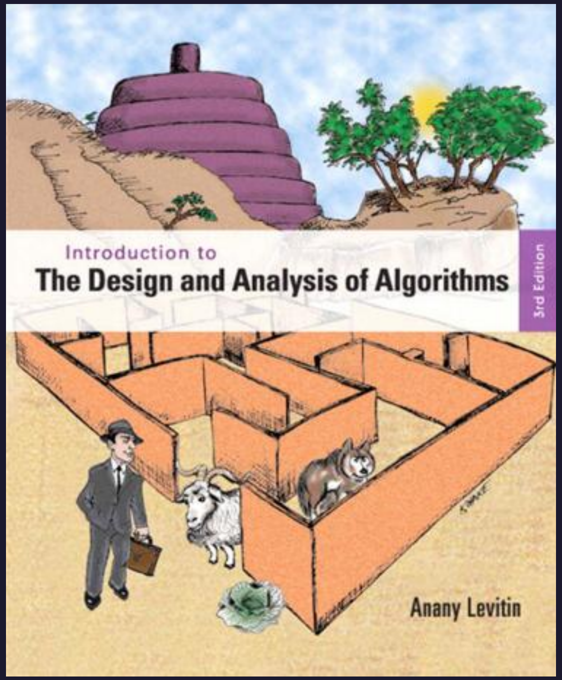
¿Y si $s > k$?
Actualiza $r = s$

¿Y si $s < k$?
Actualiza $l = s + 1$ hasta r

Repite el algoritmo de Lomuto para la nueva sección de l hasta r

Este algoritmo se llama quickSelect

Lectura sugerida



Introduction to The Design and Analysis of Algorithms
Anany Levitin, 3rd Ed

Capitulo 4: Decrease and conquer

Muy recomendable leer al menos la intro del capitulo y las secciones de los algoritmos vistos hoy:

4.1 Insertion sort
4.2 Topological sorting
4.4 Binary search, fake coin
4.5 Selection problem con algoritmo de Lomuto y quickselect

Algoritmos para generación de permutaciones y subsets

Analiza y responde:

¿Los algoritmos para generar todas las permutaciones y subconjuntos de n elementos (vistos en la clase anterior) son o no algoritmos decrease-and-conquer?

¿Porqué si? ¿Porque no?
Si tu respuesta es si, ¿qué factor de decremento tienen?
Entonces ¿estos algoritmos son también búsqueda exhaustiva o no?

Actividad 1.2

Fecha:
el día de la próxima clase

Detalles y puntajes en Canvas

- Ordenamiento topológico (grafos de los ejemplos como casos de prueba)
- Algoritmo para fake-coin
- Algoritmo quickSelect para selection problem

- Respuestas para la diapositiva anterior

No olvides incluir las complejidades!

Reminder: entregar en cada actividad

- Todos los scripts, con complejidades en comentarios
- Archivos de entrada (cuando los haya)
- Capturas de pantalla de resultados y partes relevantes del código. Pueden ser imágenes (jpg o png), o estar en un pdf
- Link a Replit / Colab, con permisos de editor (puede ir en un comentario, o dentro del pdf)