



Complejidad Computacional

- **Algoritmia**
- **Complejidad computacional**

Complejidad Computacional

- **Algoritmia:** Diseñar y analizar sistemáticamente los algoritmos específicos, donde c/u de los cuales será mas eficiente que sus predecesores para resolver un problema dado.
- **Complejidad Computacional:** se maneja paralelamente pero considera globalmente todos los posibles algoritmos para resolver un problema planteado, incluyendo algoritmos que ni siquiera pueden existir o no se ha pensado aún.
- Todo algoritmo tiene una serie de características, entre otras que requiere de recursos, algo que es fundamental considerar a la hora de implementarlos en una maquina. Estos recursos son principalmente:
 - El tiempo: período transcurrido entre el inicio y la finalización del algoritmo.
 - La memoria: la cantidad (la medida varía según la máquina) que necesita el algoritmo para su ejecución.
- Para ello empleamos la algoritmia, para resolver el problema en estudio en un tiempo $O(f(n))$ para alguna función $f(n)$ que intentamos reducir lo mas posible.

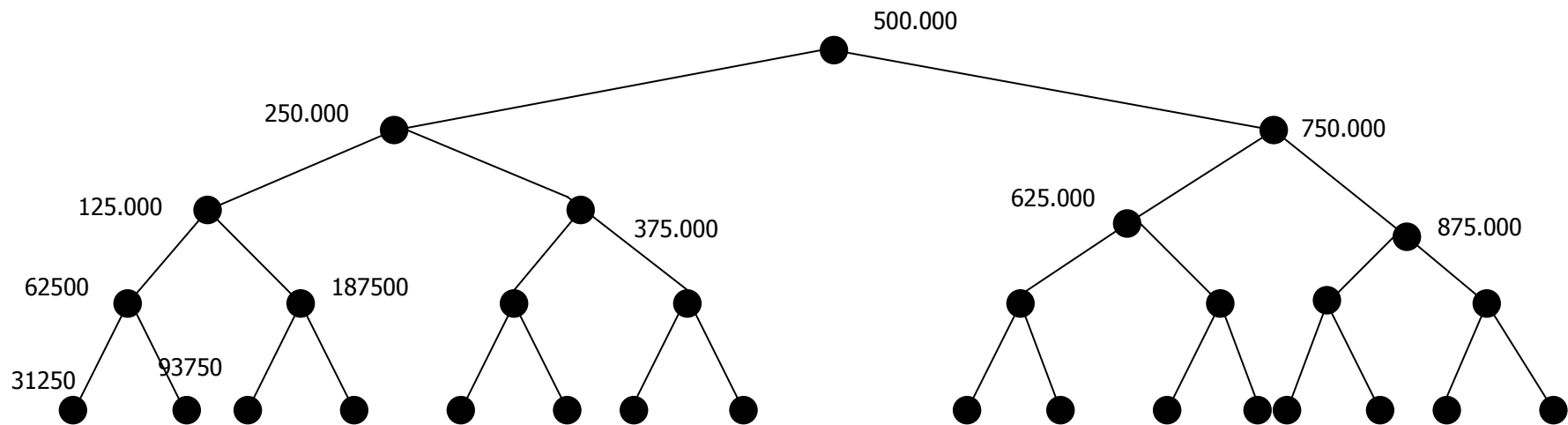
Complejidad Computacional

- También al emplear la complejidad, se intenta hallar una función $g(n)$ tan grande como sea posible, para lo cual podemos demostrar que cualquier algoritmo capaz de resolver correctamente nuestro problema en todos sus casos debe necesariamente requerir un tiempo que este en $\Omega(g(n))$, a esto se le conoce como cota inferior de la complejidad del problema.
- Al final quedamos satisfechos cuando $f(n) \in \Theta(g(n))$ y podemos afirmar que hemos encontrado el algoritmo mas eficiente posible para nuestro problema.
- Incluso, de vez en cuando podemos encontrar el número exacto de veces que es precisa una operación, como por ejemplo el de la comparación para resolver un problema.

Complejidad Computacional

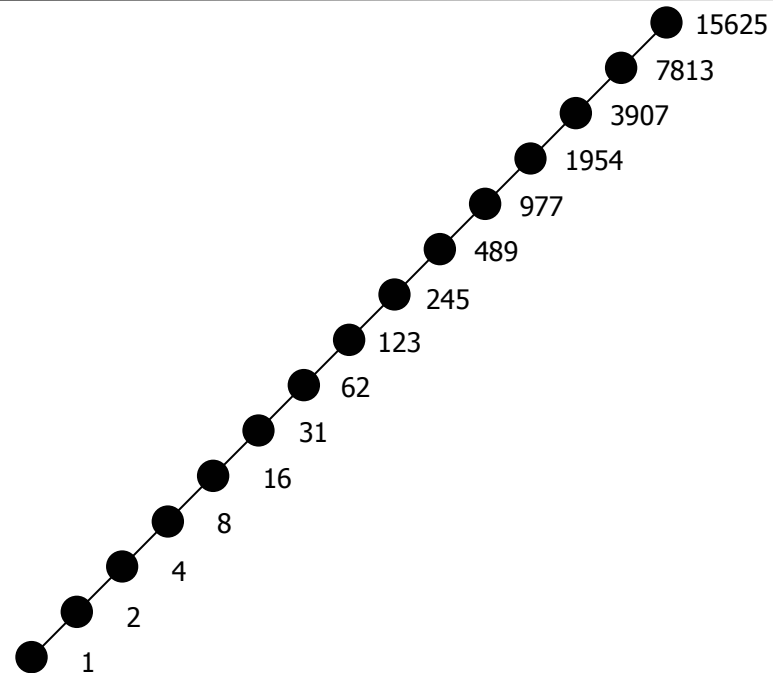
■ Un ejemplo sencillo

Buscar un número entre 1 y 1 millón



Carlos Ramirez. Catedra de Programación. Diseño y Analisis de Algoritmos. Junio 2005

Complejidad Computacional



Carlos Ramirez. Catedra de
Programación. Diseño y Analisis
de Algoritmos. Junio 2005

Complejidad Computacional

- Juego de las 20 preguntas: 1 – 1.000.000, tipo preguntas si o no, la idea es aplicar divide y vencerás para resolver el acertijo.
- La respuesta debería encontrarse en un máximo de 20 preguntas porque 1 millón es menos que 2^{20} , este algoritmo muestra que 20 preguntas son suficientes para encontrar la solución al problema y esto corresponde a la algoritmia, pero que si las 20 preguntas fuesen o no necesarias corresponde a la complejidad.
- Sea S_i el conjunto de candidatos al numero buscado después de la i -ésima pregunta, $k_i = |S_i|$ el numero de candidatos restantes. Inicialmente, S_0 contiene todos los enteros positivos hasta 1 millón, por tanto $k_0 = 10^6$, y Q_i la i -ésima pregunta y $Q_i(n)$ denota la respuesta a esa pregunta si el numero buscado es n . Entonces sea $Y_i = \{n \in S_{i-1} \mid Q_i(n) = \text{“si”}\}$, y sea $N_i = \{n \in S_{i-1} \mid Q_i(n) = \text{“no”}\}$; entonces $N_i \cup Y_i = S_{i-1}$, por tanto $|S_{i-1}| = |Y_i| + |N_i|$ lo cual implica que al menos uno de Y_i o N_i contiene $\lceil k_{i-1}/2 \rceil$ números o mas. Así es posible que $k_i \geq \lceil k_{i-1}/2 \rceil$ para todo i . Dado que $k_0 = 10^6$, puede ser que $k_1 \geq 500000$ o que $k_2 \geq 250000$, ..., $k_{19} \geq 2$. Concluyendo que pueden quedar por lo menos 2 candidatos al número buscado al cabo de 19 preguntas y por tanto se necesitan 20 preguntas para resolver el acertijo.

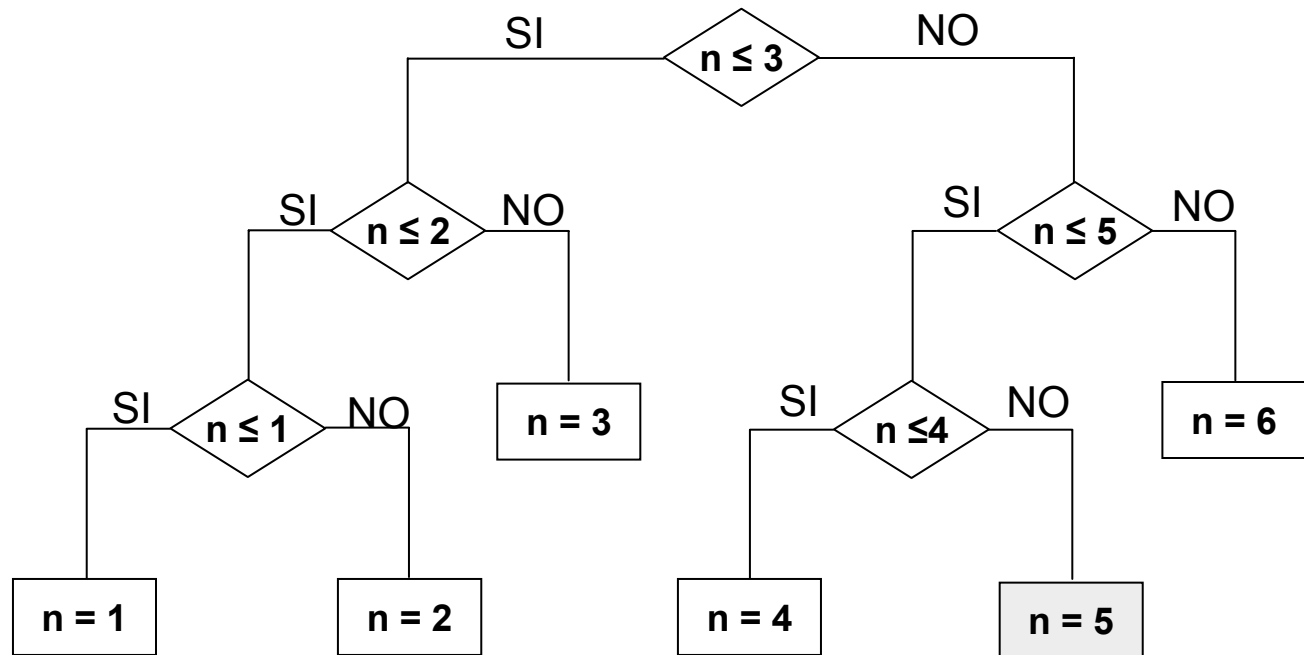


Complejidad Computacional

Argumentos de teoría de la información

- Se aplica a toda una gama de problemas, especialmente aquellos que implican la comparación de elementos. Un árbol de decisión es una forma de representar el funcionamiento de un algoritmo para todos los datos posibles de un tamaño dado.
- Este árbol es binario con raíz donde los nodos internos contienen pruebas de los datos y las hojas contienen la salida, veredicto o respuesta al problema.
- Para el juego de las 20 preguntas se dio el caso, pero explicaremos para hallar un número entre 1 y 6.. Cual sería la altura o el número de preguntas?

Complejidad Computacional



Carlos Ramirez. Catedra de
Programación. Diseño y Analisis
de Algoritmos. Junio 2005

Complejidad Computacional

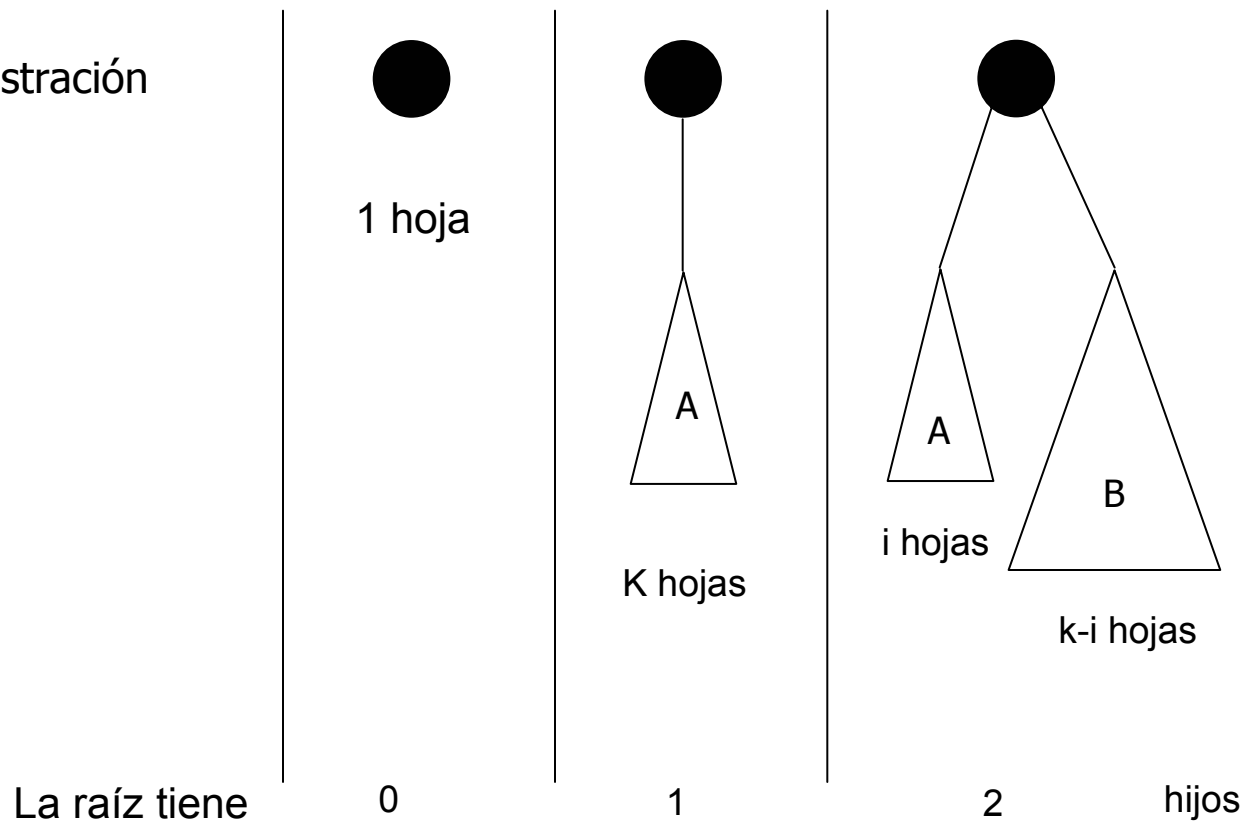
- Es relevante si alcanzamos la respuesta o no a todo algoritmo determinista, ya que para jugar le corresponde un árbol de decisión, siempre y cuando exista un límite para el número de preguntas que puede hacer el algoritmo.
- Por eso se puede pensar en un árbol de decisión como un algoritmo.
- En este caso el algoritmo está hecho para que podamos acceder a cualquier hoja del árbol. Por ello a medida que se van haciendo preguntas, la altura del árbol aumenta, que sería para el peor caso.
- Para el juego de las 20 preguntas, el árbol de decisión tendría altura 20 pero eso correspondería a un millón de hojas, pero siendo un árbol binario, la altura del árbol se calcula 2^k hojas y 2^{19} es < 1 millón, concluyendo que el juego no se puede resolver con 19 preguntas en el peor caso.
- Por eso tomamos la altura de las hojas y las sumamos todas y dividiendo entre el número de hojas obtenemos.

Complejidad Computacional

- Altura media seria $(3+3+2+3+3+2)/6 = 16/6$. Así como el árbol podado nos da el tiempo para el peor caso, la altura media nos da el caso promedio.
- **Teorema: todo árbol binario con k hojas tiene una altura media que es como mínimo $\lg_2 k$**
- Si un número buscado se selecciona aleatoriamente entre 1 y 6 según la distribución uniforme, entonces cualquier algoritmo para jugar este juego tiene que hacer al menos $\lg_2 6 \approx 2,585$ preguntas en el caso promedio. La solución dada hace $16/6$ preguntas en el caso promedio, una mejora seria $15/6 = 2,5$ preguntas pero debe descartarse porque $\lg_2 6 > 15/6$.
- Concluyendo que el árbol de decisión proporciona un algoritmo óptimo cuando el número buscado está entre 1 y 6 para el caso promedio y el peor.
- De igual manera se necesitan 20 preguntas para el caso peor para el juego original con un millón de veredictos y ningún algoritmo puede hacer menos de $\lg_2 10^6 \approx 19,93$ preguntas en el caso promedio si todos los veredictos son igualmente probables.

Complejidad Computacional

- Demostración



Complejidad Computacional

- Sea T un árbol con k hojas. Se define $H(T)$ como la suma de las profundidades de las hojas. Para el ejemplo del árbol de decisión de 1 a 6, $H(T) = 16$, la altura media de T es $H(T)/k$ y el objetivo es demostrar que $H(T) \geq k \lg(k)$. Como observamos en el dibujo anterior la raíz T puede tener 0, 1 o 2 hijos, en el primer caso, la raíz es única hoja del árbol, es decir, $k = 1$ y $H(T) = 0$, en el segundo caso el único hijo es la raíz de un subárbol A , que tiene k hojas. La distancia de cualquier hoja a la raíz de T es uno más que la distancia desde esa misma hoja a la raíz de A , así que $H(T) = H(A) + k$; y en el tercer caso tenemos 2 subárboles A y B con i y $k-i$ hojas para algún $1 \leq i \leq k$, obteniendo $H(T) = H(A) + H(B) + k$.
- Para $k \geq 1$, se define $h(k)$ como el menor valor posible para $H(X)$ cuando X es un árbol binario con k hojas. En particular, $H(T) \geq h(k)$, claramente se ve que $h(1) = 0$. Si definimos $h(0) = 0$ da a lugar

$$h(k) = \begin{cases} 0 & \text{si } k \leq 1 \\ \min_{0 \leq i \leq k} (h(i) + h(k-i) + k) & \text{en caso contrario} \end{cases}$$

Complejidad Computacional

- Tomando como hipótesis que $h(j) \geq j \lg_2 j$ para todo entero positivo $j < k$. Por definición:

$$h(k) = k + \min_{0 \leq i \leq k-1} (h(i) + h(k-i))$$

- Por la hipótesis de inducción,
- $$h(k) \geq k + \min_{0 \leq i \leq k-1} (i \lg(i) + (k-i) \lg(k-i))$$

- Sea $g: [1, k-1] \rightarrow \mathbb{R}$ define como $g(x) = x \lg_2 x + (k-x) \lg_2 (k-x)$, calculando la 1ª y 2ª derivada y hallando su mínimo tenemos:

$$\min_{i \in [1..k-1]} g(i) \geq \min_{x \in [1, k-1]} g(x) \geq k \lg(k) - k$$

- Reuniéndolo todo alcanzamos que:

$$H(T) \geq h(k) \geq k + \min_{i \in [1..k-1]} g(i) \geq k \lg(k)$$

- Y por tanto la altura media de T , que es $H(T)/k$ es al menos $\lg(k)$



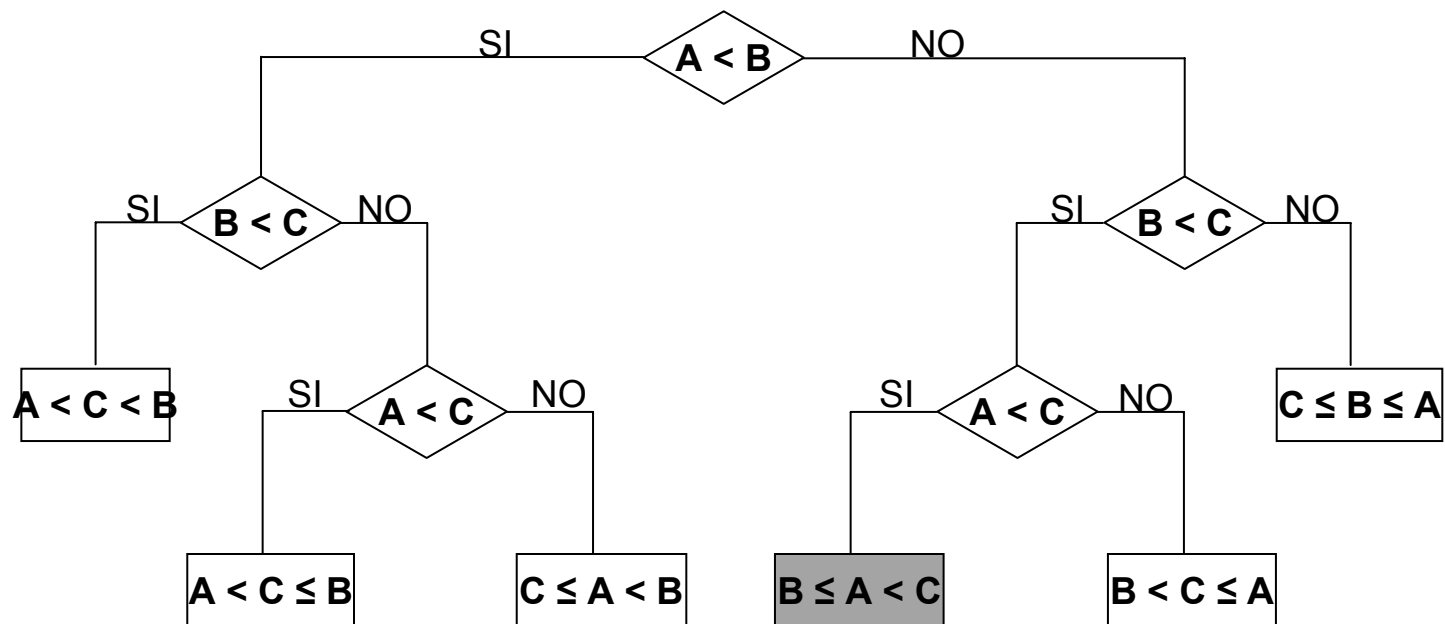
Complejidad Computacional

La complejidad de la ordenación

- Cual es el número mínimo de comparaciones necesarias para ordenar n elementos. Tomaremos en cuenta los algoritmos de ordenación basados en comparaciones, es decir, compararlos en parejas para determinar si son iguales o cual de ellos es mayor.
- Por tanto la pregunta en si seria: Cual es el numero mínimo de comparaciones necesarias en cualquier algoritmo para ordenar n elementos por comparación?
- Para ello se recurre nuevamente a los árboles de decisión, donde se asocia a cada posible relación de orden entre los elementos un veredicto o respuesta compatible con esta relación.
- Todo árbol de decisión valido para ordenar n elementos da lugar a un algoritmo de ordenación in situ (ad hoc) para el mismo número de elementos.

Complejidad Computacional

La complejidad de la ordenación



Carlos Ramirez. Catedra de Programación. Diseño y Análisis de Algoritmos. Junio 2005

Complejidad Computacional

La complejidad de la ordenación

Algoritmo para ordenar 3 elementos

Procedimiento ordenacioninsitu3(T[1..3])

A <- T[1]; B <- T[2]; C <- T[3]

si A < B entonces

 si B < C entonces { ya están ordenados }

 sino si A < C entonces T <- A, C, B

 sino T <- C, A, B

sino si B < C entonces

 si A < C entonces T <- B, A, C

 sino T <- B, C, A

sino T <- C, B, A

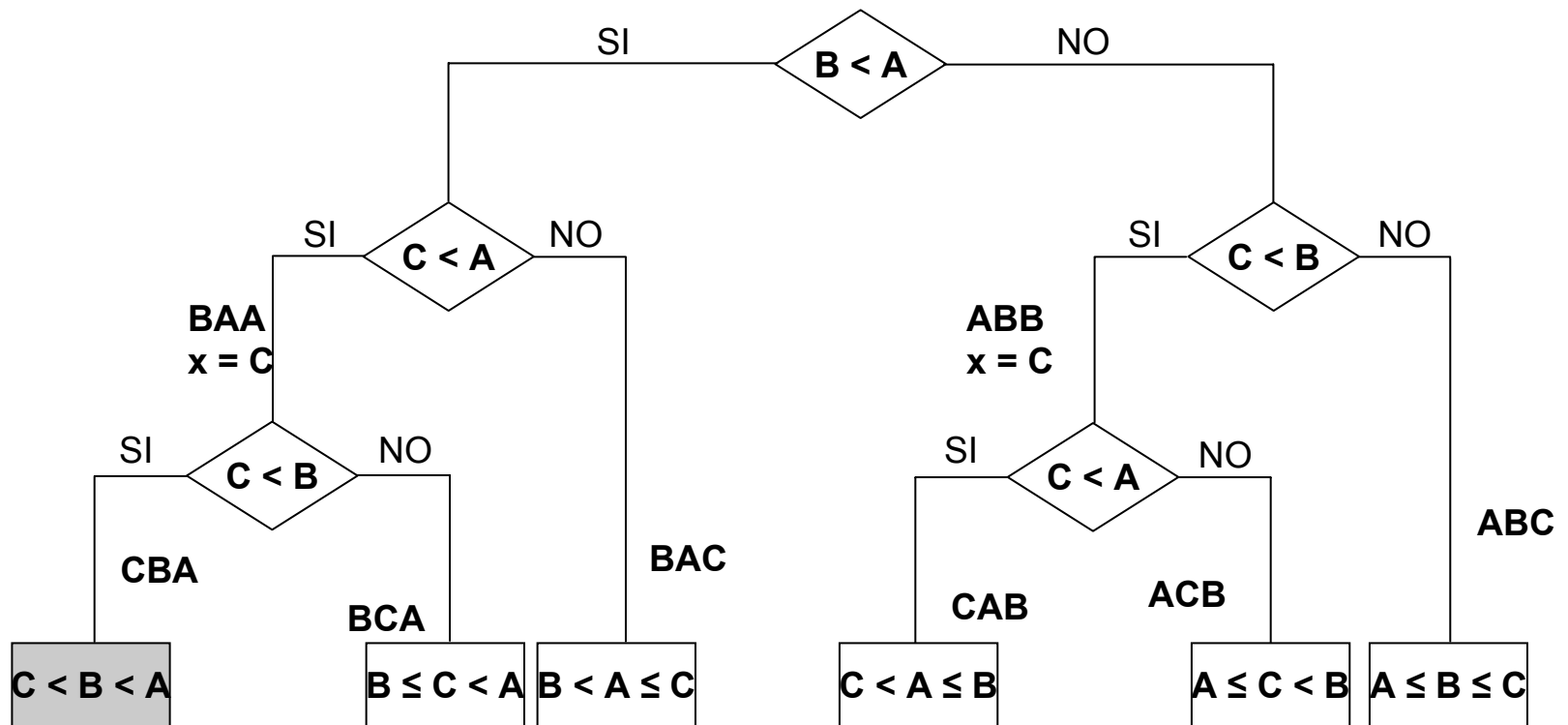


Complejidad Computacional

- Todo árbol de decisión para ordenar tres elementos son de altura 3
- Se puede hallar algún árbol de decisión valido para ordenar tres elementos con altura < 3 ?
- No, porque todo algoritmo correcto de ordenación debe ser capaz de producir al menos seis veredictos diferentes para $n = 3$, puesto que es el numero de permutaciones de tres elementos.
- Por esta razón cada árbol de decisión valido para ordenar n elementos debe contener al menos $n!$ hojas, así como tener una altura que sea mínimo $\lceil \lg_2 n! \rceil$ y una altura media que sea como mínimo $\lg_2 n!$.
- Esto implica que el algoritmo que ordena n elementos debe al menos hacer $\lceil \lg_2 n! \rceil$ comparaciones en el caso peor y $\lg_2 n!$ en el caso promedio.

Complejidad Computacional

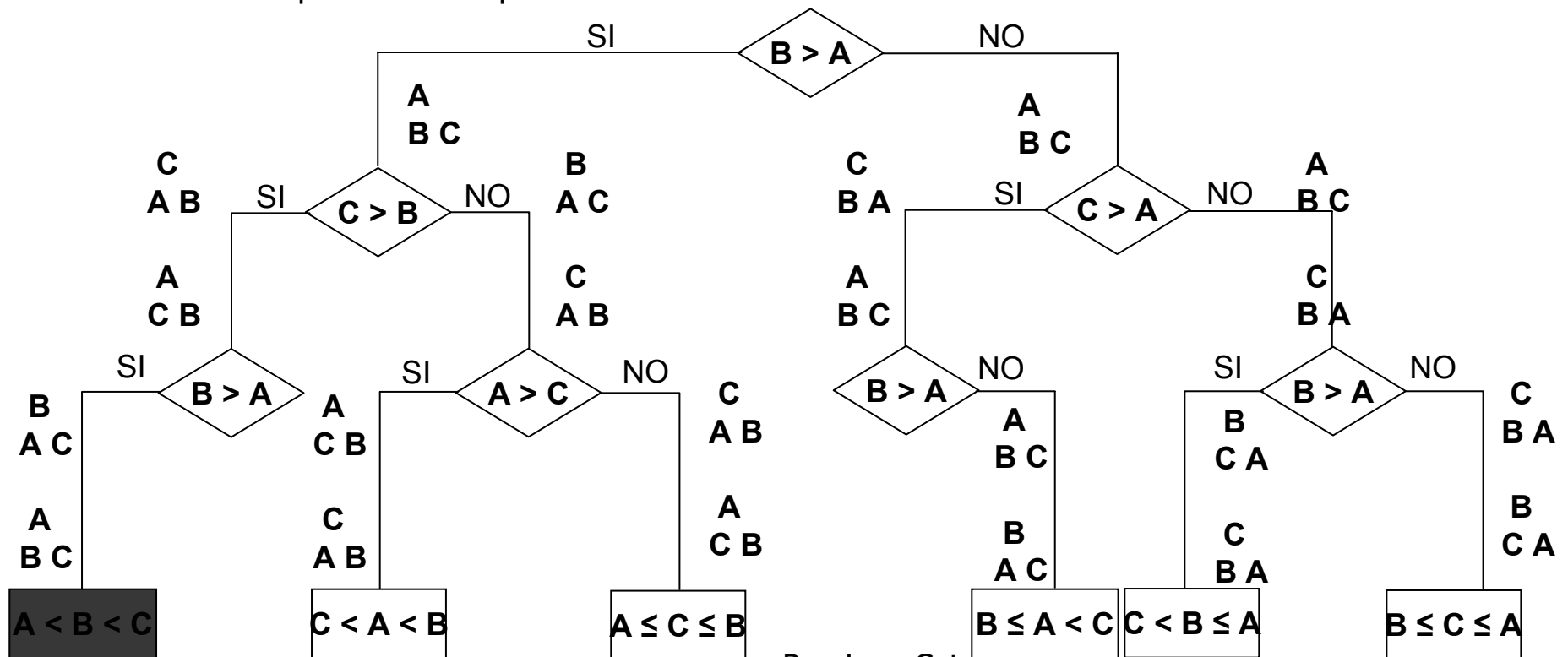
- Ordenación por inserción de 3 elementos



Carlos Ramirez. Catedra de Programación. Diseño y Analisis de Algoritmos. Junio 2005

Complejidad Computacional

- Ordenación por montículo para 3 elementos



Carlos Ramirez. Catedra de Programación. Diseño y Analisis de Algoritmos. Junio 2005



Complejidad Computacional

Gracias por su
atención.