Complejidad (I)

Gregorio Hernández

UPM

Matemática Discreta II

(MI)

Complejidad (o coste) de un algoritmo

Algoritmo: secuencia de instrucciones para resolver un problema

Programa: algoritmo escrito en un lenguaje de programación, junto con las estructuras de datos utilizadas

La complejidad mide la eficiencia del algoritmo en la resolución del problema. Se mide en:

- Tiempo (número de operaciones realizadas)
- Espacio o cantidad de memoria utilizada

  - en el peor caso
  - en media

Complejidad de un problema

- Modelo teórico de máquina
- Modelo RAM (Random Access Machine) real
  - Un nº real se almacena en una unidad de memoria
  - Operaciones primitivas de coste unitario

    suma, producto,
    comparación,
    asignación,
    operaciones lógicas …
- El coste del algoritmo se mide como una función del número de datos de entrada, T(n)

Calcular el número de aristas de un grafo de n vértices

Estrategia: A partir de la matriz de adyacencia

```
Edges:=0
        for u=1 to n-1 do
                for v=u+1 to n do
                        if Adj(u,v)=1 then Edges:= Edges + 1
return(Edges)
```

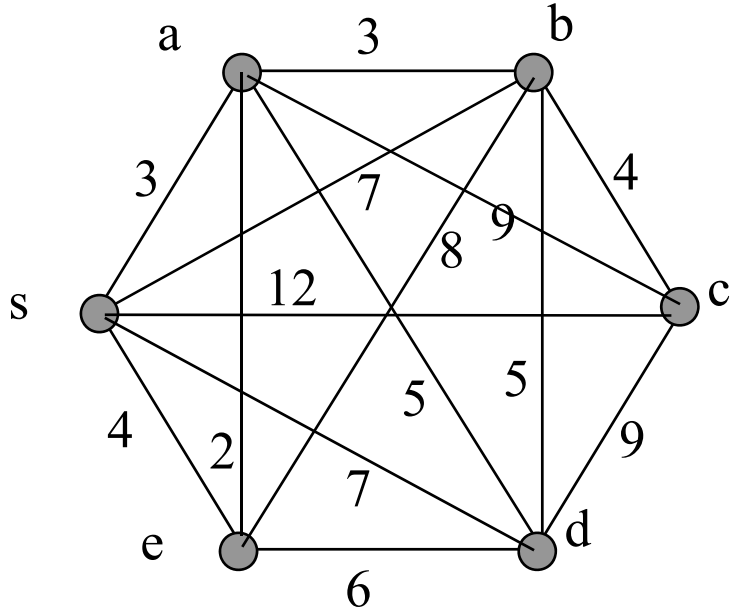
¿Cuántas operaciones?

$T(n) = 1 + (2(n-1) + 2(n-2) + \ldots + 2\cdot 1) + 1 = 2 + 2\,\frac{n(n-1)}{2}$

$T(n) = n^2 - 2n + 2$

Problema del viajante TSP

Un viajante de comercio desea visitar n ciudades volviendo al punto de partida. ¿Qué ruta debe seguir para minimizar el coste de la ruta recorrida?



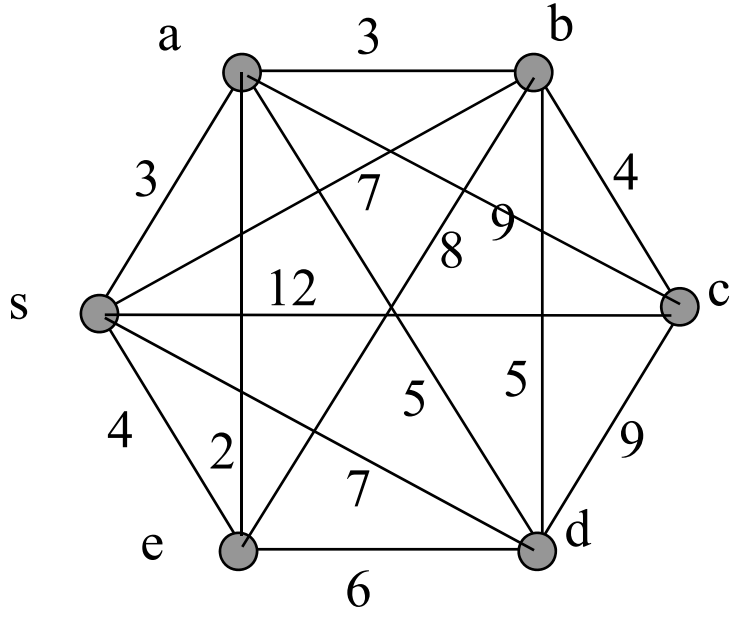
La ruta scdbeas tiene coste 39
La ruta sadcbes tiene coste 33
La ruta …….

¿Cuántas rutas posibles hay?

Cada ruta es un ciclo en el grafo

Problema del viajante TSP

Un viajante de comercio desea visitar n ciudades volviendo al punto de partida. ¿Qué ruta debe seguir para minimizar el coste de la ruta recorrida?



Estrategia

1. Construir todos los ciclos
2. Calcular el peso de cada uno de los ciclos.
3. Elegir el ciclo de menor peso

¿Cuántas operaciones?
El número de ciclos es $\frac{(n-1)!}{2}$

Problema del viajante TSP

Un viajante de comercio desea visitar n ciudades volviendo al punto de partida. ¿Qué ruta debe seguir para minimizar el coste de la ruta recorrida?

Estrategia

¿Cuántas operaciones?
El número de ciclos es $\frac{(n-1)!}{2}$

1. Construir todos los ciclos → Un ciclo se construye con n – 1 asignaciones
2. Calcular el peso de cada uno de los ciclos. → El peso de un ciclo se calcula con n – 1 sumas
3. Elegir el ciclo de menor peso → El menor elemento de una lista de tamaño k se calcula con k – 1 comparaciones y k – 1 asignaciones

Problema del viajante TSP

Un viajante de comercio desea visitar n ciudades volviendo al punto de partida. ¿Qué ruta debe seguir para minimizar el coste de la ruta recorrida?

¿Cuántas operaciones?
El número de ciclos es $\frac{(n-1)!}{2}$

Coste de la estrategia

Paso 1) (n -1)$\frac{(n-1)!}{2}$ → Un ciclo se construye con n – 1 asignaciones

Paso 2) (n -1) $\frac{(n-1)!}{2}$ → El peso de un ciclo se calcula con n – 1 sumas

Paso 3) 2 $\frac{(n-1)!}{2}$ – 2 → El menor elemento de una lista de tamaño k se calcula con k – 1 comparaciones y k – 1 asignaciones

Problema del viajante TSP

Un viajante de comercio desea visitar n ciudades volviendo al punto de partida. ¿Qué ruta debe seguir para minimizar el coste de la ruta recorrida?

Coste de la estrategia

Paso 1) (n -1)$\frac{(n-1)!}{2}$

Paso 2) (n -1) $\frac{(n-1)!}{2}$

Paso 3) 2 $\frac{(n-1)!}{2}$ $-$ 2

Coste total g(n) = 2n $\frac{(n-1)!}{2}$ $-$ 2 = n! $-$ 2

Calcular número de aristas de un grafo con n vértices

$$T(n) = n^2 - 2n + 2$$

Problema del viajante

$$g(n) = n! - 2$$

Sólo tiene interés el estudio de la función coste cuando n es muy grande

Complejidad

	log n	**n**	**n^2**	**2^n**	**n!**
2	1	2	4	4	2
8	3	8	64	256	40320
32	5	32	1024	$4{,}3\times10^{9}$	$2{,}6\times10^{35}$
100	6	100	10^{4}	$1{,}2\times10^{27}$	$9{,}3\times10^{177}$

Un siglo tiene $3{,}1\times10^{9}$ segundos

Si la edad del Universo es de 13500 millones de años,
el Big-Bang ocurrió hace $4{,}5\times10^{16}$ segundos

Notación O(f)

f es cota superior asintótica de T

$T(n) \in O(f(n)) \Leftrightarrow \exists k \in N,\ \exists a \in R^+$, tales que $T(n) \leq af(n) \quad \forall n \geq k$

$T(n) = n^2 - 2n + 2$ $\qquad$ $T(n) \in O(n^2)$

O(f) es el conjunto de funciones que crecen no más rápido que f

Si $g \in O(f)$ entonces

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = c$$ con c constante (puede ser 0)

Notación $\Omega(f)$

f es cota inferior asintótica de T

$T(n) \in \Omega(f(n)) \Leftrightarrow \exists k \in N,\ \exists a \in R^+$, tales que $T(n) \geq af(n) \quad \forall n \geq k$

$\Omega(f)$ es el conjunto de funciones que crecen al menos tan rápido como f

Si $g \in \Omega(f)$ entonces

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$

con c constante (puede ser 0)

Notación $\Theta(f)$

f y g son asintóticamente equivalentes si

$g(n) \in \Omega(f(n))$ y $g(n) \in O(f(n))$

$\Theta(f)$ es el conjunto de funciones que crecen con la misma rapidez que f

Si $g \in \Theta(f)$ entonces

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$ con c constante distinta de 0

Órdenes de complejidad

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^s) \subset O(k^n) \subset O(n!) \subset O(n^n)$$

$s>2$ $k>1$

Búsqueda

- secuencial $O(n)$
- binaria (en lista ordenada) $O(\log n)$

Ordenación

- inserción $O(n^2)$
- selección $O(n^2)$
- mezclas $O(n \log n)$

DFS Búsqueda en profundidad

Complejidad
- $O(q + n)$

BFS Búsqueda en anchura

Complejidad
- $O(q + n)$

MST Algoritmo de Prim

Complejidad
- $O(n^3)$
- $O(n^2)$
- $O(q \log n)$
- $O(n)$ si el grafo es plano

MST Algoritmo de Kruskal

Complejidad
- $O(q \log n + n^2)$
- $O(q \log n)$

Complejidad de un problema

El problema de ordenación de los elementos de una lista tiene complejidad $\Omega(n \log n)$

Para justificar la afirmación anterior veremos que el número de comparaciones efectuadas por cualquier algoritmo de ordenación para ordenar una lista de n elementos es, al menos, nlogn

Un árbol de decisión representa las comparaciones y movimientos de datos en cualquier algoritmo de ordenación

Las hojas del árbol son los resultados posibles de la ordenación
El número de hojas es $n!$

Construimos un ejemplo de árbol para un algoritmo de ordenación, la ordenación por inserción

Ordenación por inserción (Insertion sort)

Ordenación por comparación de los elementos de una lista L.
En cada iteración un elemento se inserta en su lugar definitivo en una lista ya ordenada.
http://en.wikipedia.org/wiki/Insertion_sort

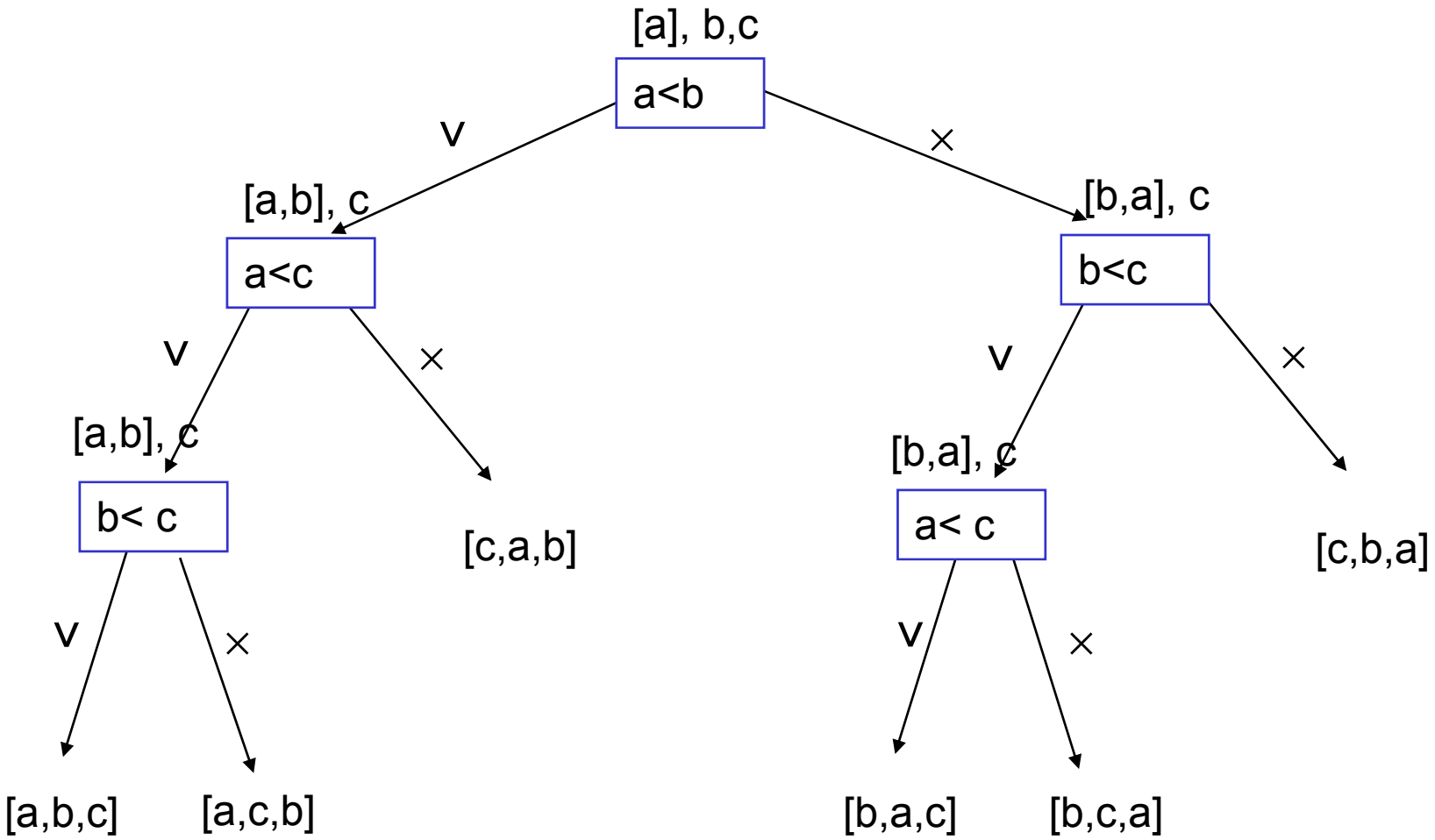
Inicialmente, paso 1, tenemos un elemento que ya está ordenado.
Paso (k). Tenemos una lista ordenada con los $k - 1$ primeros elementos de L.
El elemento k se inserta en su posición ordenada en la lista anterior, efectuando a lo sumo $k - 1$ comparaciones.

El número total de comparaciones efectuadas es, por tanto,

$$T(n) = 1 + 2 + 3 + \ldots + (n - 1) \in O(n^2)$$

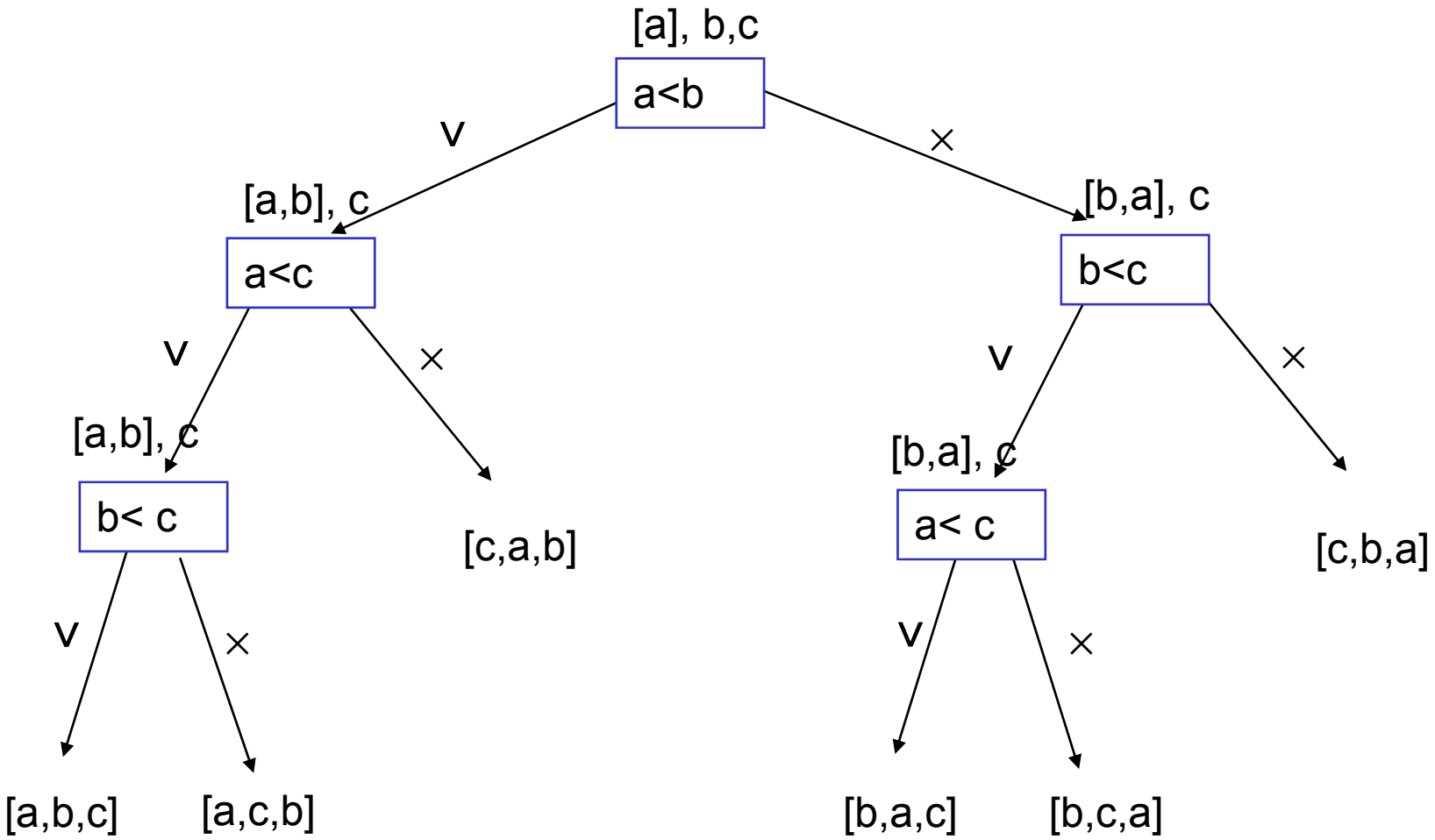
ÁRBOL DE DECISIÓN Y ALGORITMO DE ORDENACIÓN

El árbol de decisión que representa las comparaciones y movimientos de datos en el algoritmo de ordenación por inserción (lista de tamaño 3)



ÁRBOL DE DECISIÓN Y ALGORITMO DE ORDENACIÓN

Las hojas del árbol son todas las posibles ordenaciones, en total n!



Complejidad de un problema

Las hojas del árbol son todas las posibles ordenaciones, en total n!

La altura del árbol es el número de comparaciones que se deben realizar para ordenar una lista de tamaño n

Recordando la relación entre hojas y altura en un árbol binario,

$$|\text{comparaciones}| \geq \log_2 n!$$

Por tanto, el número T(n) de comparaciones realizadas por CUALQUIER algoritmo de ordenación es

$$T(n) \in \Omega(\log_2 n!)$$

$$\boxed{T(n) \in \Omega(n \log_2 n)}$$

El problema de ordenación tiene una cota inferior $\Omega(n \log_2 n)$

Complejidad de un problema

$\Omega(\log_2 n!) = \Omega(n \log_2 n)$

Dem.
(si n es par)
$n! = 1 \times 2 \times \ldots \times (n-1) \times n = (1 \times n) \times (2 \times (n-1)) \times \ldots \times (n/2) \times (n - n/2 + 1)$

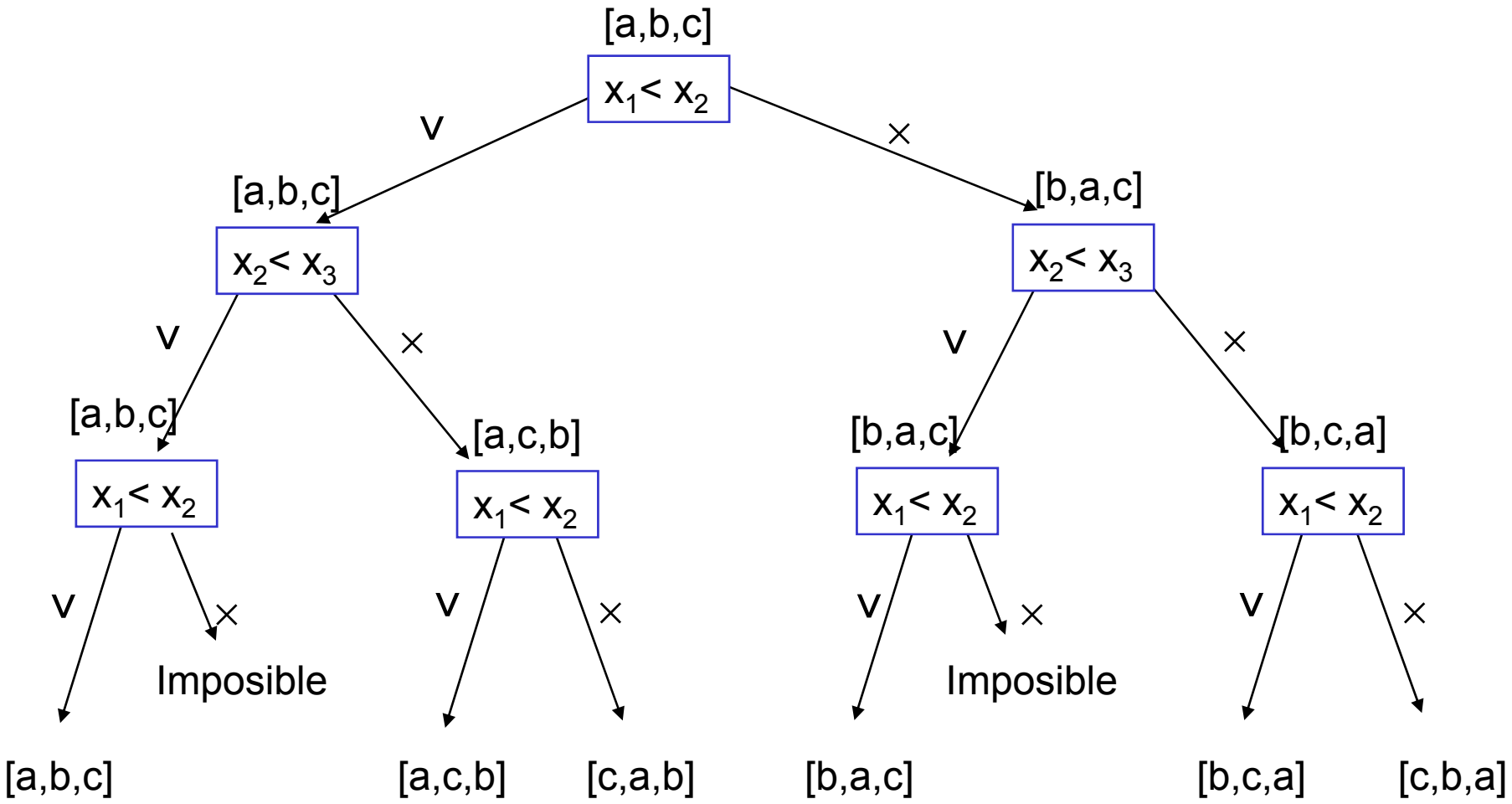
Cada factor de este producto es $\geq n$, porque si $1 \leq i \leq n$, entonces
$i\,(n - i + 1) - n = (i - 1)(n - i) \geq 0$

Luego, $n! \geq n^{\frac{1}{2}n}$ porque hay n/2 factores

Por tanto,
$$\frac{1}{2} n \log_2 n \leq \log_2 (n!) \leq \log_2 1 + \log_2 2 + \cdots + \log_2 n \leq n \log_2 n$$

ÁRBOL DE DECISIÓN Y ALGORITMO DE ORDENACIÓN

El árbol de decisión que representa las comparaciones y movimientos de datos en el algoritmo de ordenación por burbuja (lista de tamaño 3)



ÁRBOL DE DECISIÓN Y ALGORITMO DE ORDENACIÓN

Las hojas del árbol son todas las posibles ordenaciones, en total n!