El problema de ordenar

No podemos ordenar más rápido que $O(n^2)$ resolviendo solo una inversión a la vez

¿Cómo podemos resolver más de una inversión por paso?

El problema de ordenar

Debemos intercambiar elementos no adyacentes

¿Qué tan rápido puede ordenar un algoritmo haciendo esto?

El problema de ordenar

Definimos **problema** como una relación entre input y output

Una **instancia** de un problema es un input específico

¿Cómo podríamos definir formalmente el **problema de ordenar**?

El problema de ordenar

De la manera más general posible:

Input: Una secuencia $A = p(\mathcal{A})$

Output: Una permutación p^{-1} tal que $p^{-1}(A) = \mathcal{A}$

Es decir buscamos la permutación p^{-1} que “deshace” p

El problema de ordenar

¿Cuántas posibles soluciones tiene una instancia?

¿Qué pasa si hay elementos repetidos?

Una instancia sin elementos repetidos tiene una única solución.

Si hay elementos repetidos, significa que hay más de una posible solución. Esto hace que el problema sea **más fácil**.

Nos importa analizar que tan difícil es el problema, por lo que analizaremos el caso más difícil: sin elementos repetidos.

El problema de ordenar

Para una instancia, debemos encontrar la única permutación que la resuelve.

¿Cuántas permutaciones hay?

¿Cuántas posibles instancias hay?

El conjunto de instancias se define como:

$$P(\mathcal{A}) = \{\, p(\mathcal{A}) \quad | \quad p \text{ es una permutación} \,\}$$

Una solución para una instancia $p(\mathcal{A})$ es una permutación p^{-1} tal que $p^{-1}\big(p(\mathcal{A})\big) = \mathcal{A}$. Todas estas permutaciones forman el conjunto de soluciones:

$$S = \{\, p \quad | \quad p \text{ es una permutación} \,\}$$

El problema de ordenar

Hay $n!$ instancias y $n!$ permutaciones, y cada instancia tiene como solución una permutación en particular.

¿Es cada permutación solución de una única instancia?

La respuesta es si, pero debemos definir formalmente las permutaciones para ver por que. Una forma de hacerlo es usando matrices de permutación:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

La inversa de una matriz de permutación es su traspuesta

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Definiendo las permutaciones con matrices es fácil ver que

$$p^{-1}\big(p(\mathcal{A})\big) = p\big(p^{-1}(\mathcal{A})\big)$$

Ya que $p^{-1} \cdot p = I$ (la matriz identidad)

Como la inversa de una matriz es única, cada permutación es solución de una sola instancia.

El problema de ordenar

Tenemos entonces una correspondencia 1:1 entre el conjunto de instancias y el de soluciones.

Volviendo a la pregunta original,

¿En cuantos pasos podemos encontrar la solución haciendo intercambios?

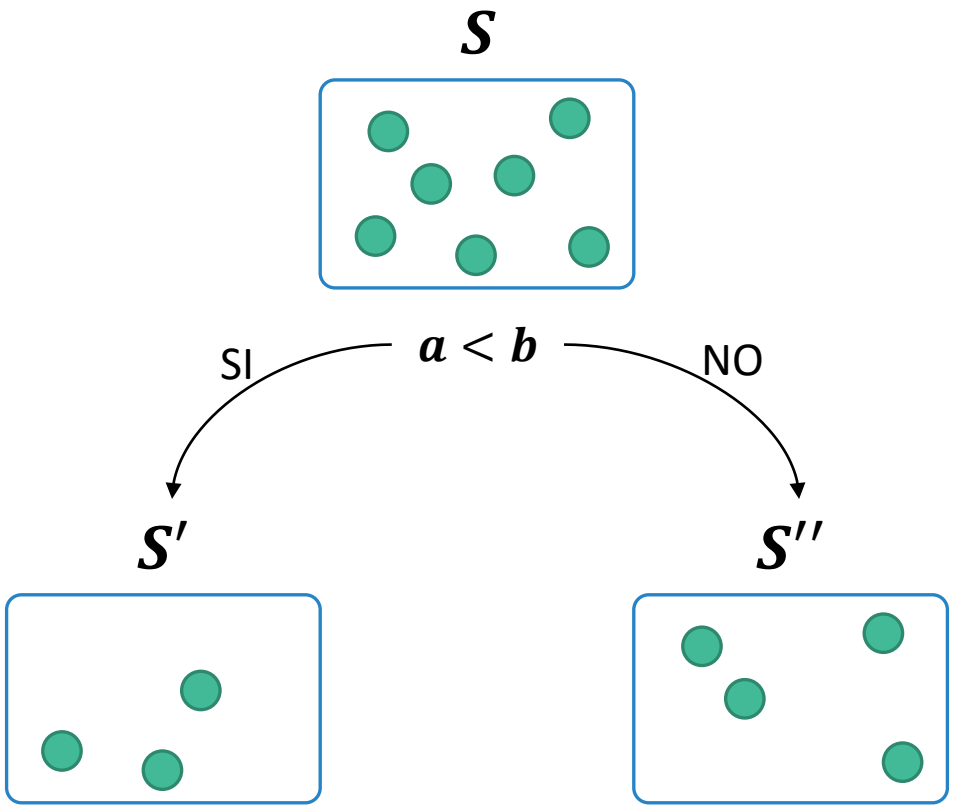
Lo que busca cualquier algoritmo de ordenación es encontrar la permutación solución para esa instancia.

Para esto debe ser capaz de descartar todas las otras permutaciones que no son solución.

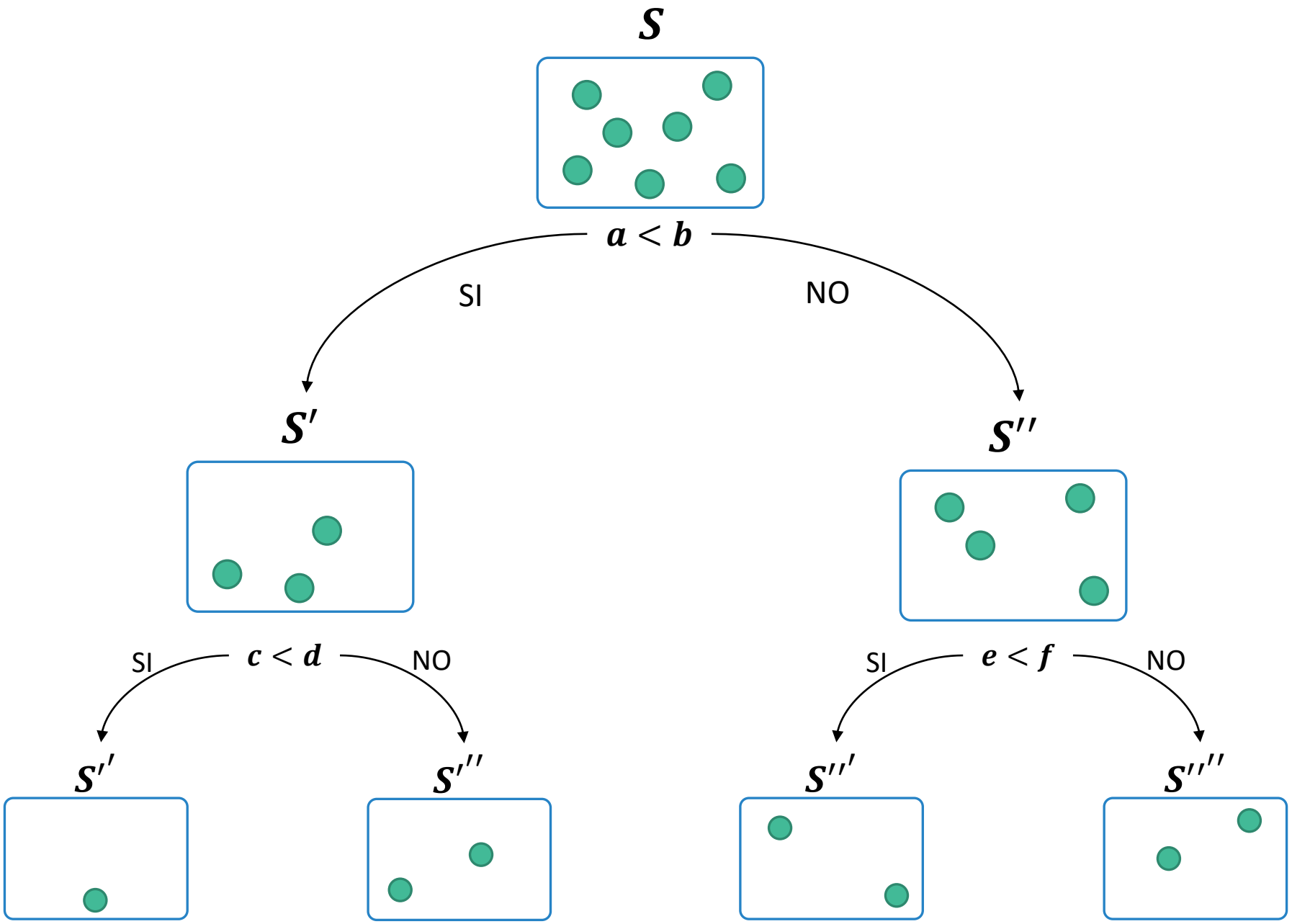
Los algoritmos que funcionan en base a comparación e intercambio cuentan con esta única herramienta para decidir que elementos de S son o no posibles soluciones.

Qué tan rápido se puede ordenar

¿Qué hace el algoritmo al hacer una comparación?



En cada comparación, podemos separar el conjunto de permutaciones en 2: las que sabemos que no son solución, y las que podrían serlo. En el mejor caso esta separación se hace de manera equitativa, pero no tiene por que serlo.


S
a < b
SI
NO
S′
S′′
c < d
SI
NO
e < f
SI
NO
S′′′
S′′′′
S′′′′′
S′′′′′′

Qué tan rápido se puede ordenar

Pero necesitamos que sea posible llegar a cada permutación para cada posible instancia.

Para esto el algoritmo debe realizar suficientes comparaciones de manera que cada hoja del árbol tenga una única permutación.

Si dividimos equitativamente en cada paso y ejecutamos h comparaciones, tenemos un árbol con 2^h hojas.

Para que cada permutación pueda tener su propia hoja, necesitamos tener al menos $n!$ hojas en el árbol. Para esto se debe cumplir que:

$$2^h > n!$$

$$h > \log_2 n!$$

Podemos acotar $n!$

$$n! = 1 \cdot 2 \cdot 3 \ldots \left(\frac{n}{2} - 1\right) \cdot \left(\frac{n}{2}\right) \cdot \left(\frac{n}{2} + 1\right) \ldots (n-1) \cdot n$$

$$n! < n \cdot n \cdot n \ldots\ n \cdot n \cdot n \ldots\ n \cdot n = n^n$$

Por lo tanto

$$\log_2 n! < n \cdot \log_2 n$$

$$n! = 1 \cdot 2 \cdot 3 \dots \left(\frac{n}{2} - 1\right) \cdot \left(\frac{n}{2}\right) \cdot \left(\frac{n}{2} + 1\right) \dots (n-1) \cdot n$$

$$n! > 1 \cdot 1 \cdot 1 \dots 1 \cdot \frac{n}{2} \cdot \frac{n}{2} \dots \frac{n}{2} \cdot \frac{n}{2} = \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

Por lo tanto

$$\log_2 n! > \frac{n}{2} \cdot \log_2 \frac{n}{2}$$

Cota inferior de ordenación

$$\log_2 n! \in \Theta(n \cdot \log n)$$

Entonces

$$h \in \Omega(n \cdot \log n)$$

Esto significa que:

Cualquier algoritmo de ordenación por comparación debe ejecutar como mínimo $n \cdot \log n$ comparaciones en su peor caso.