

2015年9月21日 (月)

AUX # 2

[P1] Adversario.

- Idea: construiremos un adversario que mantiene 2 grafos, C y A.
 - ambos serán consistentes con las respuestas dadas al algoritmo
 - C tiene un ciclo
 - A no tiene ciclo.

Estas props se mantienen ante \leq^* (2) consultas

Adv:

$$C \leftarrow K_n \quad // \text{clique de } n \text{ nodos}$$

$$A \leftarrow \emptyset \quad // n \text{ nodos, } 0 \text{ arcos.}$$

adyacente (u, v):

Si $A + (u, v)$ es acíclico

$$A \leftarrow A + (u, v)$$

return si

else

$$C \leftarrow C - (u, v)$$

return no.

Veremos varias props de A y C para demostrar que C tiene 1 ciclo

a) $A \subseteq C$

b) $C - A$ es el conjunto de los arcos no preguntados

(todo arco preguntado fue agregado a A o fue quitado de C)

c) Si A es desconexo, C tiene todos los arcos que conectan las componentes conexas de A

(B) ALGORITMO

S # XVA

- Sea un arco de este tipo. Ese arco no puede crear un ciclo en A, luego no puede haber sido quitado de C.

d) C es conexo.

Dem: Por contradicción. Supongamos en un paso se quita el arco e y C queda desconexo. Luego A

• sería cíclico de tener e.

Pero C queda desconexo \Rightarrow e no es parte de un ciclo en C

Pero por a), $A \subseteq C$

e) Si $A \neq C$, C tiene un ciclo.

$\Rightarrow \Leftarrow$ Luego C es conexo.

e) Si $A \neq C$, C tiene un ciclo.

Dem: Por contradicción sea C acíclico.

• Es conexo por d) \Rightarrow es árbol.

• Como $A \neq C$ y $A \subseteq C \Rightarrow A$ es desconexo.

• Sea $e \in G(C-A)$

- e conecta 2 componentes de A.

- como C es árbol, e debe ser el único que conecta estas 2 componentes.

- Para $n \geq 3$, en K_n existe al menos un e' que las une, $e \neq e'$.

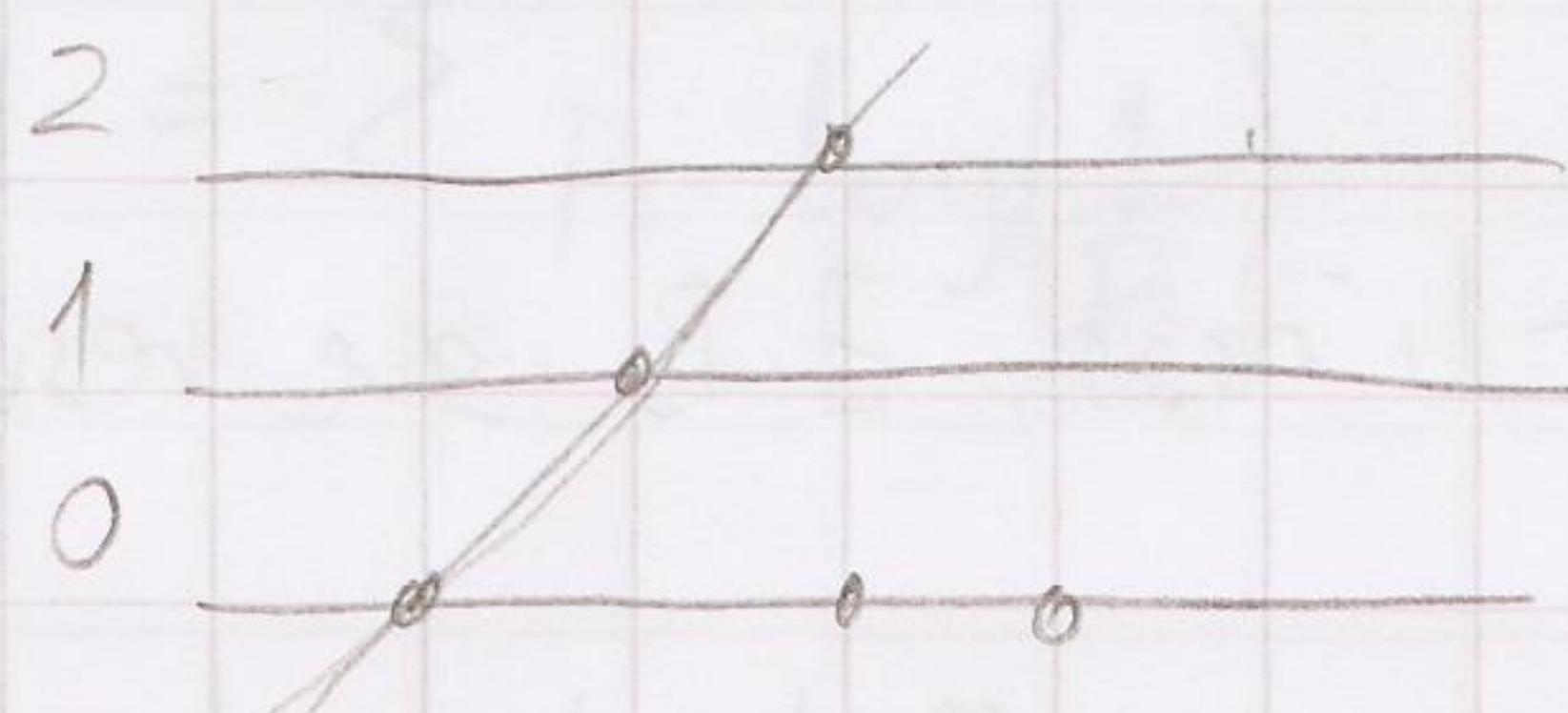
Por b), $e' \in C-A \Rightarrow \Leftarrow$

\Rightarrow Si $A \neq C$, C tiene ciclo.

* La gracia es que el adv. da las mismas respuestas para 2 instancias \neq 's del problema. El algoritmo no sabe en cuál de los 2 está si no hasta hacer $\binom{n}{2}$ preguntas.

[P2]

a)



No es trivial modelar

b) Reduciremos 3-colineal a esto. (iba, cambiar palabra "recta" por "recta tg")

- Tenemos una instancia de 3-colineal: n puntos, buscamos 1 recta tg

$$y_1 = ax_1 + b \quad (x_i, y_i) \rightarrow (a, b)$$

$$y_2 = ax_2 + b \quad | \quad |$$

$$y_3 = ax_3 + b$$

• buscamos convertir puntos en líneas y viceversa

• Definimos la i-ésima recta como:

$$y = (x_i) x + (-y_i)$$

$$(a_i, b_i) \rightarrow (x, y)$$

Supongamos un alg encuentra (o determina que #) solución (x^*, y^*)

$(\exists \text{ pto } (x^*, y^*))$

$$\begin{aligned} & (\exists \text{ 3 rectas}) (0) \quad y^* = a_1 x^* + b_1 \quad (1) \quad y^* = x_1 x^* - y_1 \\ & \Leftrightarrow y^* = a_2 x^* + b_2 \quad \Leftrightarrow \quad y^* = x_2 x^* - y_2 \\ & \quad y^* = a_3 x^* + b_3 \quad \quad \quad y^* = x_3 x^* - y_3 \end{aligned}$$

$$\begin{aligned} & (2) \quad \Leftrightarrow y_1 = x^* x_1 - y^* \quad (3) \quad \Leftrightarrow \\ & \quad y_2 = x^* x_2 - y^* \\ & \quad y_3 = x^* x_3 - y^* \end{aligned}$$

los puntos

(x_1, y_1)

(x_2, y_2)

(x_3, y_3)

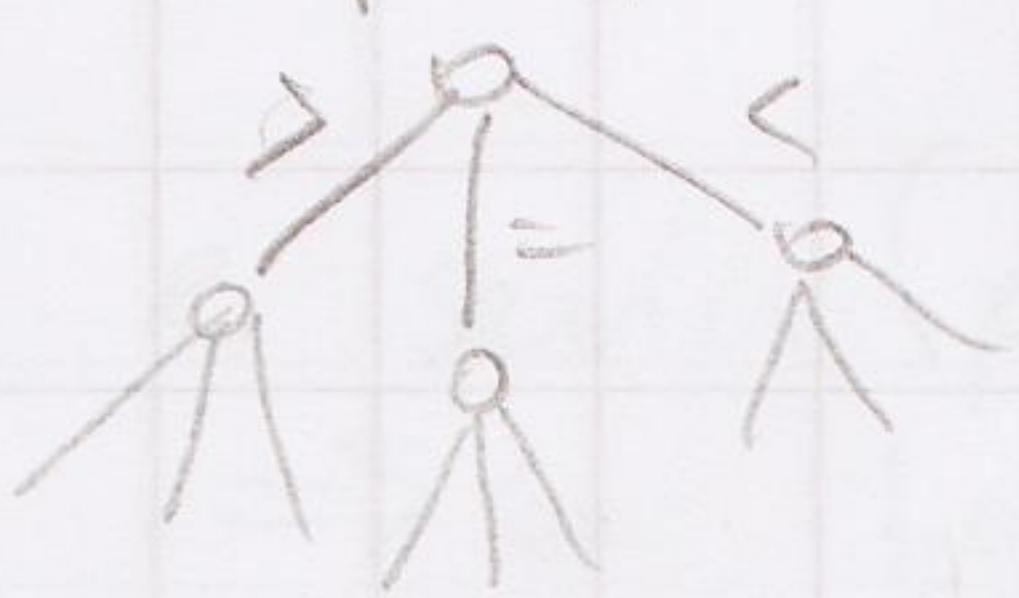
son colineales.

* Hay que verificar que la reducción tome $\Theta(n^{2-\epsilon})$, $\epsilon > 0$
 Bueno, el mapeo es lineal ($y \mapsto 1$)

Luego para cualquier A que resuelva 3-líneas, $\exists B$ que resuelve 3 colineal, tq: \sim reducción

$$T(B) = T(A) + \Theta(n)$$

P3 Todo algoritmo correcto debe tener al menos tantas hojas
 (a) Cómo resp. posibles.



En este caso, los n elem. de A deben estar en las hojas.
 \Rightarrow al menos n hojas.

- Cada nodo interno produce a lo más 2 nodos internos
 \Rightarrow hay a lo más 2^k nodos internos en el piso k .
 \Rightarrow hay a lo más $\sum_{i=0}^{k-1} 2^i$ nodos internos en los 1's k pisos
 $(= 2^k - 1)$

- Cómo debe haber al menos n hojas.

$$2^k - 1 \geq n \Rightarrow k \geq \log_2(n+1) \quad (\text{por caso})$$

Supongamos que solo busco los elem. que estén
 \Rightarrow la rta me da un código para el elemento.
 $A[i] > < >> < >>$ \Rightarrow puedo usar 1 bit por signo
 $\Rightarrow k_i$ bits para cada elem

$$\text{Pero } H = \sum p_i \log_2 \frac{1}{p_i}$$

Suponiendo una entrada uniforme, $p_i = \frac{1}{n} \Rightarrow H = \log_2 n$
 $\Rightarrow k_i \geq \log_2 n$ es el caso promedio.

2015年9月22日 (K)

Búsqueda Binaria con probabilidades de acceso.

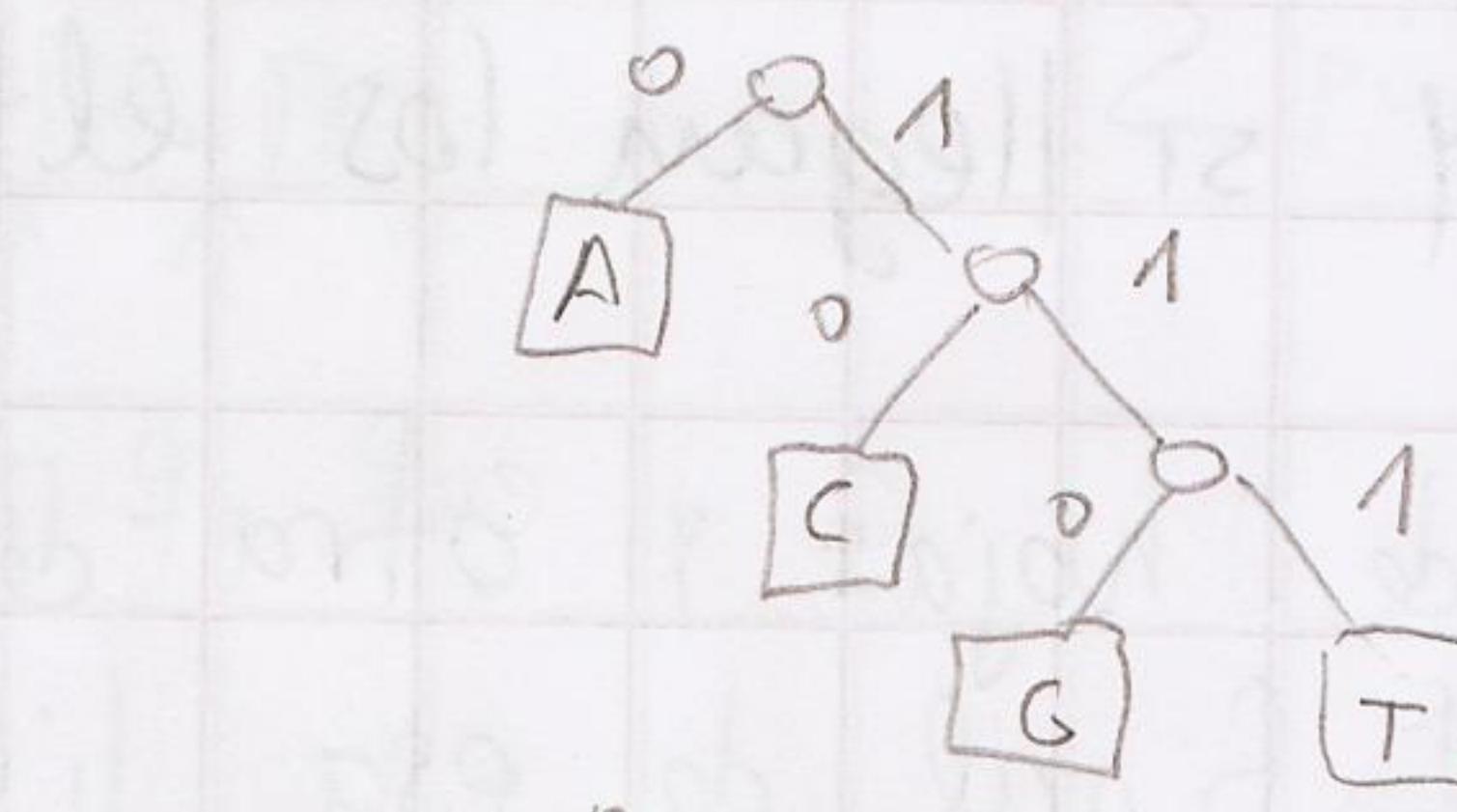
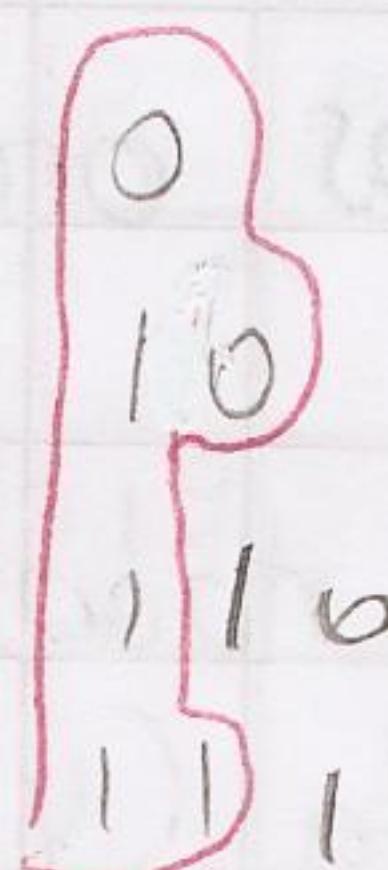
$$H = \sum p_i \log\left(\frac{1}{p_i}\right)$$

mín número de bits por símbolo que puede usar una codificación si las prob. de los símbolos son p_1, \dots, p_n

- Algoritmo de Huffman

A	00	0.5
C	01	0.25
G	10	0.125
T	11	0.125

induce a errores. Un código no puede ser prefijo de otro.



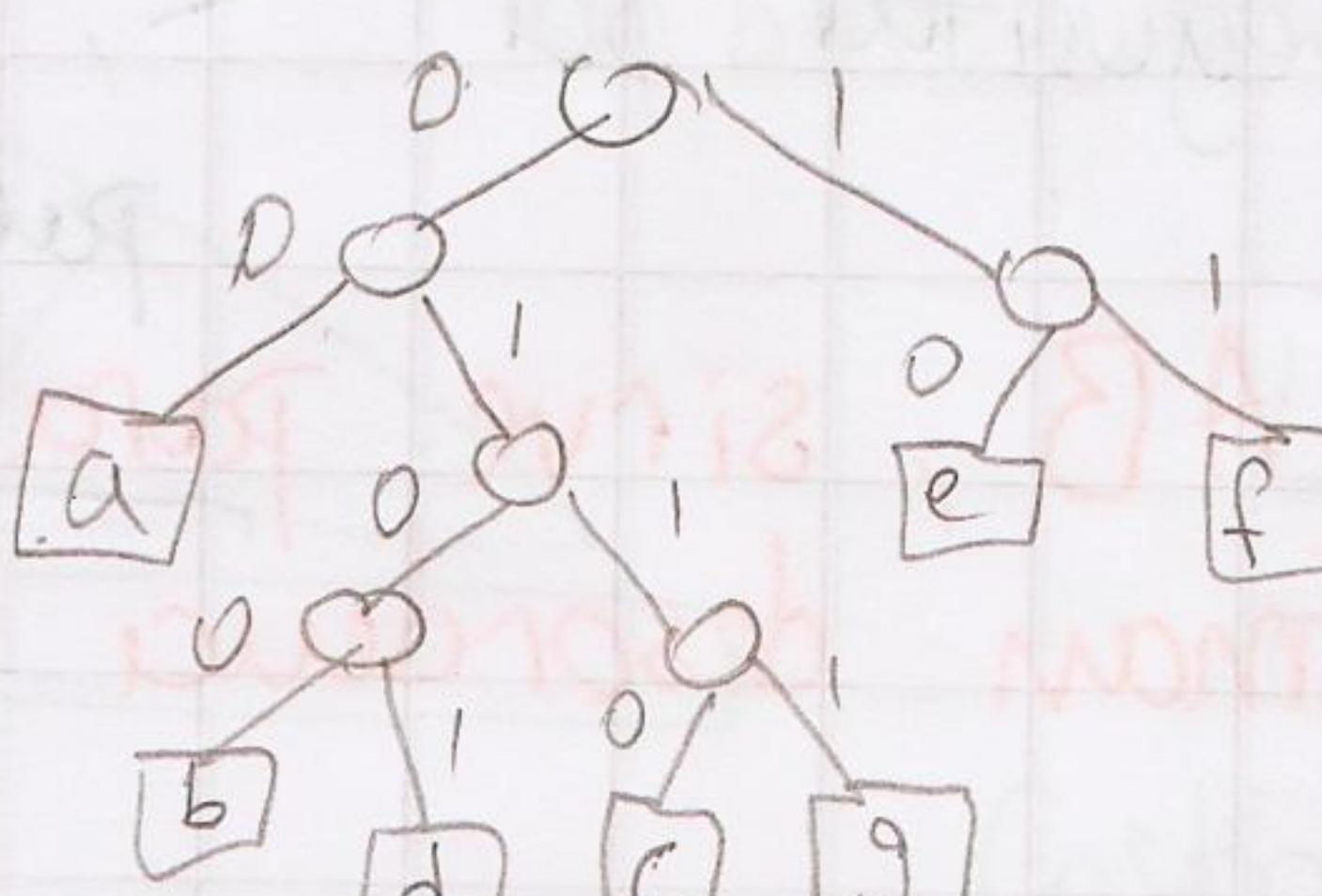
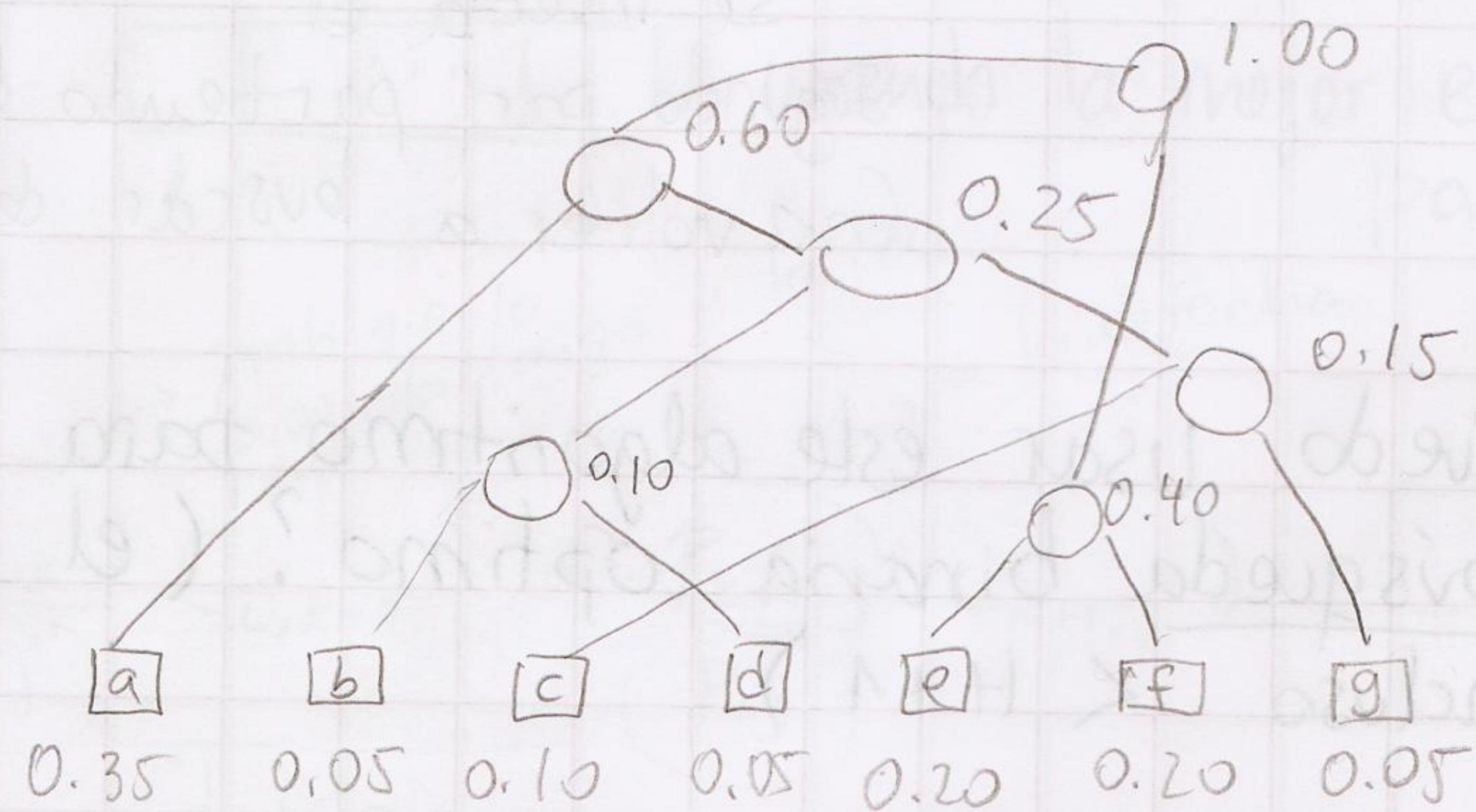
$$H \leq \left| \sum_{i=1}^n p_i l_i \right| < H + 1$$

largo promedio de un código

$(\left(\sum_{i=1}^n p_i l_i \right) \cdot n = \text{largo total del texto})$

$$\begin{aligned} 00 &= P(a) = 0.35 \\ 0100 &= P(b) = 0.05 \\ 0110 &= P(c) = 0.10 \\ 0101 &= P(d) = 0.05 \\ 10 &= P(e) = 0.20 \\ 11 &= P(f) = 0.20 \\ 0111 &= P(g) = 0.05 \end{aligned}$$

Árbol de peso mínimo



¿En qué tiempo se construye el árbol de peso mínimo?

Heapify: armas un heap, luego sacas los 2 mínimos, los uno, los sumo, y el nuevo nodo lo reinserto en el heap.

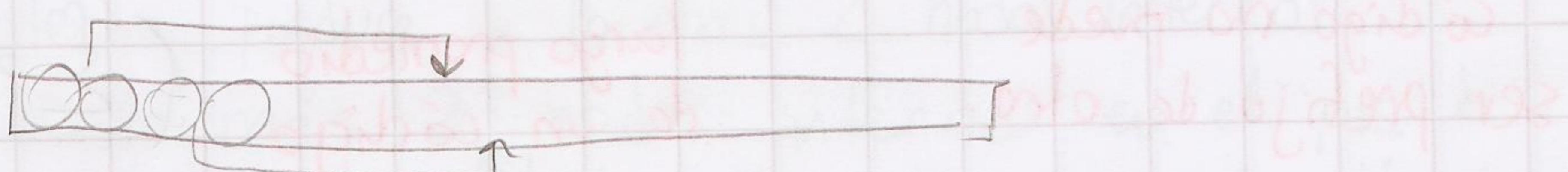
$\Theta(n \log n)$

¿Otra manera? y si llegan los ellos ordenados?

- tener lista de hojas y otra de nodos, e ir agregando los nodos al final de esa lista (ya queda ordenado)

$\Theta(n)$

- En una sola lista



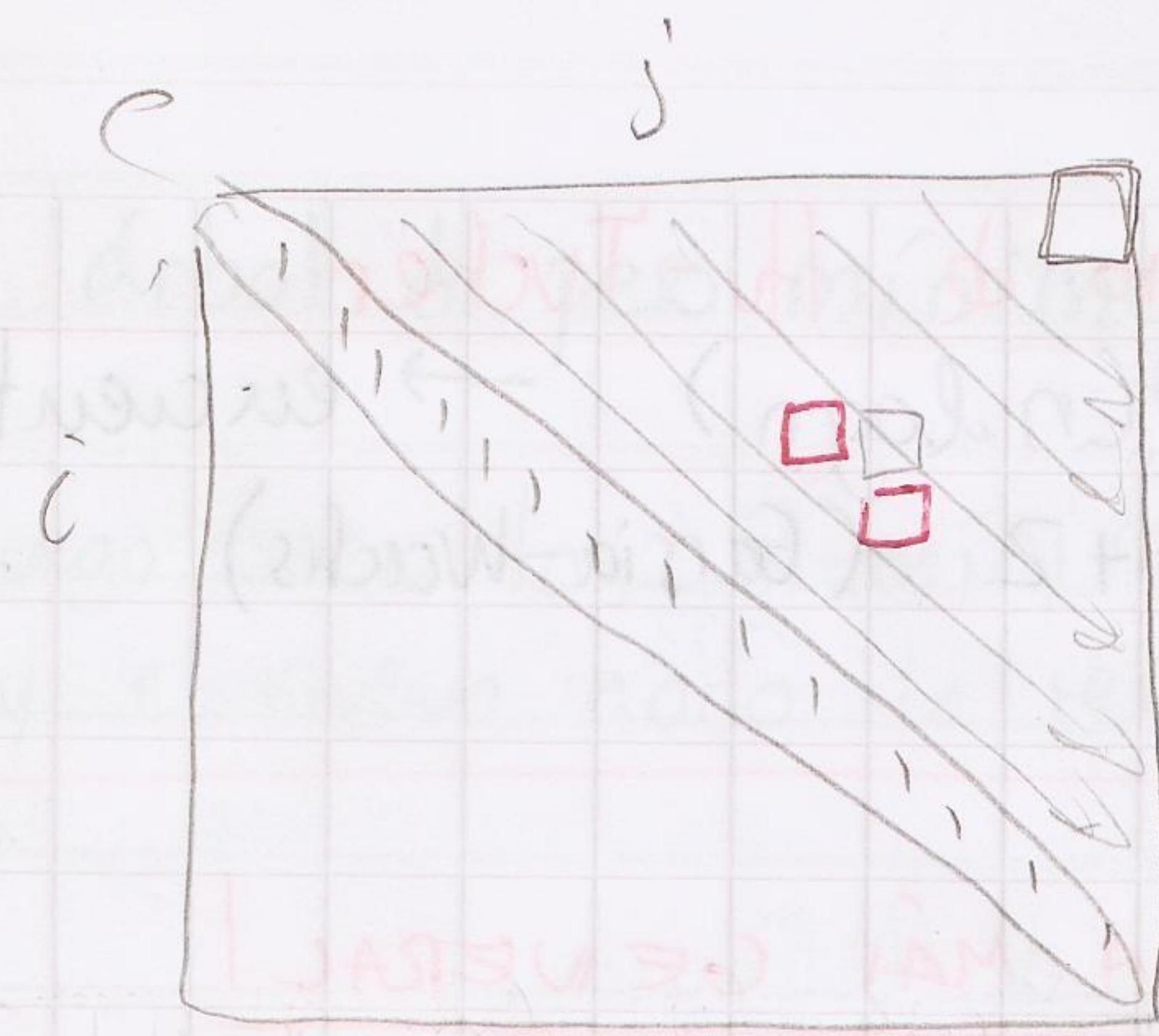
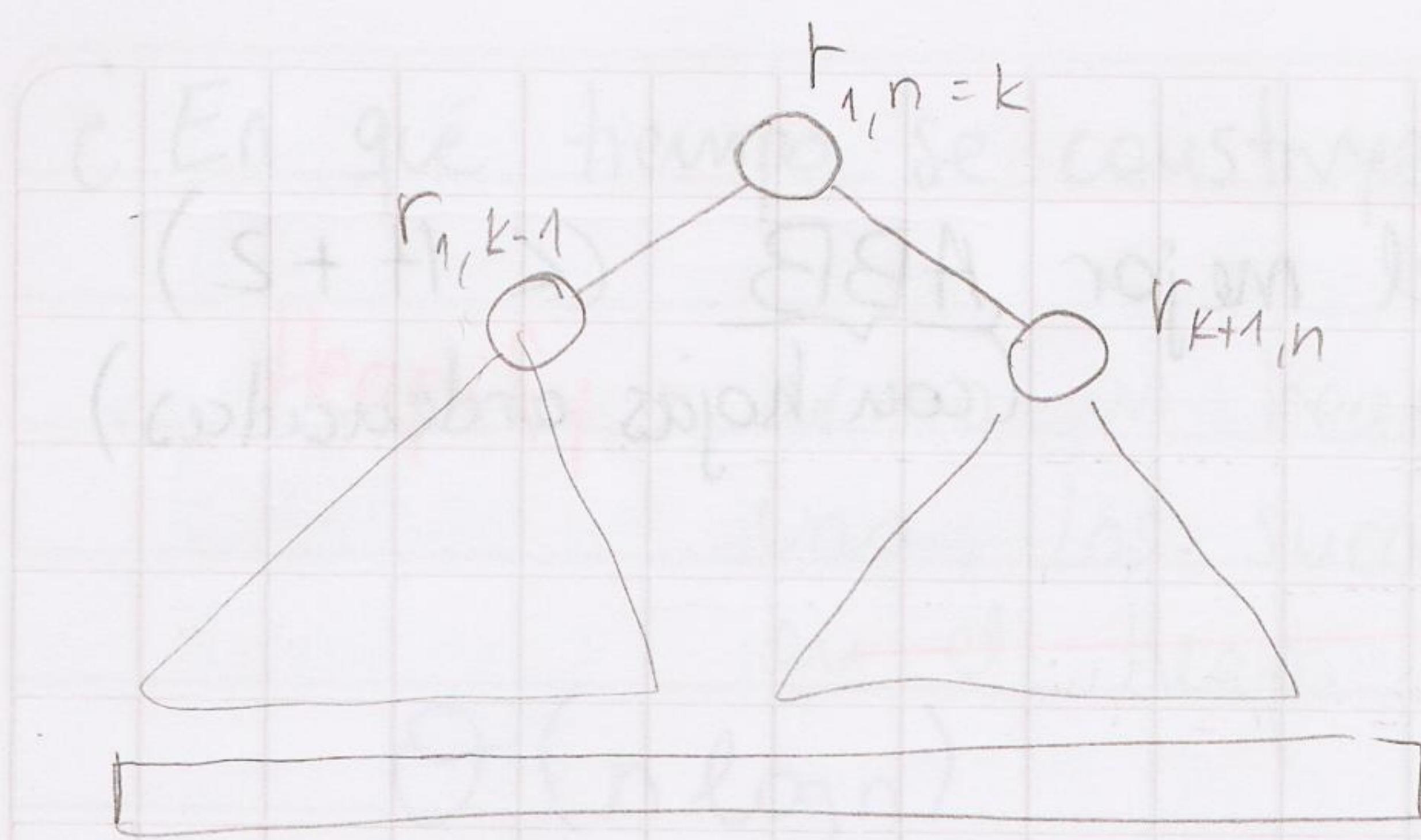
se inserta el segundo par partiendo de la última inserción!
(no volver a buscar del comienzo)

¿Puedo usar este algoritmo para crear un árbol de búsqueda binaria óptima? (el costo esperado sería mínimo, incluso $< H+1$)

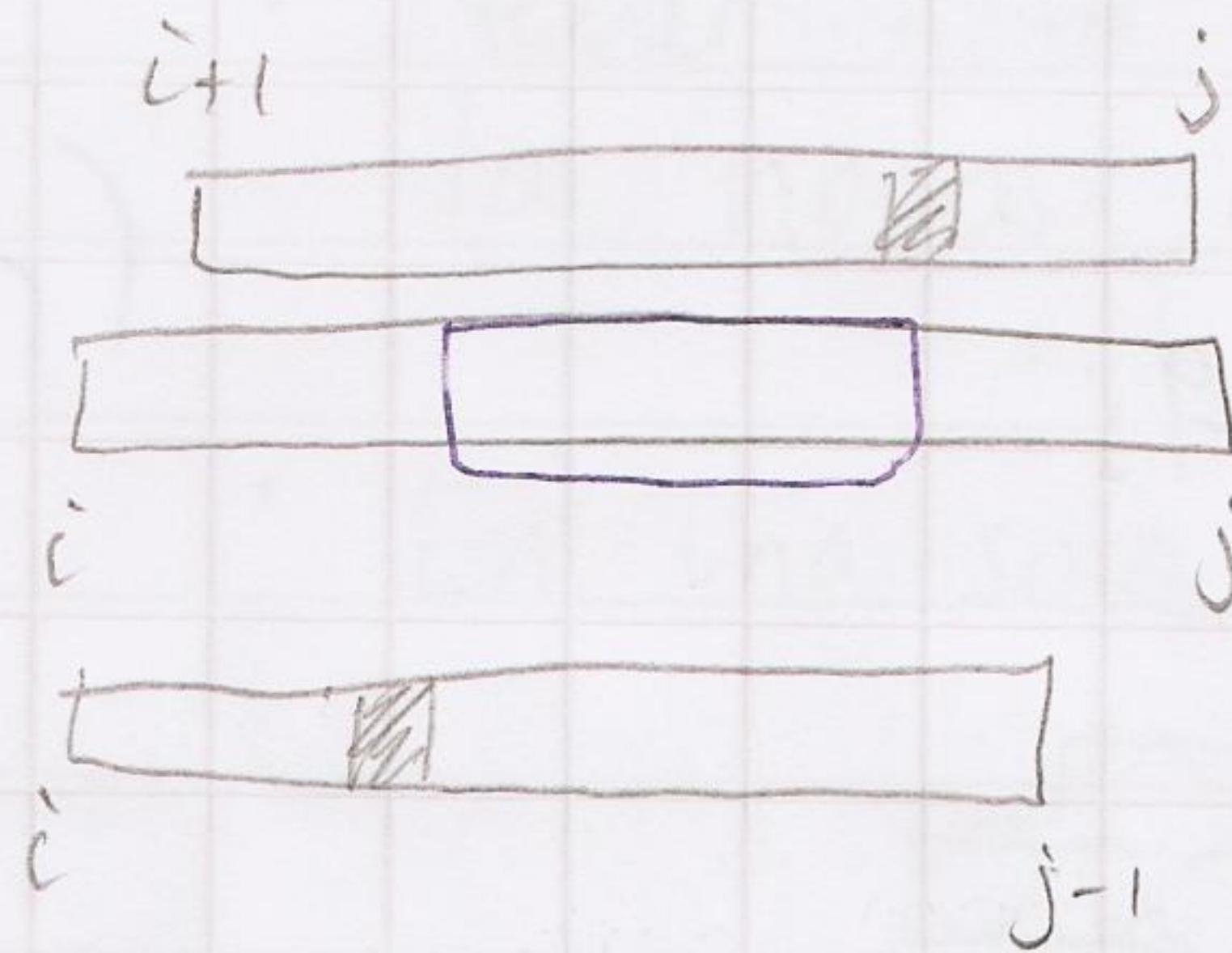
¿Qué hago? no puedo preguntar por $<, =, >$.

No es un ABB! es AB, sirve para codificar, pero no para buscar! Huffman desordena las hojas.

Entonces cómo puedo construir el mejor ABB con un algoritmo que no ordene las hojas?



¿y $O(n^2)$?



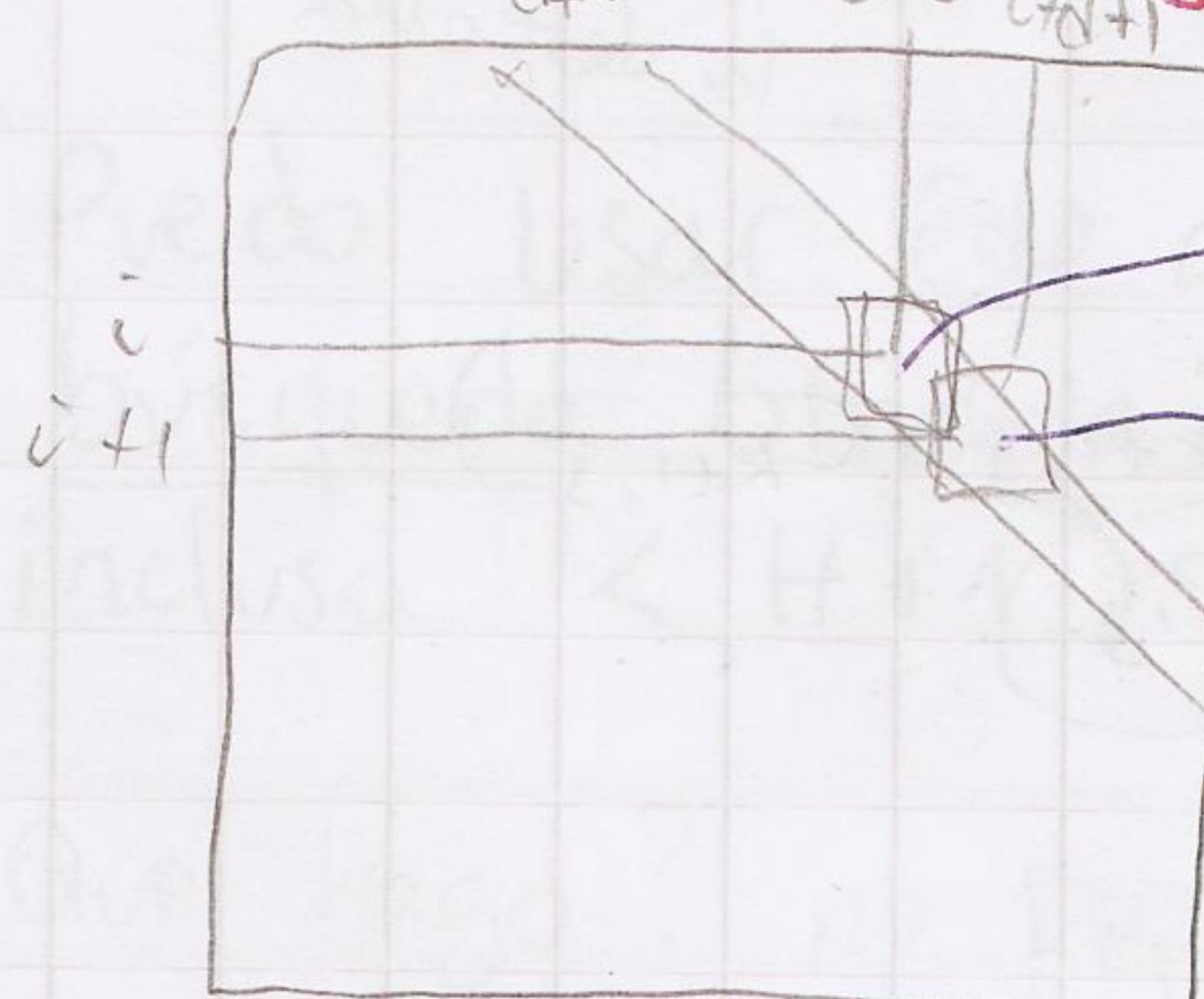
con programación dinámica, el trabajo es rellenar esa matriz

$$\sum_{i=1}^n \sum_{j=i}^n j - i + 1 = \frac{n^3}{6} = \Theta(n^3)$$

vas
de j a i
buscando
el mínimo

$$r_{i,j-1} \leq r_{ij} \leq r_{i+1,j}$$

Ahora $\rightarrow C_{ij} = 1 + \min \left(\frac{P_{i,k-1}}{P_{ij}} C_{i,k-1} + \frac{P_{k+1,j}}{P_{ij}} C_{k+1,j} \right)$



: TELESCÓPICA

$$r_{i+1,j} - r_{i,j-1} + 1$$

Sobrevive
algún
 $r \leq n$

$$= O(n) + n \approx 2n$$

costo
del algoritmo

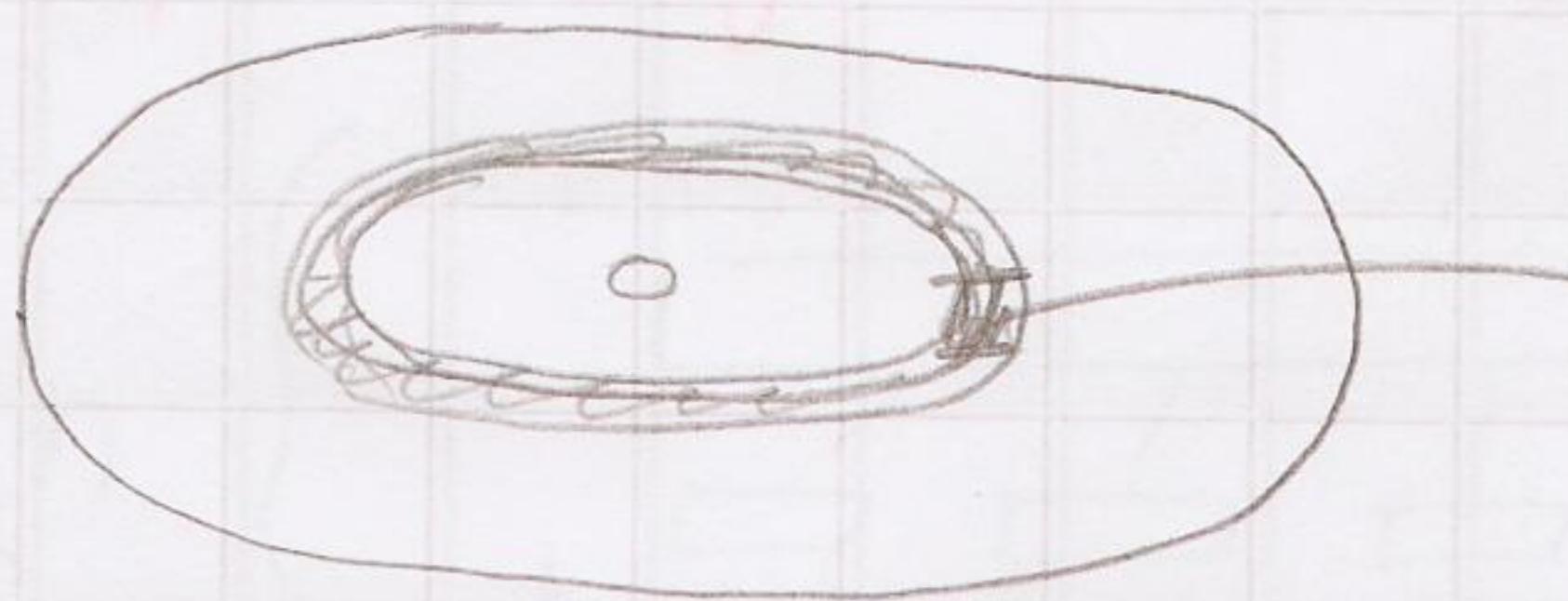
por diagonal

y son n diagonales
 $\Rightarrow O(n^2)$

Y si en vez de probabilidades & acceso
tengo COSTO de acceso?

Memoria Secundaria

2015年9月24日(木)



tiempo:

- seek → mover puntero a pista
- latencia → tiempo para que el bloque pase de ^{bajo del} cabezal.
- transferencia.

Para algoritmos en mem secundaria:

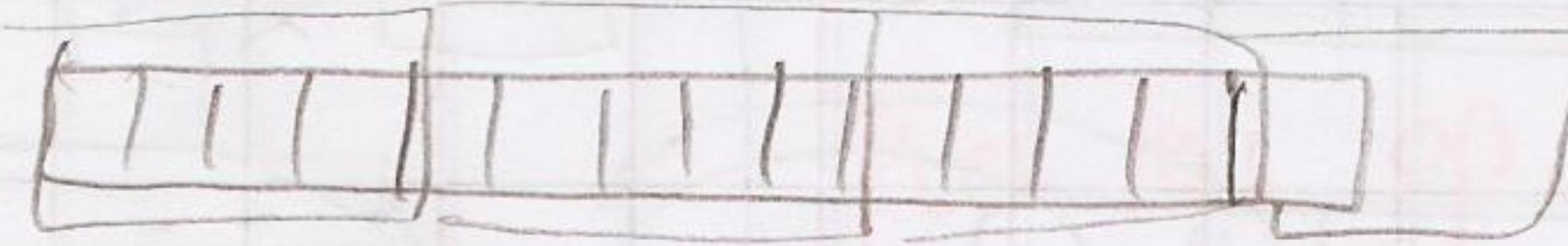
- minimizar los accesos a disco
- aprovechar localidad de bloques y entre bloques
(pasadas secuenciales)

I/O model: costo = # bloques leídos y escritos (no toma en cuenta que un alg. secuencial es mejor que uno saltado.)
Tampoco toma en cuenta trabajo en CPU.)

- n = tamaño del input
- B = n de palabras que caben en un bloque de mem secundaria
- M = n de palabras que caben en RAM.

El costo en I/O model se puede definir/medir en función de n, B y M.

Ej: Si leo todos los n enteros de un arreglo en disco, el costo es,
 $\Theta(\frac{n}{B})$ secuencial
 $\Theta(n)$ al azar.

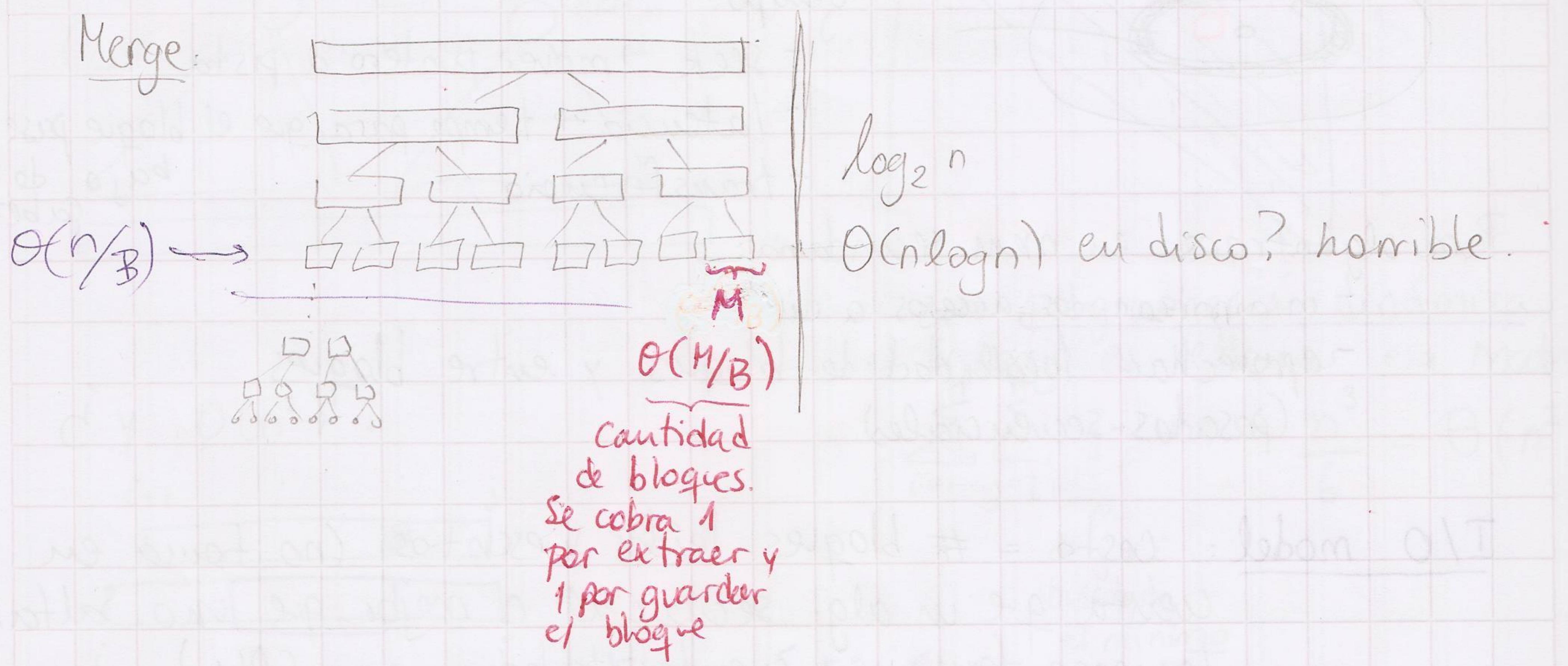


$$\Theta(\frac{n}{B})$$

Proprietary Material

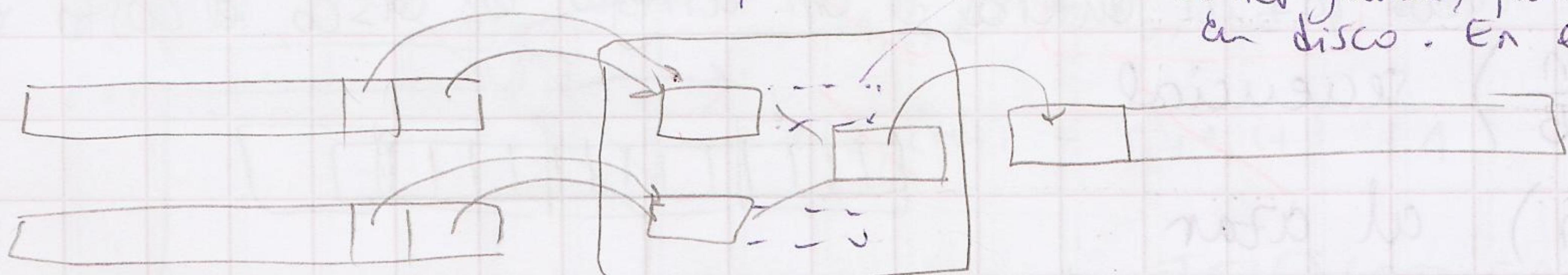
Ordenar en Disco:

Merge:



El algoritmo deja de dividir hasta que los pedazos son de tamaño M . Luego, en ese piso se hacen $\Theta(\frac{n}{B})$ accesos.

Costo total: $\Theta\left(\frac{n}{B} \cdot \log \frac{n}{M}\right)$

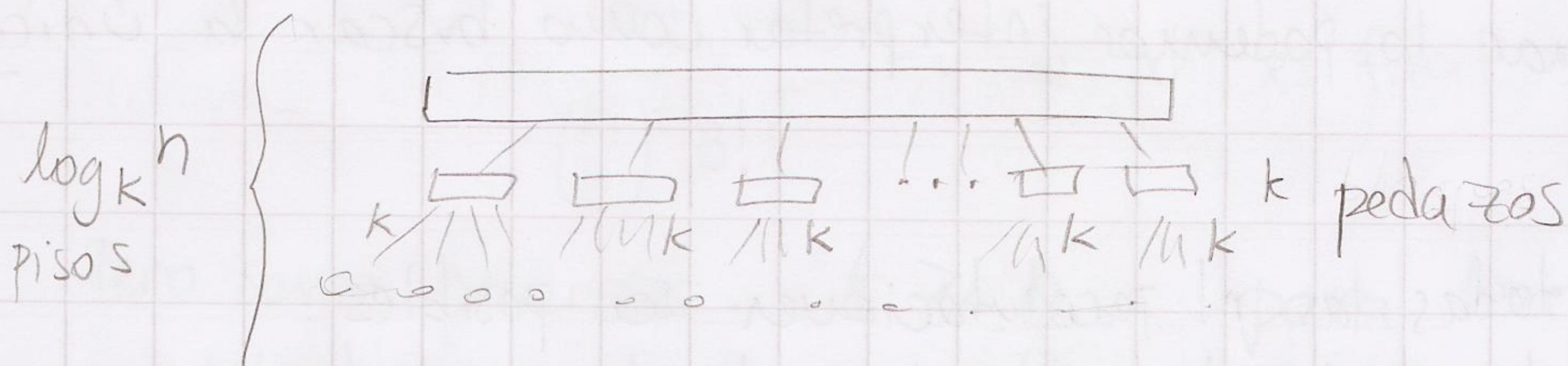


en la vida real es mejor tener un buffer grande, para leer secuencialmente en disco. En el modelo I/O da lo mismo

RAM
el merge es
asíncrono, luego
se lleva el bloque
de atrap en
cualquier momento.

¿se puede hacer mejor?

¿Por qué merge divide de a 2? ¿Si dividimos de a más?



$$\Theta(n(\log_k n)(k))$$

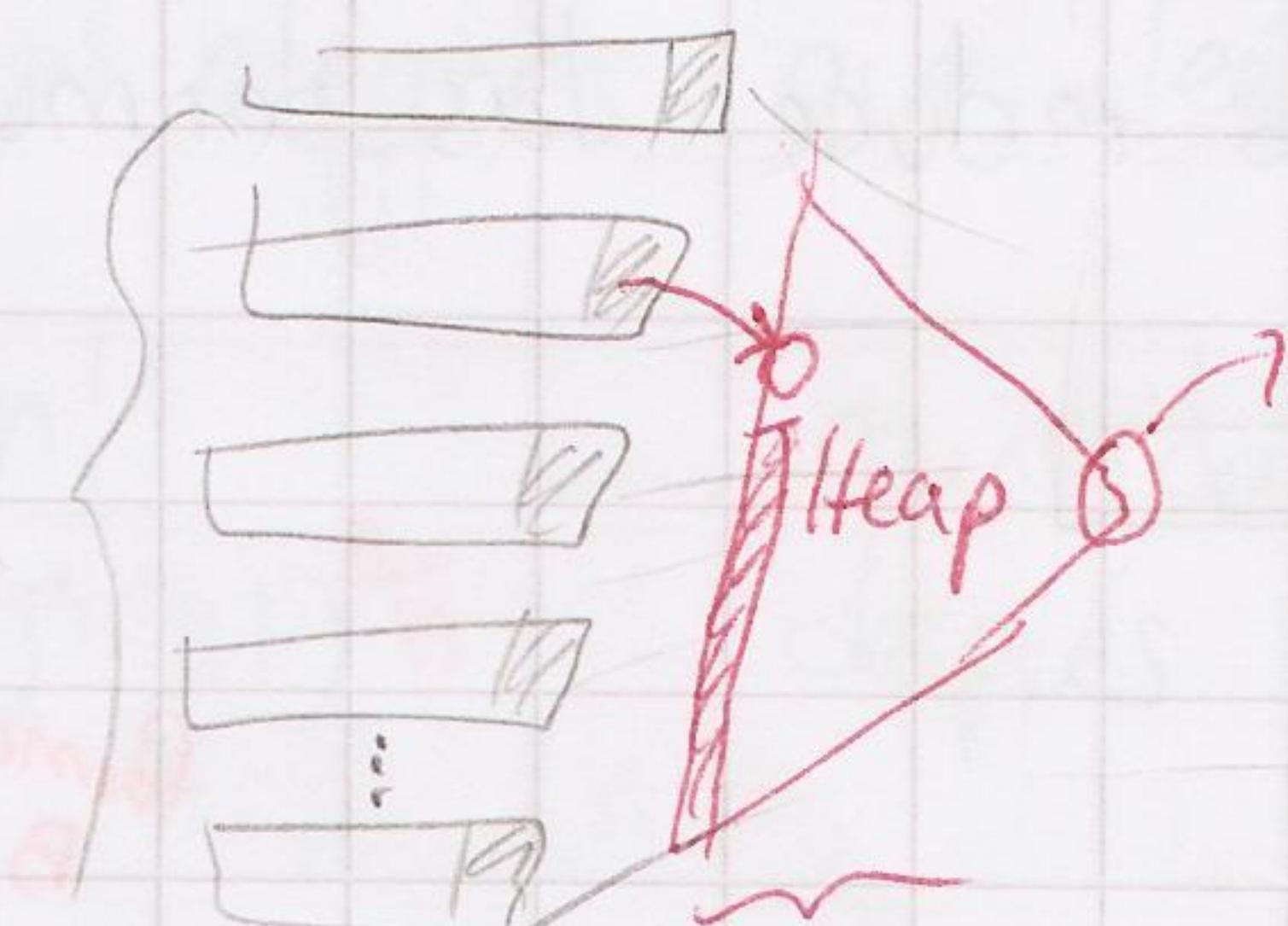
esencialmente para elegir min k

$$= \Theta(n(\log_k n) \log_2 k)$$

$$= \Theta(n \log_2 n)$$

:

no es mejor...?



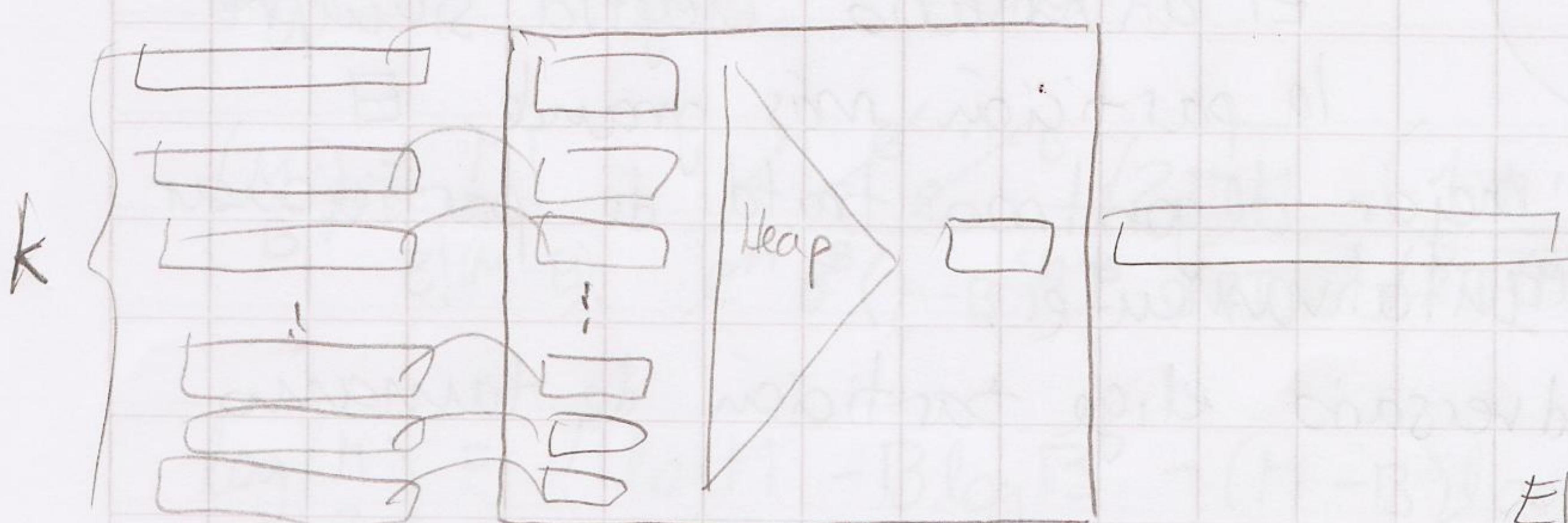
de tamaño k

Se extrae el min y se inserta esto al heap del mismo arreglo del cual se extrajo min
 $\Rightarrow \Theta(\log_2 k)$

¿Cómo extraigo el mínimo de k eficientemente?

\rightarrow Usar heap

Sirve este merge para el modelo I/O de ordenamiento en disco?



$$\Rightarrow \Theta\left(\frac{n}{B} \log\left(\frac{n}{m}\right)\right)$$

pues las op. de heap se hacen en memoria

\Rightarrow costo 0.

El buen k es $\leq \frac{M}{B}$ (lo máximos de bloques que caben en memoria)

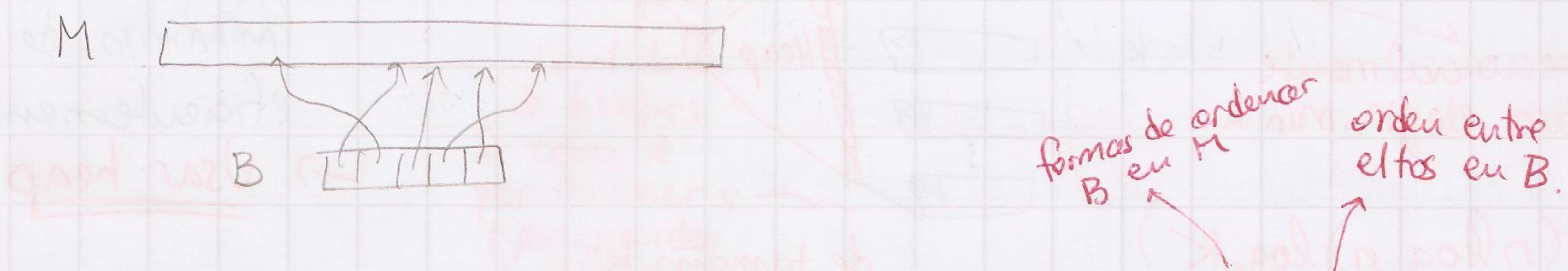
$$\Rightarrow \Theta\left(\frac{n}{B} \log_{\frac{M}{B}}\left(\frac{n}{m}\right)\right)$$

Pero ese k óptimo en la vida real no b es tanto, pues conviene tener buffers más grandes.

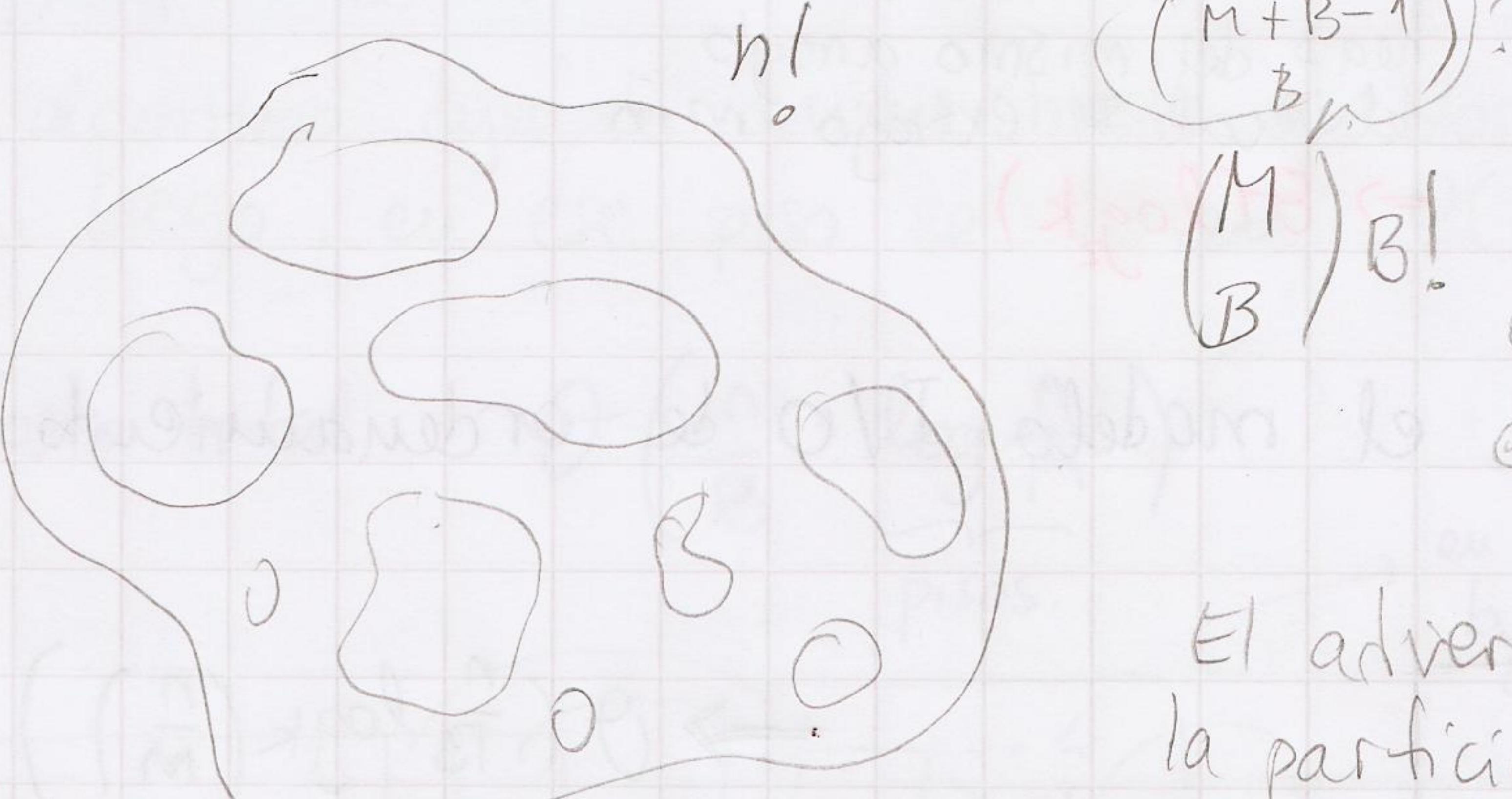
Cota inferior para ordenar en disco

Recordar que ordenar lo podemos interpretar como buscar la única permutación.

- Inicialmente, todas las $n!$ permutaciones son posibles.
- El algoritmo lee el input y va descartando permutaciones.
- Se puede terminar cuando reduce las permutaciones a 1.



En este procedimiento, el proceso puede resultar en $\binom{M}{B} B!$ casos



$$\binom{M+B-1}{B-1}$$

$\binom{M}{B} B!$ es la cantidad de clases en las que se particiona el conjunto de $n!$ perms.

El adversario elegirá siempre la partición más grande. El mejor algoritmo trata de particionar equitativamente.

En cualquier caso, el adversario elige partición de tamaño al menos $\frac{n!}{\binom{M}{B} B!}$

$$\binom{M}{B} B!$$

$$\frac{n!}{\binom{M}{B} B!}$$

Aux #3

Luego de t lecturas de bloques, el tamaño del conjunto es

$$\frac{n!}{(M)^t (B!)^{\frac{t}{B}}} \quad y \text{ de ahí es fácil sacar } t.$$

Pero somos generosos. $t \geq \frac{n!}{B}$ para leer todo.

En tal caso, hay bloques que leemos más de una vez.

Ahora $\frac{n!}{(M)^t (B!)^{\frac{n!}{B}}}$

cantidad de permutaciones admisibles después de leer t veces.

$$\frac{n!}{(M)^t (B!)^{\frac{n!}{B}}} \leq 1 \Rightarrow \frac{n!}{(B!)^{\frac{n!}{B}}} \leq \left(\frac{M}{B}\right)^t$$

$$\Rightarrow \log n! - \left(\frac{n!}{B}\right) \log B! \leq t \log \left(\frac{M}{B}\right)$$

$$n! = \frac{n^n}{e^n} \sqrt{2\pi n} \left(1 + O\left(\frac{1}{n}\right)\right)$$

$$\log n! = n \log n - O(n)$$

$$\left(\frac{M}{B}\right) = \frac{M!}{B!(M-B)!} = \frac{M^M B^{M-B}}{e^M B^B (M-B)^{M-B}} \sqrt{\frac{2\pi M}{2\pi B}} \sqrt{\frac{2\pi (M-B)}{2\pi B}}$$

$$\log \left(\frac{M}{B}\right) = \underbrace{M \log M - B \log B}_{B \log M + (M-B) \log M} - (M-B) \log (M-B) + O(\log M)$$

$$= B \log \frac{M}{B} + (M-B) \log \frac{B}{M-B} + O(\log M) = B \log \frac{M}{B} + O(B + \log M)$$

$$\ln 1+x \leq x$$

$$\log \frac{M-B+B}{M-B} = \log 1 + \frac{B}{M-B} = O\left(\frac{B}{M-B}\right)$$

$$O\left(B \log \frac{M}{B}\right)$$

Luego $n \log n - n \log B \leq t$

$$B \log \frac{M}{B}$$

$$\Rightarrow \frac{n}{B} \log_B \left(\frac{n}{B} \right) \leq t.$$

Cota Superior $\frac{n}{B} \log_B \left(\frac{n}{B} \right) = \mathcal{O} \left(\frac{n \log_M n}{B} \right)$

↳ no es tan temible.

$$M > n^\alpha \quad \forall \alpha < 1$$

$$\frac{n}{B} \log_B \left(\frac{n}{B} \right)$$

M tendría que ser demasiado grande para cambiar el orden de magnitud.

Tan grande que no tendría sentido práctico.