

CC4102 - Diseño y

Análisis de Algoritmos.

Prof: Gonzalo Navarro

of. 312

gnavarro@dcc.uchile.cl

1 [Cotas inferiores
Experimentación

2 [Memoria Secundaria
- ordenar
- colas de prioridad
- hashing

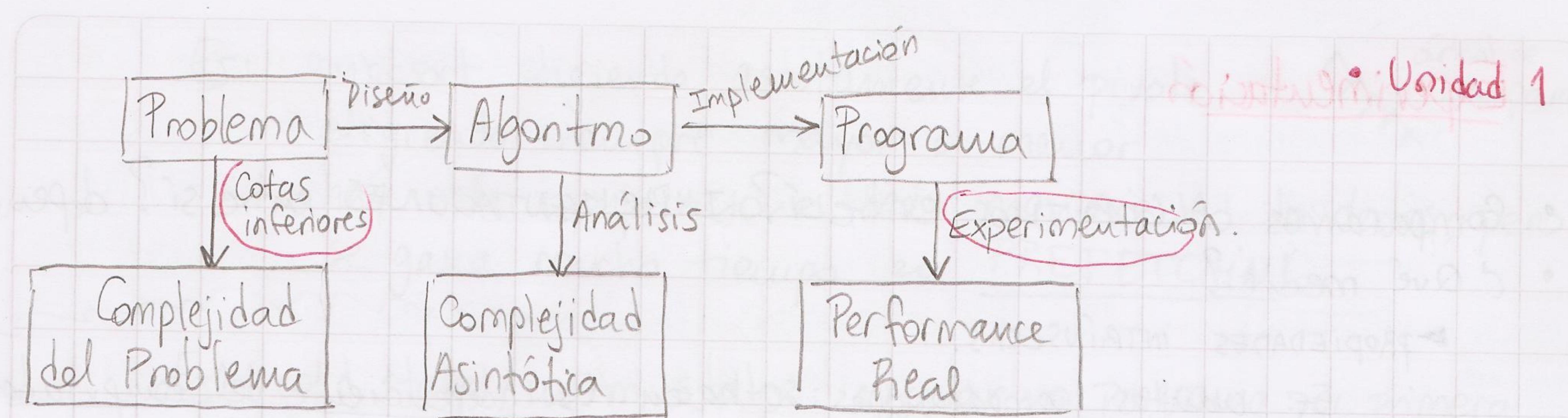
3 [Algoritmos avanzados
- amortización
- universos discretos
- competitividad

A [Algoritmos no convencionales
- probabilísticos/aleatorizados
- aproximados
- paralelos

↓
Examen

- 2/3 • 2 controles y 1 Examen
1/3 • 3 Tareas (en pareja o de a 3)
Si reprobaban tareas
→ Habrá una tarea recuperativa
(habiendo hecho de nuevo las tareas reprobadas)
• Ejecución con 5.0

2015年9月1日 (火)



2015年9月3日(木)

Experimentación

- Comparamos ALGORITMOS entre sí? IMPLEMENTACIONES entre sí? depende
- ¿Qué medir?

► PROPIEDADES INTRÍNSECAS:

Ej: cuántas comparaciones se hacen en el que trabaja.

MERGESORT vs QUICKSORT
en prom $n \log_2 n$ $\approx 1.44n \log_2 n$

Es independiente del computador en el que se trabaja.

pero quicksort es más rápido! en la práctica, pues mueve menos datos.

Ej: cantidad de accesos a discos.

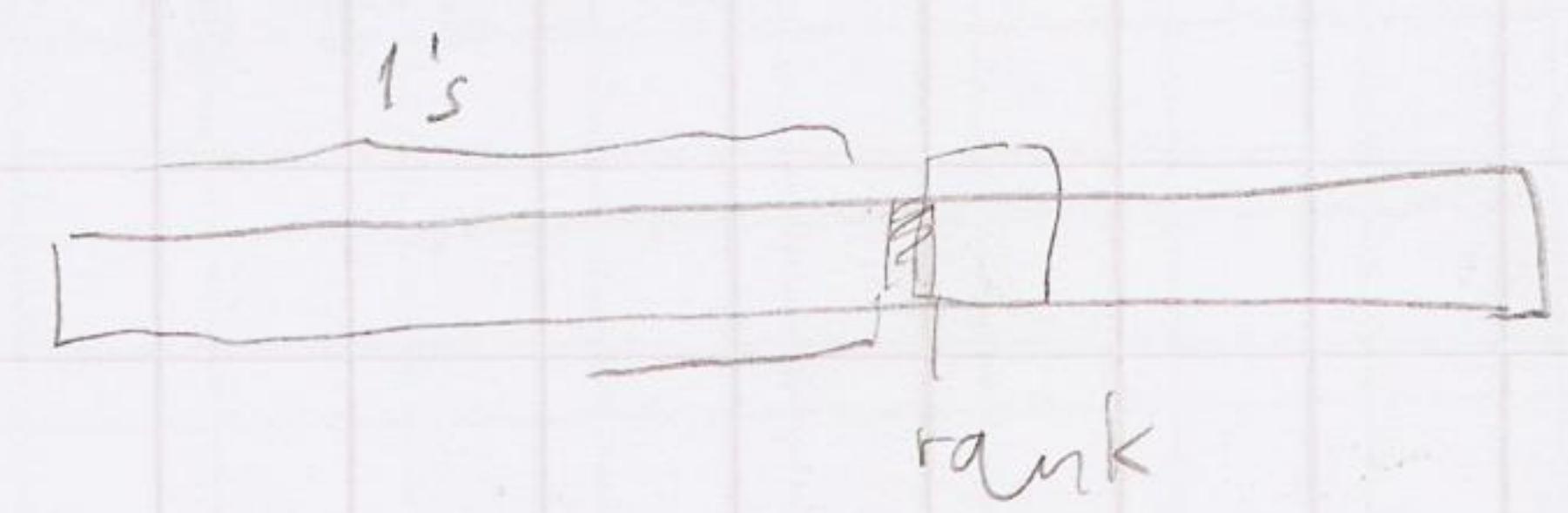
► TIEMPOS

I/O +
- usuario → CPU
- sistema → page-faults y seeks. El proceso no está corriendo realmente
- elapsed → tiempo medido por reloj. No se usa mucho. Se pueden contar también otros procesos que se ejecutan al mismo tiempo

Cosas que pueden afectar los tiempos:

- Cache
- garbage collection
- prefetching de instrucciones (adivinar por dónde irse en las branches de los ifs, usando incluso estadística)
- traducción de direcciones (memoria virtual a real)

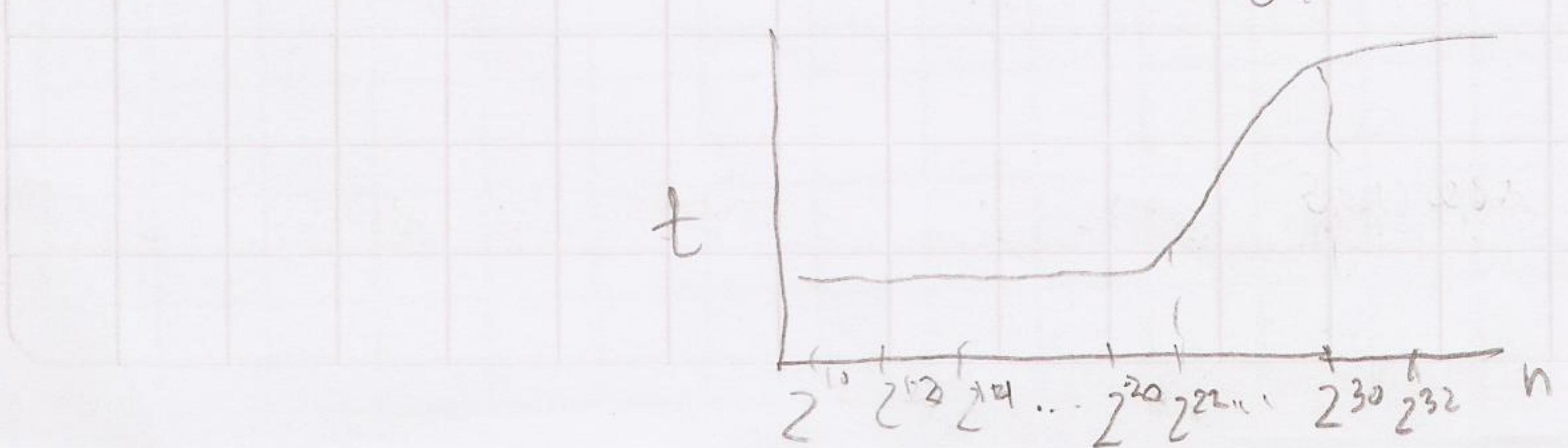
Ej: rank



$$\Theta(1) = 3 \text{ accesos a disco}$$

$n > 2^{20}$ el cache ya no sirve tanto.

$n > 3^0$ se hacen 3 accesos a disco si o si

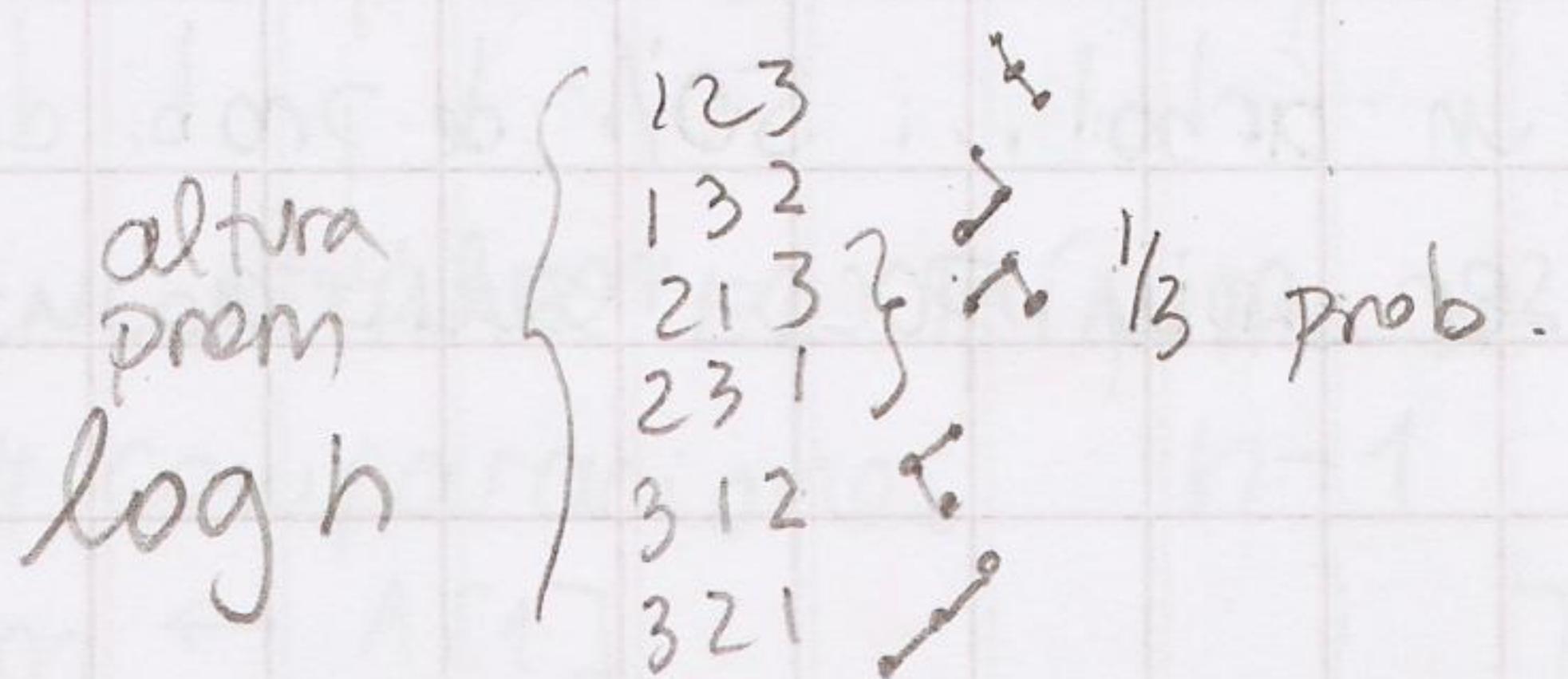


Ej: quicksort eligiendo astutamente el pivote.
→ eligiendo siempre mayor o menor
hay más comparaciones, más movimientos de datos, pero
se gana mucho tiempo en PREFETCHING

Ej: cold state / warm state : ejecutar un programa por primera vez, versus ejecutado muchas veces seguidas. El SO se acuerda del mapeo de memoria, y se dañara tanto menos en warm state.

• ¿Qué inputs?

Ej: árboles binarios aleatorios.

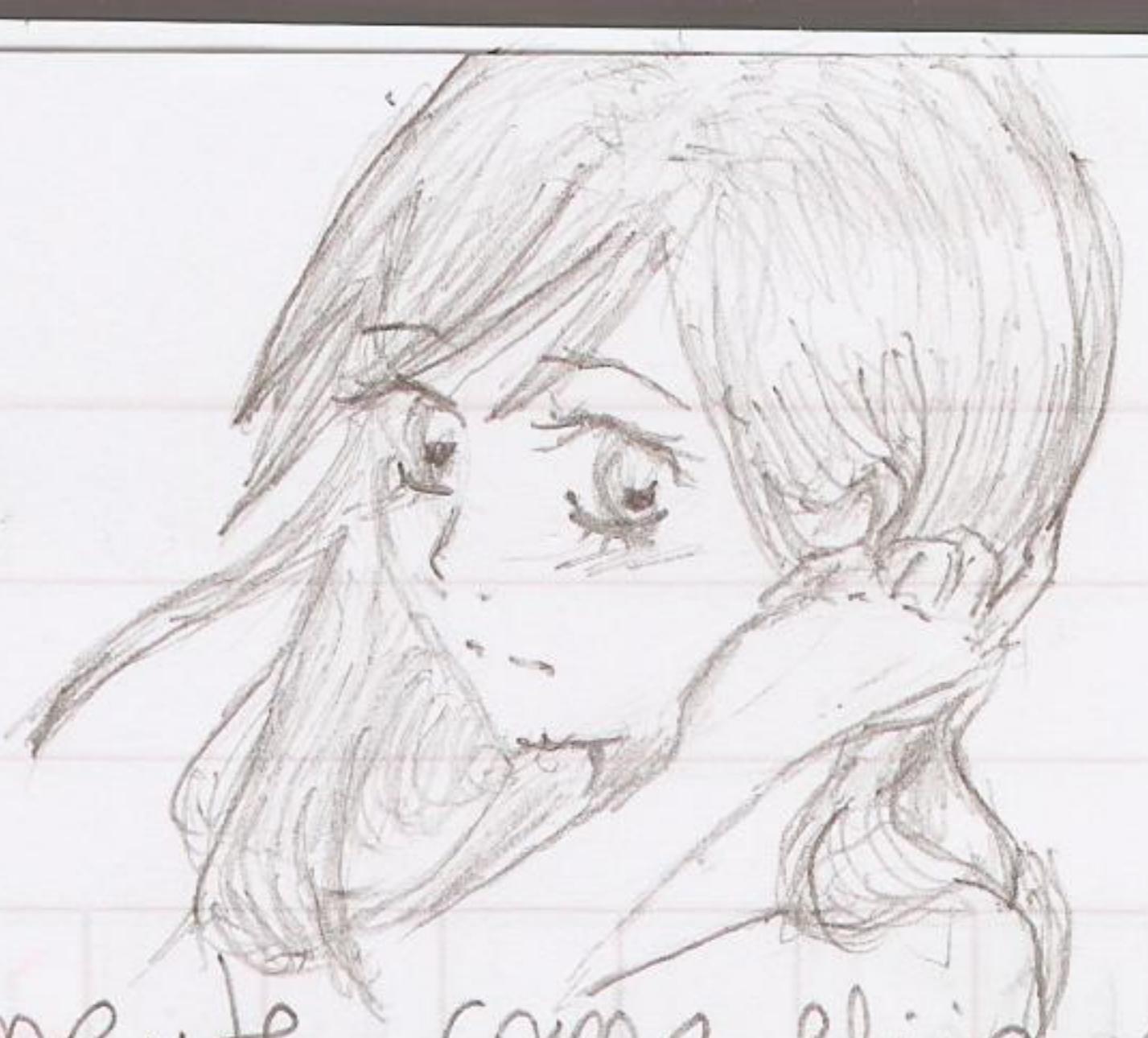


Ej: listas invertidas (inverted index), Intersección

[P1] → uuuuuu l_1 } $\Theta(l_1 + l_2)$
[P2] → uuuuuu l_2 } recover seavencialmente

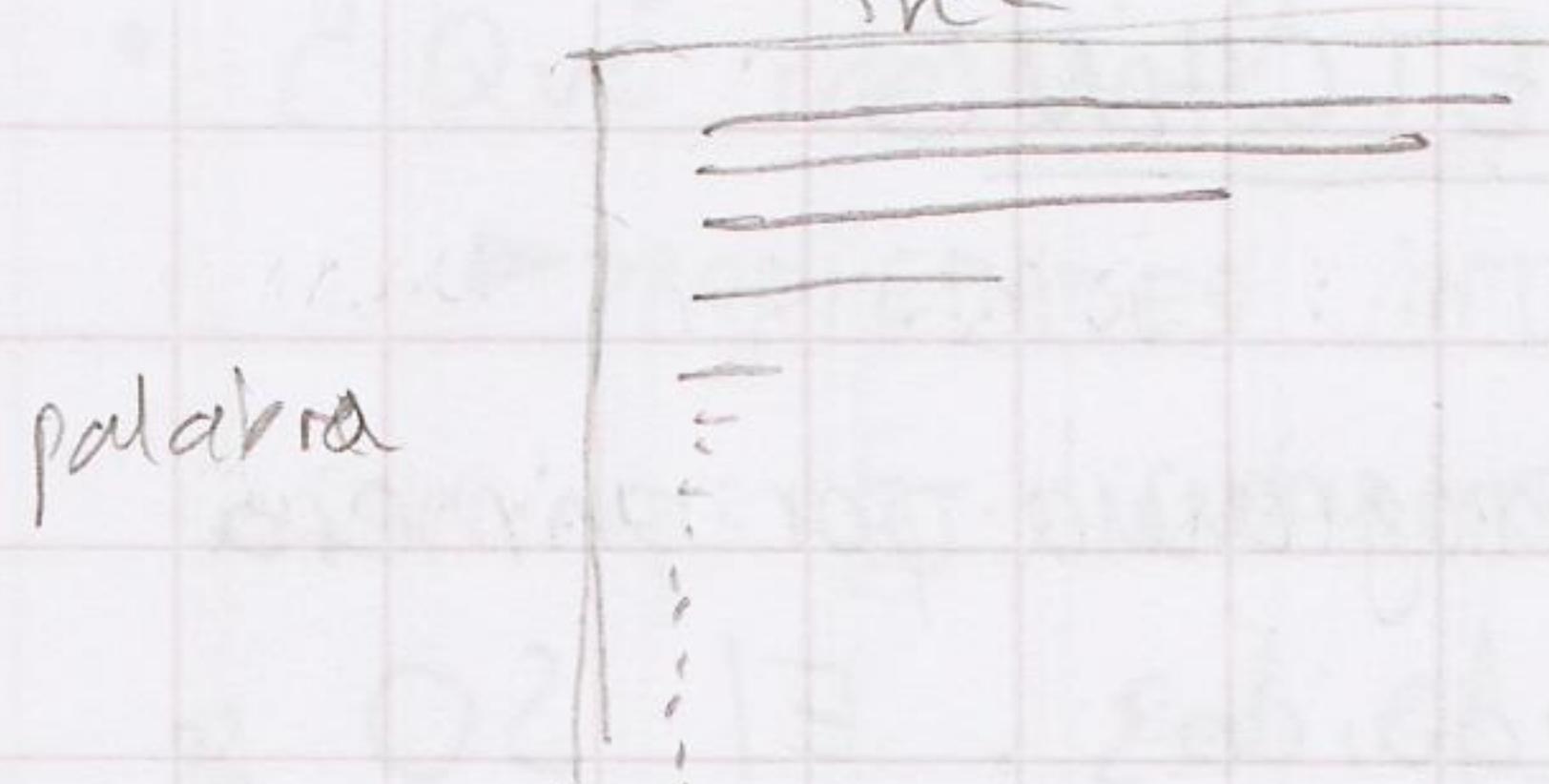
P3 \rightarrow new l_3 con $l_3 \ll l_2$

Es mejor hacer búsqueda binaria de todos los elfos en l_3 en l_2 . $\rightarrow \mathcal{O}(l_3 \log l_2)$



Experimentalmente como elijo cual de los 2 métodos usar?

Aleatoriamente? pero pocas palabras aparecen MUCHO y muchas palabras aparecen POCO (Zipf's law)



En este caso aleatoriamente no es muy buena idea experimentar para decidir qué método elegir, pues en la realidad la cuestión no es nada aleatoria!

Otro ejemplo es elegir una buena función de hashing para palabras, p.e. suma de carac ascii, tomar los primeros k carac... problema: comparten prefijos, palabras que tienen las mismas letras, etc. Probar permutaciones de letras aleatoriamente es pésima idea, porque el desempeño de mis funciones de hashing pararía que se comportan bien, pero en la realidad NO!

Otro ejemplo: elegir nodos al azar en un árbol... 50% de prob. de elegir una hoja! A veces eso puede ser muy poco realista.

2015年9月10日 (木)

Cotas inferiores de problemas

- Adversario (combate naval contra alguien que pone los barcos al final)
- Reducción (transformar un problema a otro)
- Teoría de la información (permite incluso deducir cotas inf promedio)
(analiza redundancia de la información)

• BÚSQUEDA EN UN ARREGLO DESORDENADO. $A[1, n]$

→ min # accesos en el peor caso = n

¿por qué? hay que razonar en base a cualquier algoritmo. No importa el orden en que lo haga, si reviso $n-1$ celdas, en el peor caso siempre el número va a estar en la celda que falta.
(como el ejemplo de combate naval)

→ min # accesos en arreglo ordenado? con búsq. binaria sabemos que el peor caso es $\log n \rightarrow$ el adversario tiene más restricciones con respecto a donde poner el elemento maliciosamente.

• ENCONTRAR EL MÍNIMO DE $A[1, n]$.

comparaciones $n-1$

$m \leftarrow A[1]$

for $i \leftarrow 2$ to n

if $A[i] < m$

$m \leftarrow A[i]$

return m .

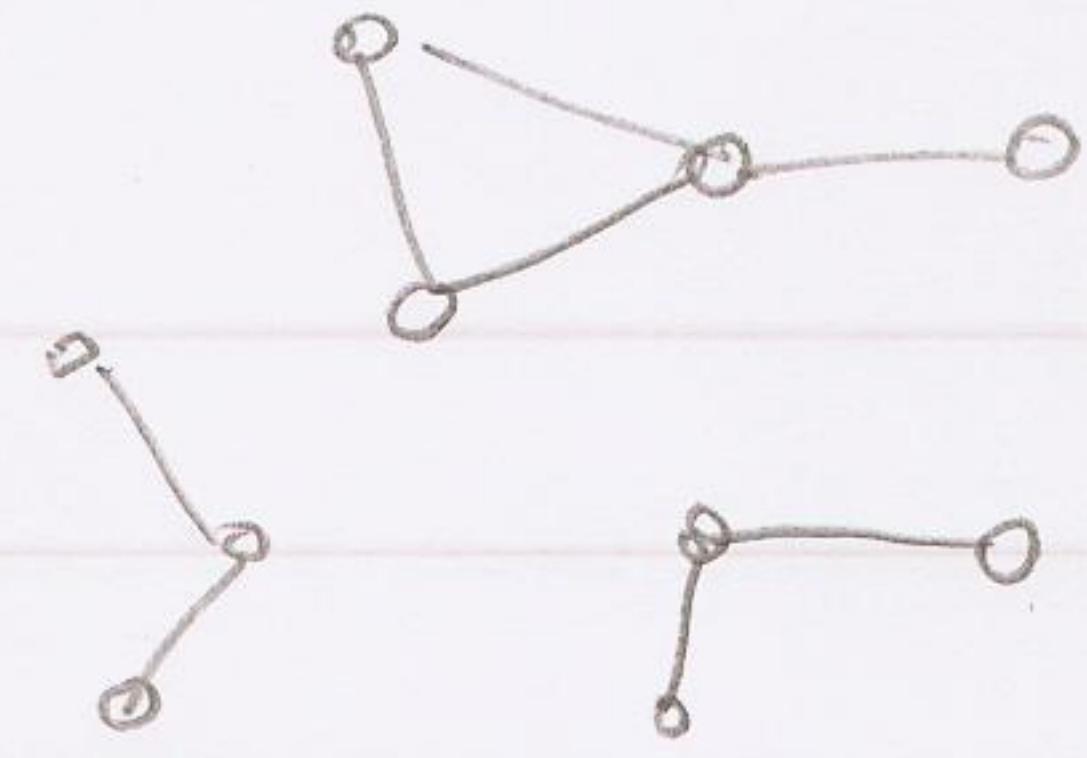
También podemos pensar en el TORNEO... se ve lindo, es un buen algoritmo paralelo pero también se realizan $n-1$ comp.

¿Podemos hacer menos de $n-1$ comp?

podríamos usar un argumento similar al del problema anterior pero... ¡en el primer paso del torneo TODOS los elementos ya se compararon con otro!

→ $\Omega(n \log \frac{n}{2})$

(本) BOL RAZIOS



los arcos son comparaciones

Argumentamos con grafos. Debemos hacer suficientes comparaciones para producir un grafo CONEXO \rightarrow si tengo n nodos, necesito $n-1$ arcos para producir un grafo conexo.

Lo importante es que el adversario puede seguir siendo consistente cuando coloca el mínimo en una componente conexa que he descartado (pues el adversario solo dice respuestas SÍ-NO)

Vamos a modelar el conocimiento del algoritmo sobre el input de otra forma

(a, b, c) $a = \#$ 2tos nunca comparados.

$b = \#$ 2tos comparados alguna vez y siempre menores

$c = \#$ 2tos comparados alguna vez y alguna vez mayores

Cualquier algoritmo correcto parte de $(n, 0, 0)$ y llega a $(0, 1, n-1)$

¿Cómo es la evolución $(n, 0, 0) \rightarrow (0, 1, n-1)$?

	a	b	c
a	$(a-2, b+1, c+1)$ para los 2 casos	$(a-1, b+1, c)$	$(a-1, b, c+1)$
b	lo mismo	$(a, b-1, c+1)$	(a, b, c) $(a, b-1, c+1)$
c	lo mismo	lo mismo	(a, b, c)

en el peor caso el adv.
Puede hacer que ocurra
eso sin entrar en inconsistencias

→ NO SIRVE, no me conviene

¡Es directo!

Para pasar de $C=0$ a $C=n-1$, como en cada paso c aumenta a lo sumo en 1, necesito hacer $n-1$ comparaciones.

Otra cosa interesante es que deduzco las comparaciones que más me convienen! \rightarrow Puedo deducir un algoritmo :)

Alternativa 1: comparar a con $a \rightarrow (n-2, 1, 1)$ y luego hacer el resto de las comparaciones a con b
 ↳ algoritmo for!

Alternativa 2: comparar a con $a \frac{n}{2}$ veces $(n, 0, 0) \rightarrow (0, \frac{n}{2}, \frac{n}{2})$
 y luego seguir comparando b con b .

- ENCONTRAR EL MÁX Y EL MÍN DE $A[1, n]$.
 $n-1 + n-2 = 2n-3$ comp?

Extendamos el modelo anterior...

(a, b, c, d)

- $a = \#$ eltos nunca comparados
- $b = \#$ eltos comparados algunas vez y siempre menores
- $c = \#$ " " " " y siempre mayores
- $d = \#$ " " " " y alguna vez mayores y otras menores.

$$(n, 0, 0, 0) \rightarrow (0, 1, 1, n-2)$$

	a	b	c	d
a	$(a-2, b+1, c+1, d)$ si "a" gana $(a-1, b, c+1, d)$	$(a-1, b+1, c, d+1)$ si "c" gana $(a-1, b+1, c, d)$	$(a-1, b+1, c, d)$	$(a-1, b, c+1, d)$
b		$(a, b-1, c, d+1)$ "b" gana $(a, b-1, c-1, d+2)$	(a, b, c, d) si "b" gana $(a, b-1, c, d+1)$	(a, b, c, d) si "b" gana $(a, b-1, c, d+1)$
c			$(a, b, c-1, d+1)$	(a, b, c, d) $(a, b, c-1, d)$
d				(a, b, c, d)

- Notemos que la gran masa va a d .
- d crece a lo sumo de $a+1$ (adversario) y en ese caso a NO decrece.
 $\rightarrow n-2$ comp necesarias.

• Para hacer pasar a de n a 0 , hago $\lceil \frac{n}{2} \rceil$ comp.

⇒ cota inferior $\lceil \frac{3}{2}n \rceil - 2$

¿Podemos derivar un algoritmo que nos lleva a la cota inferior?

Veamos ...

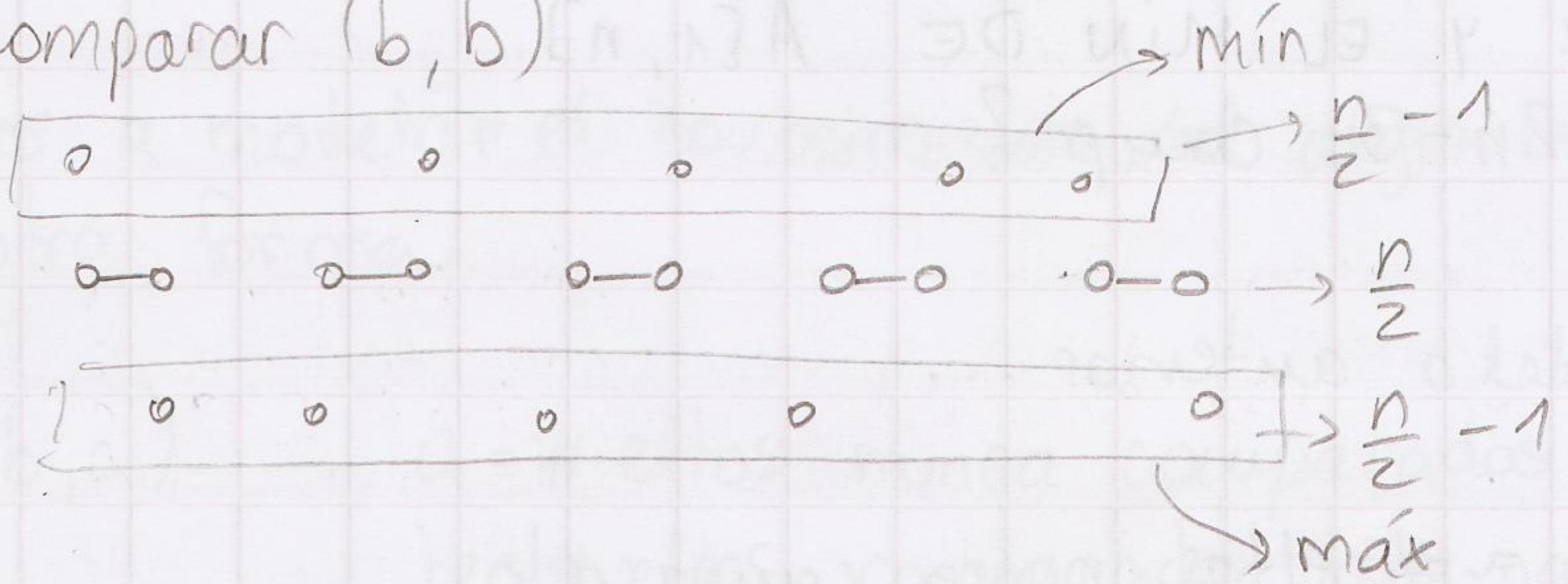
→ (a, b) no me conviene mucho... es como el for.

→ Es mejor empeñar a comparar pares desconocidos (a, a)

$$(n, 0, 0, 0) \xrightarrow{n/2} (0, \frac{n}{2}, \frac{n}{2}, 0) \rightarrow \text{más rápido}$$

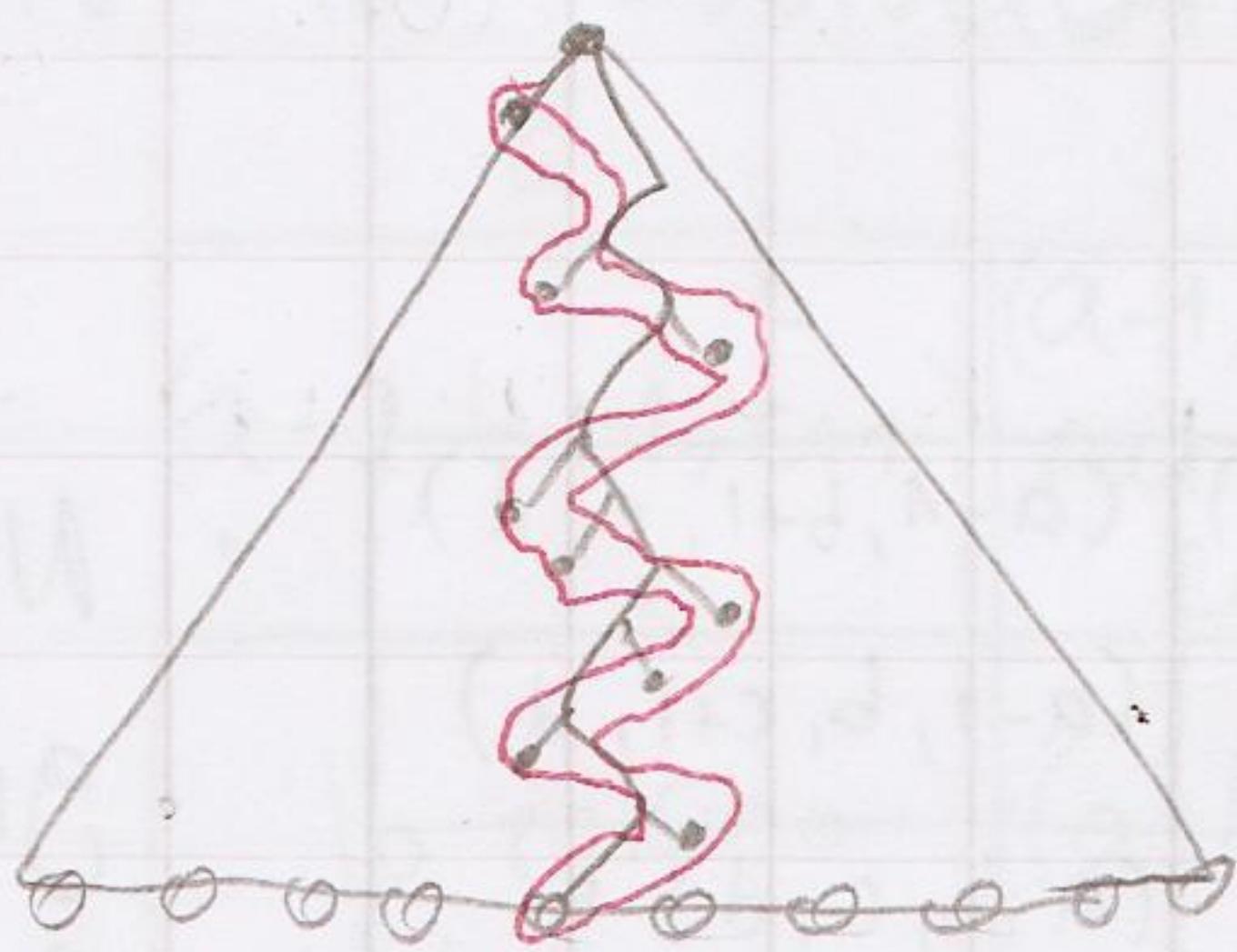
→ Comparar (b, b)

⇒



Como encontré un algoritmo, ahora SÉ que no hay cota inferior más alta. (A veces hay gap)

• Máx y $2^{\text{º}} \text{MÁX}$ $n-1 + \lceil \log_2 n \rceil$



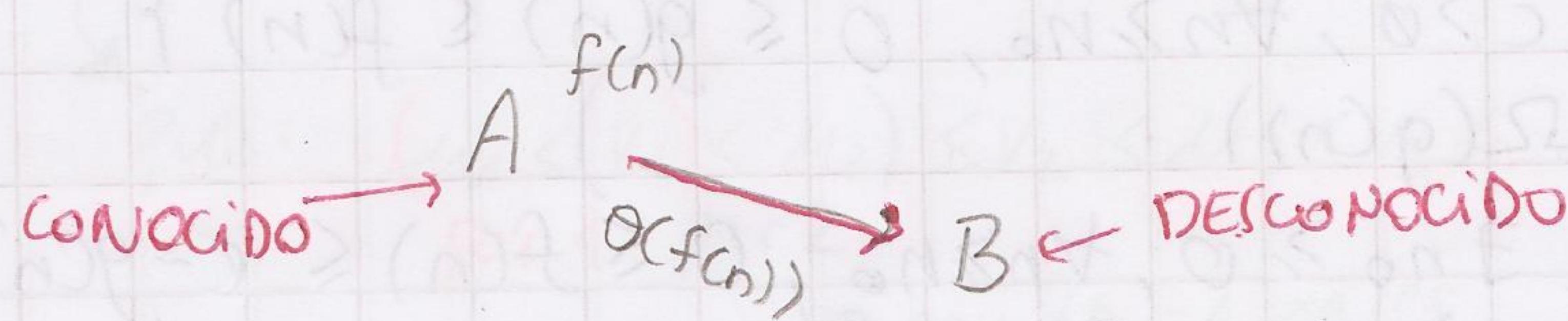
¿Es el óptimo? Aux.

Hacer la tablita del adversario es bien complicado.

Esto fue ADVERSARIO.

★ Técnicas de reducción:

Si A tiene una cota inferior de $f(n)$ y se puede reducir a B en tiempo $\Theta(f(n))$ entonces B tiene una cota inferior $\Omega(f(n))$

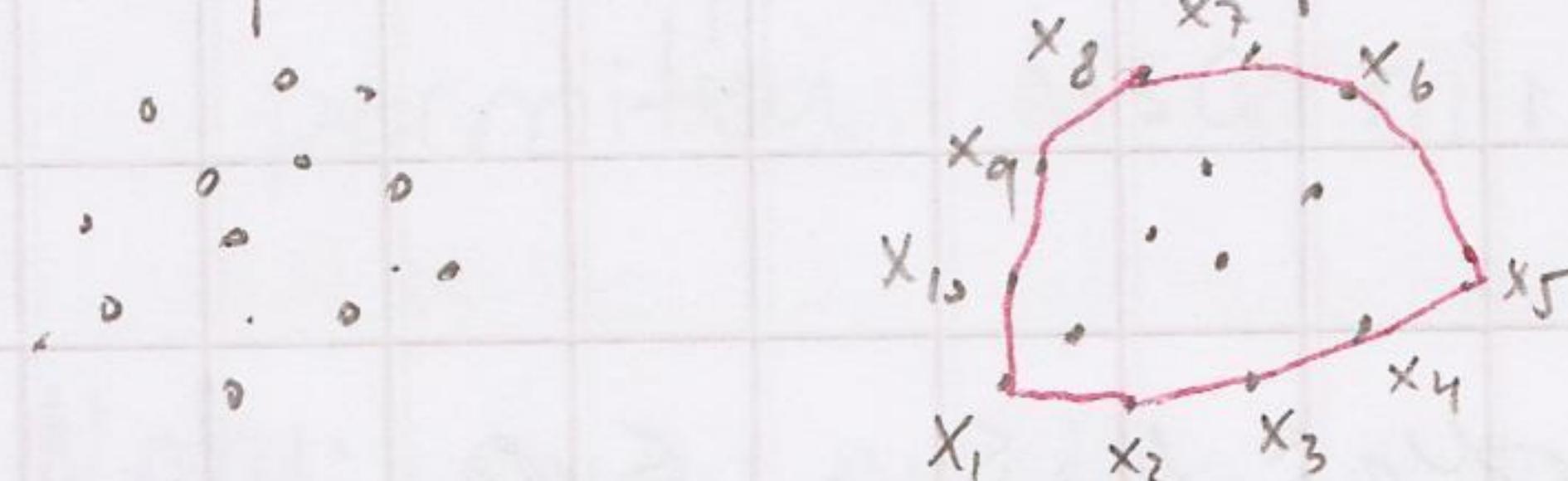


Si se hace lo único que hago es complicar más mi problema desconocido.

Ej. Ordenar es $\Omega(n \log n)$

→ Cápsula convexa "convex hull"

input → output



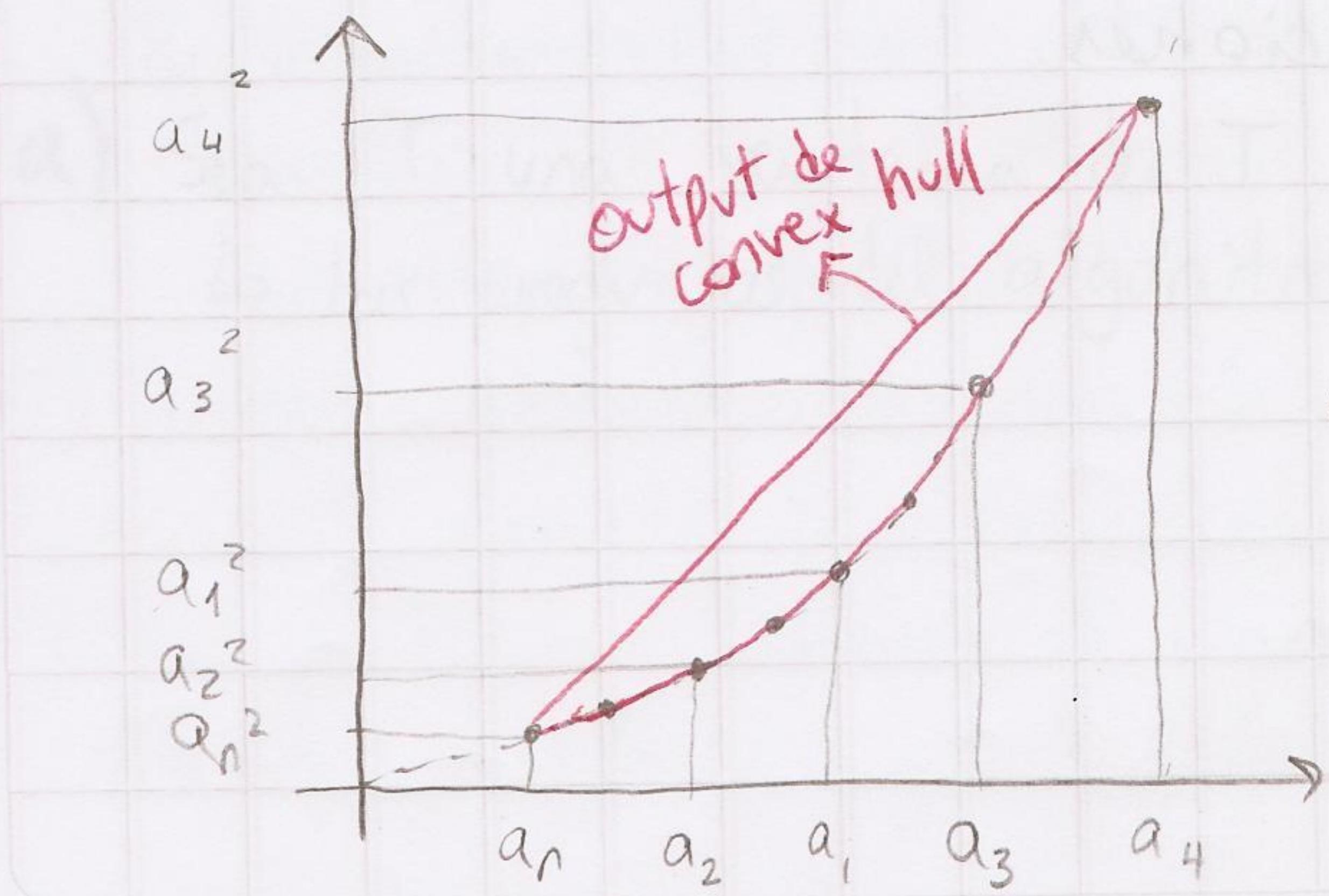
lista
 x_1
 x_2
 x_3
 x_4
 x_5
 x_6
 x_7
 x_8
 x_9
 x_{10}

(ejemplo, encontrar el mínimo, reducirlo a ordenar y extraer el mínimo)

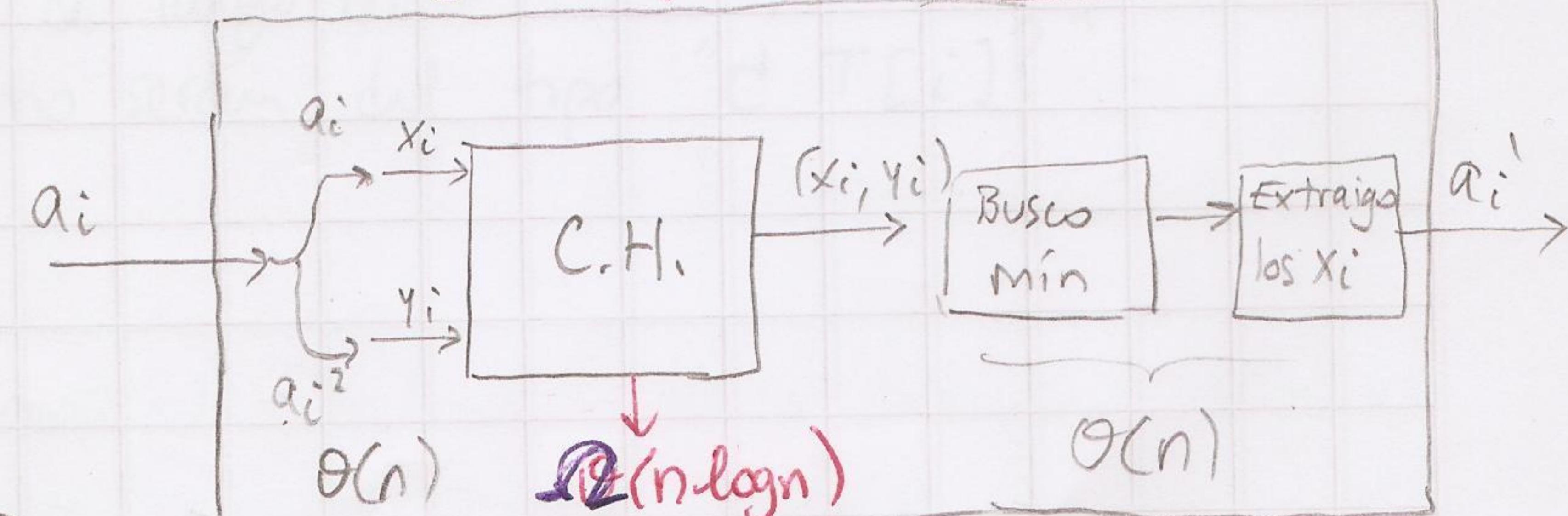
mínimo (en área) polígono que contiene a todos los puntos. Entrega una lista ORDENADA de los puntos (recorre los puntos en sentido antihorario).

Tomemos un problema de ordenamiento y lo transformamos a un problema convex hull. Luego veremos cómo, de la solución de convex hull podemos traducir "fácilmente" a la solución del problema de ordenamiento.

$$(a_1, a_2, a_3, \dots, a_n) \xrightarrow{\quad} (a_1, a_1^2), (a_2, a_2^2), (a_3, a_3^2), \dots, (a_n, a_n^2)$$



Sort $\Theta(n \log n)$



Si C-H. se demorara menos, contradicción con SORT.