Resumen C1 Logaritmos

Cotas inferiores:

Estrategia del adversario:

Se encarga de asegurarse que el algoritmo siempre está en el peor caso.

- **Encontrar un valor en un arreglo desordenado:**
  Cota inferior de n accesos al arreglo. Sin importar como lo resuelva si no miro una celda puede pasar que justo esa sea la que contenga al elemento que estoy buscando.
  Adversario-> produce el peor caso todo el rato. Si estuviera ordenado, el adversario no puede cambiar el arreglo.
- **Encontrar el mínimo en un arreglo de n:**
  Modelo del grafo: Si los nodos representan los elementos y las aristas son las comparaciones, se necesitan n-1 aristas es un grafo de n elementos para que todo esté conectado.
  Si tengo un grafo no conexo entonces habrá dos mínimos porque hay dos que no se habrán comparado => se necesitan n-1 comparaciones.
  Otro modelo:
  A= # de elemento que nunca se han comparado
  B= # de elementos comparados y que 100pre ganan
  C= # de elementos comparados y que perdieron alguna vez
  La idea es pasar de n elementos nunca comparados a uno que siempre ha ganado y n-1 que perdieron (n,0,0) ->(0,1,n-1)

	a	b	C
A	(a-2,b+1,c+1)	(a-1,b,c+1)	(a-1,b+1,c) (a-1,b,c+1)
B		(a,b-1,c+1)	(a,b,c)(b gana) (a,b-1,c+1) (c gana)
C			(a,b,c)

  Si voy viendo como crecen a,b,c, c crece de a 1 entonces necesitamos n-1 comparaciones para llegar al estado final esperado.
- **Encontrar el mínimo y el máximo:**
  O= # de elemento que nunca se han comparado
  G= # de elementos comparados y que 100pre ganan
  P= # de elementos comparados y que 100pre perdieron
  E= # de elementos que ganaron y perdieron
  (n,0,0,0)->(0,1,1,n-2)

	$a \in O$	$a \in G$	$a \in P$	$a \in E$
$b \in O$	$o-2, g+1, p+1, e$	$o-1, p, e+1$ $\mathbf{o-1, g, p+1, e}$	$o-1, g, p, e+1$ $\mathbf{o-1, g+1, p, e}$	$o-1, g+1, p, e$ $o-1, g, p+1, e$
$b \in G$		$o, g-1, p, e+1$	$\mathbf{o, g, p, e}$ $o, g-1, p-1, e+2$	$\mathbf{o, g, p, e}$ $o, g-1, p, e+1$
$b \in P$			$o, g, p-1, e+1$	$\mathbf{o, g, p, e}$ $o, g, p-1, e+1$
$b \in E$				o, g, p, e

Un adversario puede maximizar la complejidad del algoritmo eligiendo el resultado de cada comparación. En la tabla se marcan las opciones que maximizan la complejidad del algoritmo.
El E crece de a 1 -> o(n-2) para el estado final. Cuando E crece O se mantiene igual, pero si se reduce a lo sumo de a 2 -> O($\left\lceil \frac{n}{2} \right\rceil$) comparaciones.

⇨ N-2+n/2 = $\left\lceil \frac{3n}{2} \right\rceil - 2$ comparaciones.

Otro método: Tomo elementos de a 2. A los ganadores les hago un torneo de tenis y a los perdedores lo mismo. n/2 comparaciones, el torneo de tenis de mínimo y máximo es n-1, entonces (n/2 -1)+ n/2 + n/2-1 + n/2-1=O(3n/2-1).

- Segundo máximo: Encontrar el máximo es n-1, si hacemos un árbol, habrá que buscar todos los que perdieron con el máximo ($\log_2 n$). De los $\log_2 n$ encuentro el máximo de esos entonces la cota es $\log_2 n$-1 => n-1 + $\log_2 n$-1= $\log_2 n$ + n -2.

Reducción:

Un problema desconocido lo reduzco a uno conocido. Algoritmo A se puede modificar para resolver un algoritmo B. La cota inferior para el algoritmo B es válida también para el algoritmo A.

Ejemplos

- Ordenar n elementos por comparaciones: $\Omega(\mathrm{n} \log n)$
  Lo reducimos el problema al problema *convex hull* (cápsula convexa)(Tengo puntos y busco la envoltura convexa más pequeña que contenga todos los puntos).

- Multiplicación de dos matrices:

- 3 puntos colineales:

Teoría de la información:

Algoritmos de memoria secundaria:

B= tamaño de un bloque

Complejidad del algoritmo : # de lecturas o escrituras del algoritmo.

M= tamaño de la RAM (memoria ppal)
