Adaptive Sorting

Note Title 3/30/2010

① Merge Sort (and variants)

② $O(n \lg n)$

$f(n) \in O(n \lg n)$

MergeSort $(A, 1, n)$ $f(n) =$

MergeSort $(A, 1, \lfloor n/2 \rfloor)$ $f(\lfloor n/2 \rfloor)$

MergeSort $(A, \lfloor \frac{n}{2} \rfloor + 1, n)$ $+ f(\lfloor n/2 \rfloor)$

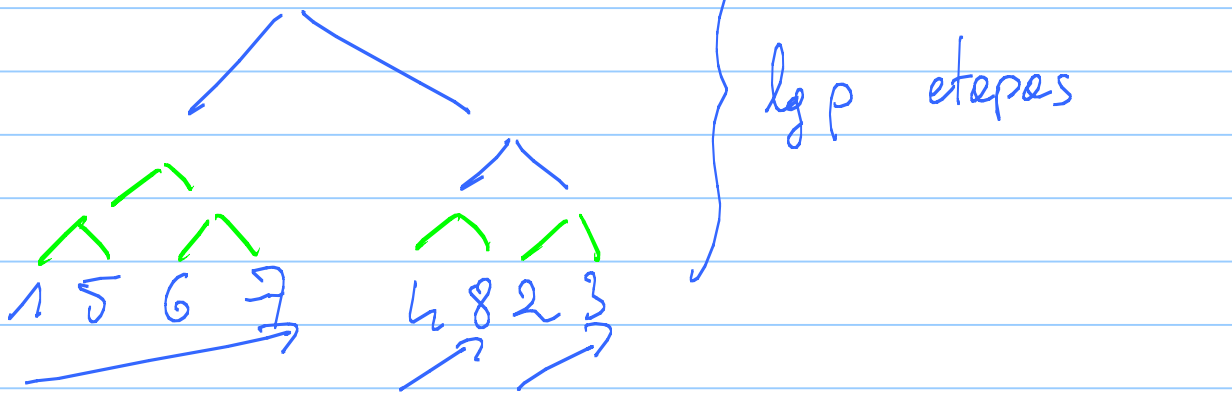
Merge $(A, 1, \lfloor \frac{n}{2} \rfloor, \lfloor \frac{n}{2} \rfloor + 1, n)$ $+ n$

(b) $O(n \lg p) \subseteq O(n \lg n)$

Idea ① Detectar "Runs" en tiempo lineal

② Hacer un "Merge" de los runs

$\Rightarrow$ Complejidad $= O(n) + O(n \lg p) = O(n \lg p)$

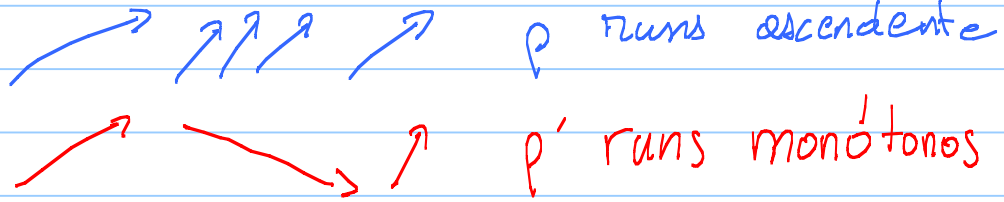


- Rq en A = 87654321

  el algoritmo performa $n \lg n$ porque $p = n$

  Pero A es ordenado!

- Rq en A = 14876523



p runs ascendente

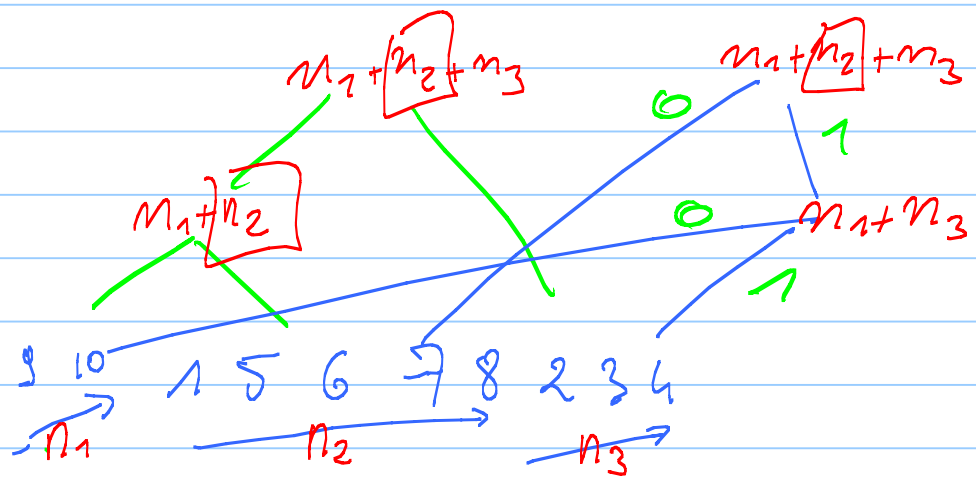
p' runs monótonos

- Rq $p' \leq p$ y $O(n \lg p') \subset O(n \lg p)$

© $O(n\, H(Runs))$

Runs = vector de los tamaños de los runs.

r = # runs



Cuando los "runs" son de tamaños distintos, algunos arboles de union son mejores que otros. Cual elegir?

* Minimisar la cantidad de comparaciones en un merge
  ⇒ es análogo a la optimisacion de un codigo para describir de cual bloc viene cada elemento del output.

$\Rightarrow$ Calcular el árbol de Huffman sobre $(n_1, \dots, n_p)$

$n(H+1)$ bits de código

$\Longrightarrow$ $n(H+1)$ comparaciones para Merge

$+ \; n$ comparaciones para Detectar los runs

$n(H+2)$

donde $H = H(\text{Runs}) = H(n_1, \dots, n_p)$

II Relacion con Compression de Permutaciones

Algoritmo de ordenamiento adaptivo con comparaciones

da un codigo por permutaciones

una compression

III Otras Medidas de Dificultades

	Best	Worst
- Bubble Sort	$O(n)$	$O(n^2)$
- Insertion Sort con arreglo	$O(n)$	$O(n^2)$
- Insertion Sort con Splay Tree	$O(n)$	$O(n \log n)$
- Quick Sort con Mediana	$O(n \log n)$	$O(n^2)$