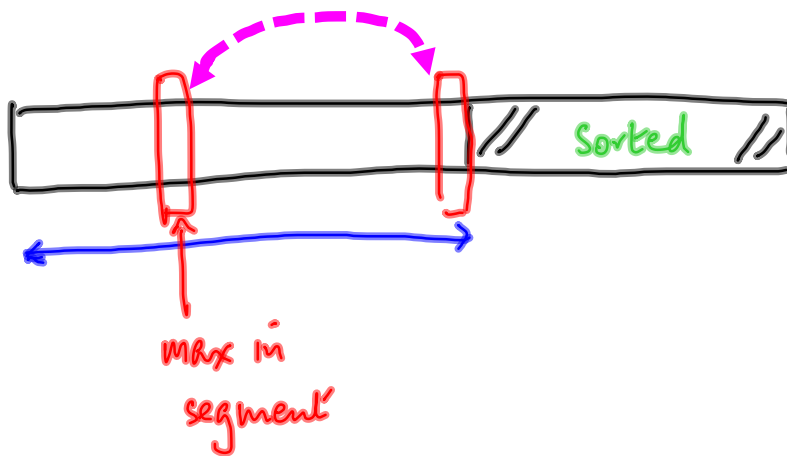


In place sorting

Selection Sort  $O(n^2)$

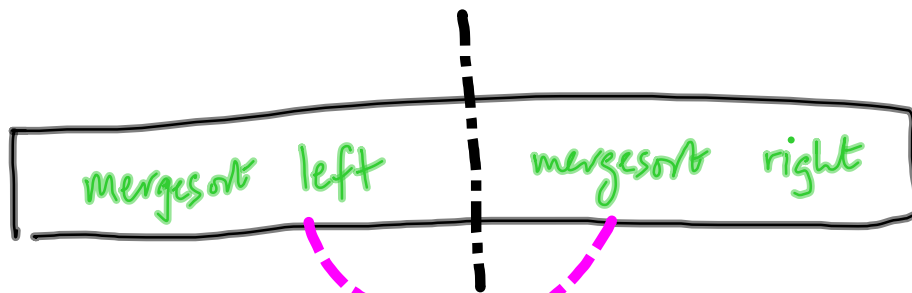


exchange max value with last value  
in unsorted segment  
extend sorted suffix by one

More efficient sorting

Merge sort

Merging 2 sorted lists/arrays of length  $n$  takes  
 $O(n)$



merge into a single sorted sort.

Complexity - set up & solve recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + \cancel{O(n)} n$$

2 recursive sorts                      merging

$$= 2 \left( 2T\left(\frac{n}{2 \cdot 2}\right) + \frac{n}{2} \right) + n$$

$$= 2^2 T\left(\frac{n}{2^2}\right) + 2n$$

$$= 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$T(0) = T(1) = 1$$

At  $k = \log_2 n$   $\frac{n}{2^k}$  becomes 1

$$T(n) = 2^{\log_2 n} T(1) + (\log_2 n) \cdot n$$

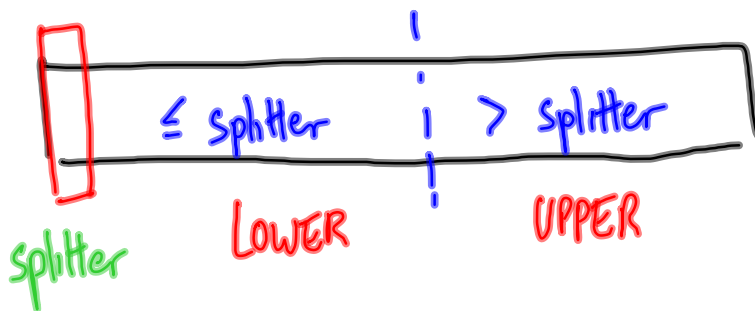
$$= n \cdot 1 + n \cdot \log n$$

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

$n$	$10^3$	$10^6$	Any standard CPU can do at most $10^9$ operations/second
$n^2$	$10^6$	$10^{12}$	
$n \log_2 n$	$10^4$	$2 \cdot 10^7$	

For all practical purposes,  
 $O(n \log n)$  is as good as  $O(n)$

## Quicksort



$qsort\ l = (qsort\ lower) ++ [splitter] ++ (qsort\ upper)$

needs no merging

Good point : No merging required

Bad point : Split may be uneven

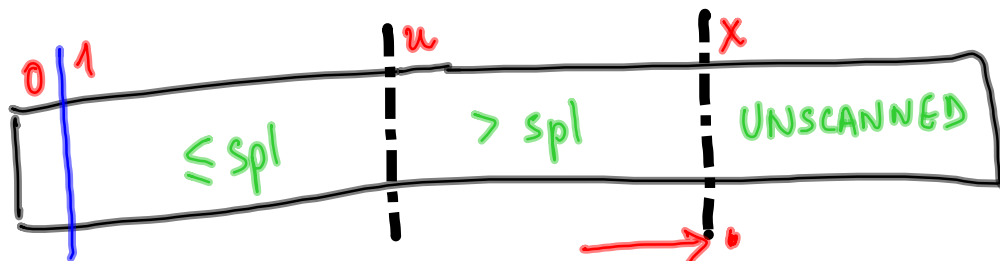
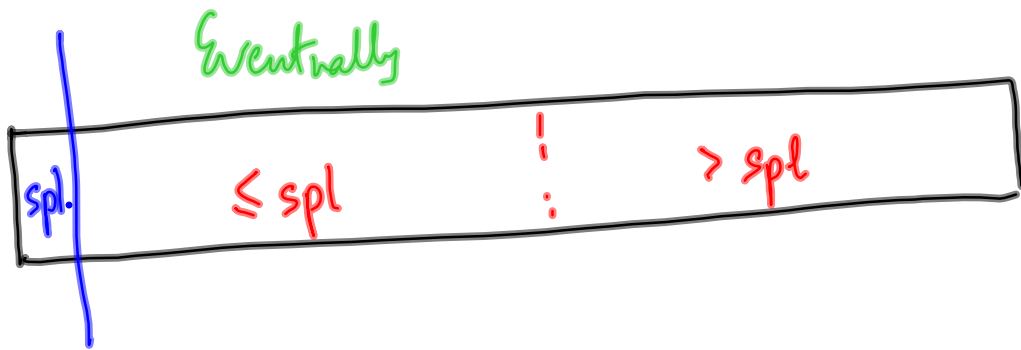
Worst Case recurrence

$$T(n) = T(n-1) + n$$

$$\vdots \quad n + n-1 + \dots = O(n^2)$$

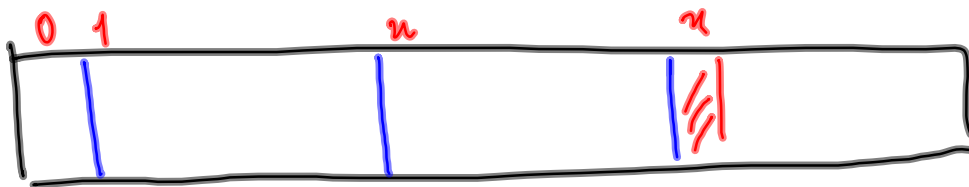
But on the average quicksort is  $O(n \log n)$   
randomly choose splitter - achieve this on the average

Quicksort can be done in place



→ Scan and place  $e[n]$  in appropriate partition

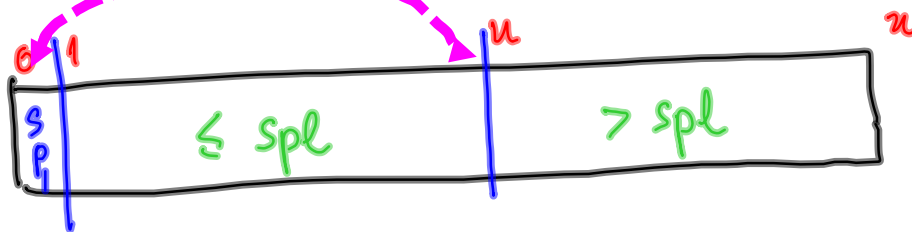




$l[r] \leq \text{splitter}$  : Exchange  $l[u], l[r]$   
Increment  $u, r$

$l[r] > \text{splitter}$  : increment  $r$

After one scan



Want spl between lower & upper segments.

Exchange  $l[0]$ ,  $l[u-1]$

To sort lower, upper in place

Add left, right boundaries as arguments

def qsort( $l$ , left, right)

Can actually implement qsort as a loop,  
without recursion

Typically, built in sort functions use quicksort

Quicksort was invented by C.A.R. Hoare  
approx 50 years ago, before PL's had  
recursion; in fact, written in assembly lang

Mergesort & Quicksort are both examples of

Divide & Conquer

Netflix challenge

Movie rental shop

Customers rate movies

Make good recommendation

↳ identify users with similar preferences

2 users,  $N$  items, each user ranks  $N$  items

$$i : i_1 <_i i_2 < \dots <_i i_N$$

$$j : j_1 <_j j_2 < \dots <_j j_N$$

$i$  &  $j$  agree on a pair of items  $x$  &  $y$  if  
the relative rankings are the same

$$x <_i y \text{ iff } x <_j y$$

Measuring similarity

How many of the  $\frac{N \cdot (N-1)}{2}$  pairs do  $i$  &  $j$  agree on?

Fix an ordering of the  $N$  items as ranked by  $i$

For  $i$ :  $1 <_i 2 <_i 3 \dots <_i N$

For  $j$ :  $j_1 <_j j_2 <_j j_3 \dots <_j j_N$

$(j_1, j_2, \dots, j_N)$  is a permutation of  $(1, 2, \dots, N)$

e.g. 3 2 1 4 5 vs 1 2 3 4 5

(3,2) is an inversion

For each position, check for smaller values to right

(3,2), (3,1)

$O(n^2)$

(2,1)

=> 3 inversions (out of 10)

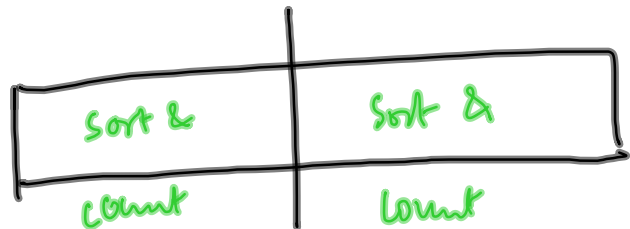
Can we do better

There could be  $O(n^2)$  inversions

Can we count them without enumerating them?

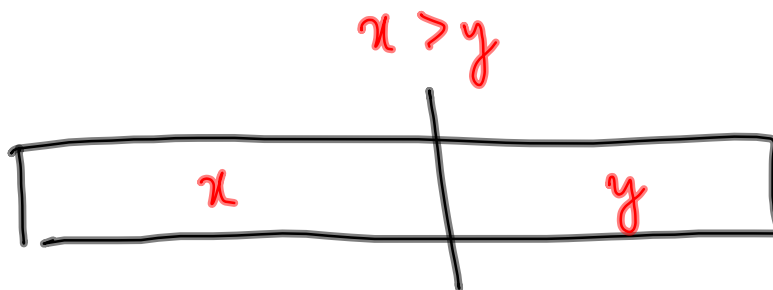
Can we divide & conquer?

Like merge sort



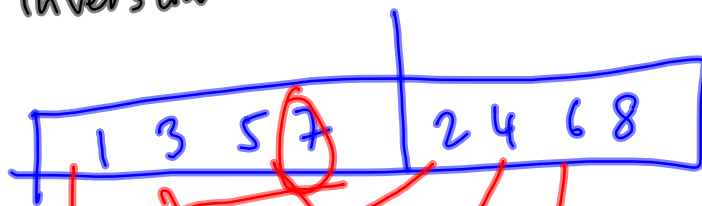
Need to count inversions across boundary





each value from right "overtakes" all current values in left

In merge, whenever we add an element from right half, count current left & add to inversion



1 2 3 4 5 6 7 8

Inversion count

$$0 + 3 + 0 + 2 + 0 + 1 + 0 + 0$$

$$= 6 \text{ inversions}$$