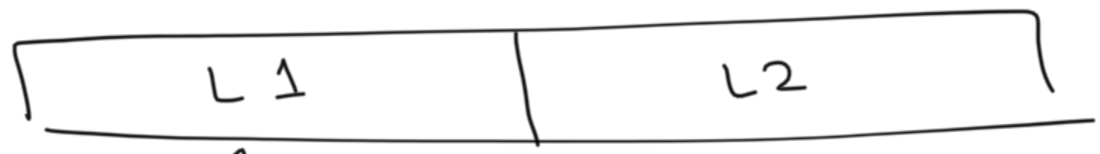
Quick sort

(Tony Hoare, ~1960s)

merge sort :



L

↑ ↑

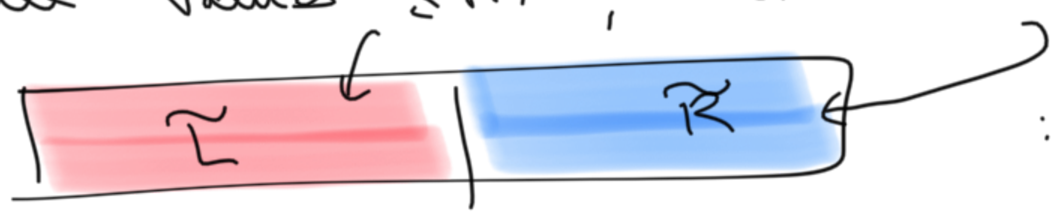
→ merge sort L1   merge sort L2

→ Then merge the two } needs |L| space....

Quick sort : (Avoids need for extra space)

→ Let median of L = m

→ move all values ≤ m , all values > m



: L

} Claim: can be done in $O(|L|)$ time

→ Quick sort ($\tilde{L}$), Quick sort ($\tilde{R}$) in place

→ No need to merge!

complexity analysis

If so, $|L| = n \Rightarrow |\tilde{L}|, |\tilde{R}| = n/2$

as m = median

$$T(n) = 2T(n/2) + O(n)$$

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

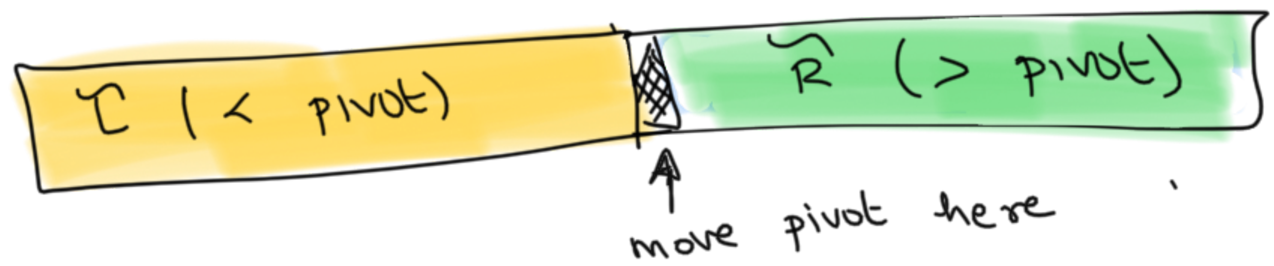
---

step 0   How to find median ?

– Oops...

– Pick some value in L "pivot"

Step 1



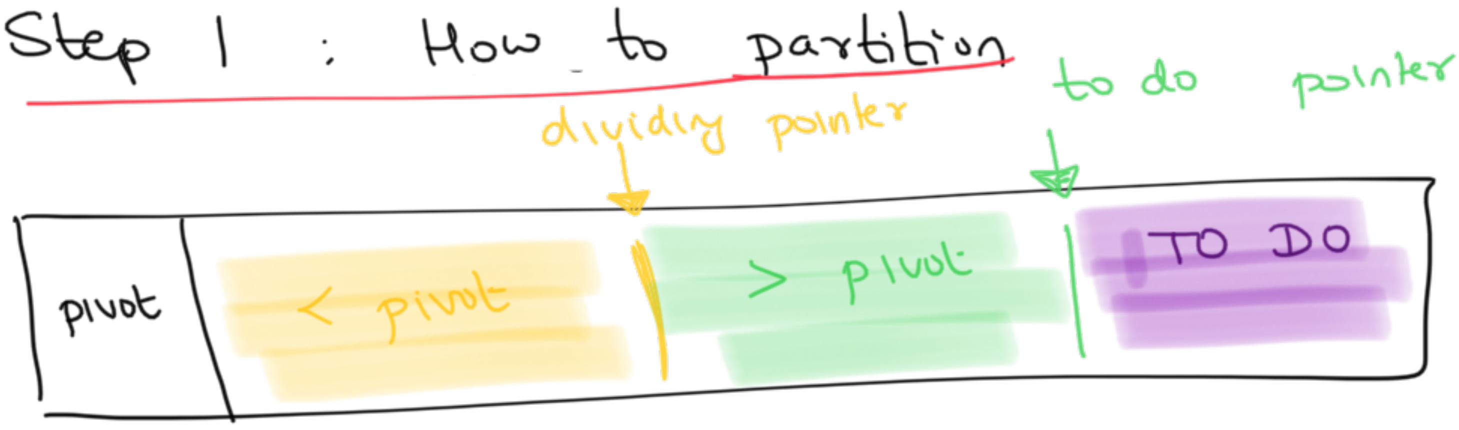
will see how to do this

Step 2 : Quicksort (L̃), Quick sort (R̃) [Recursive call]

Step 1 : How to partition

Forward partioning algorithm

1A



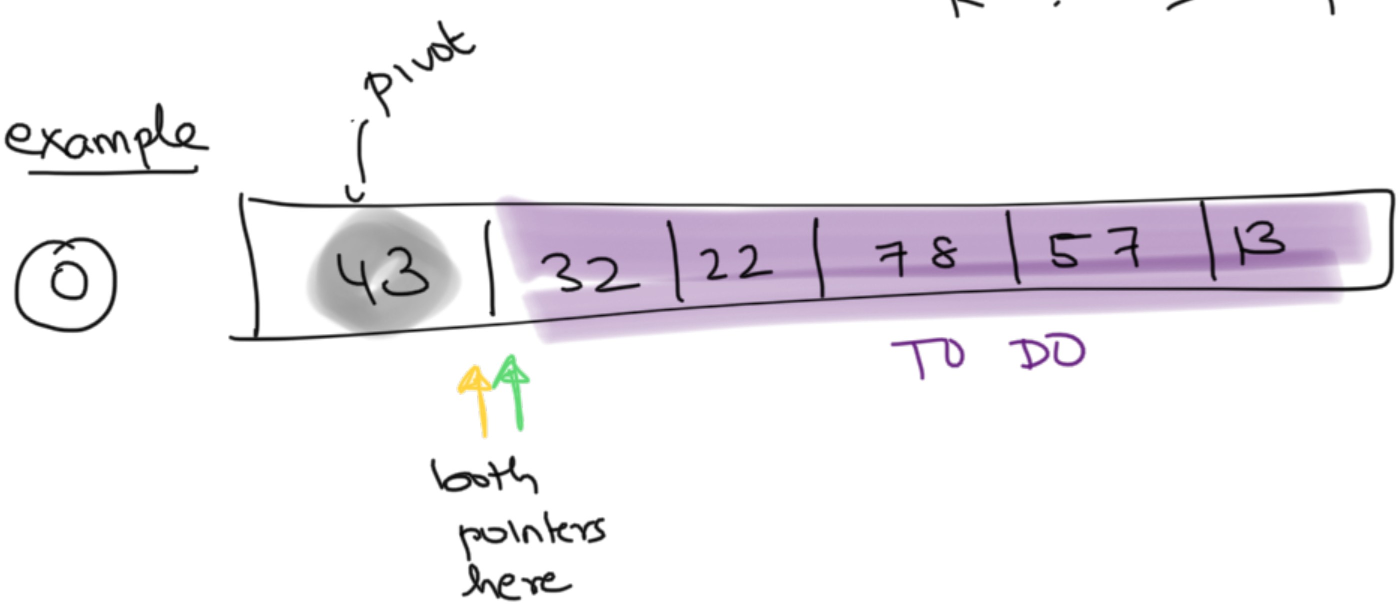
partition the list into

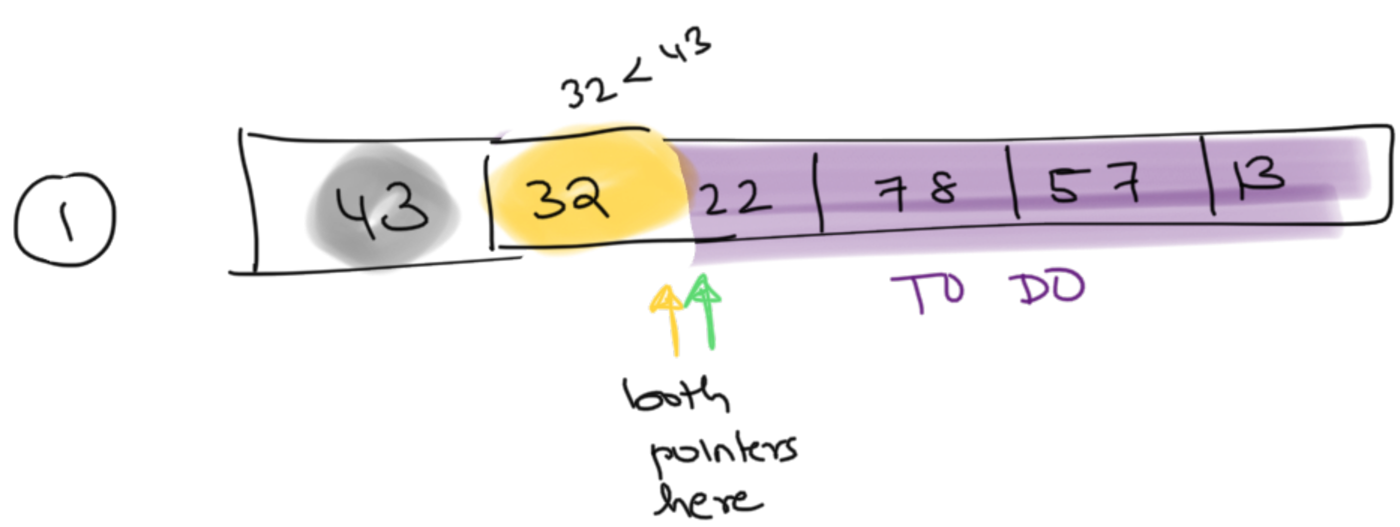
L̃ : < pivot

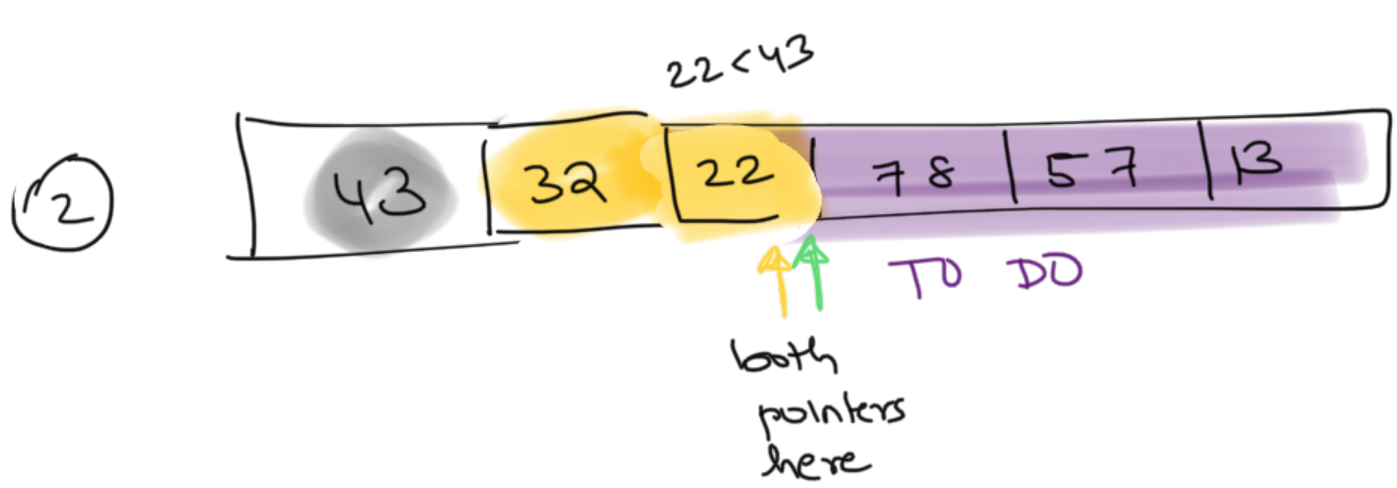
R̃ : > pivot

first

example

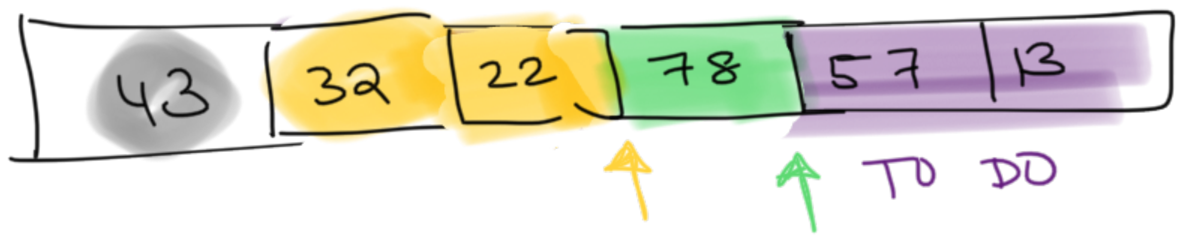






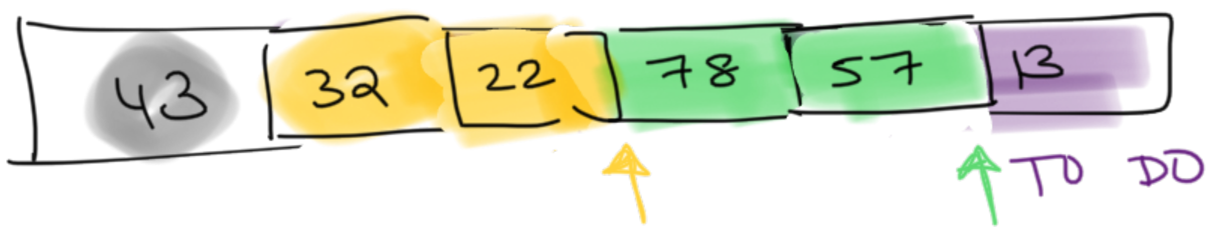
78 > 43

3



57 > 43

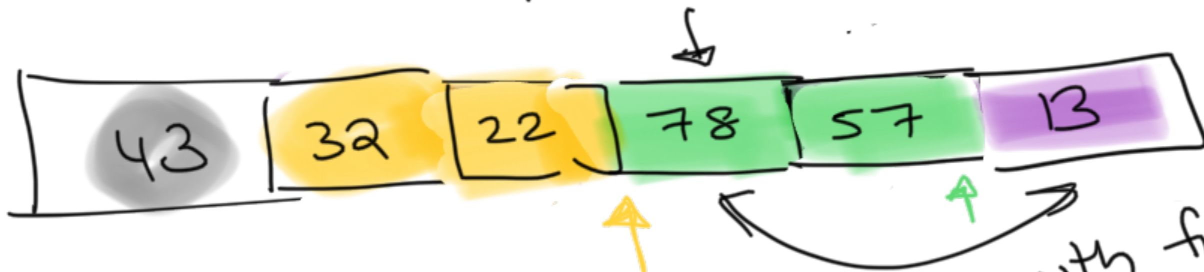
4



78 > 43

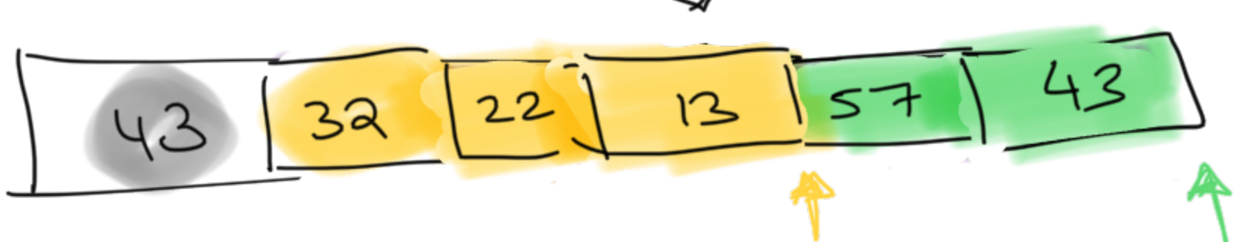
13 < 43

5



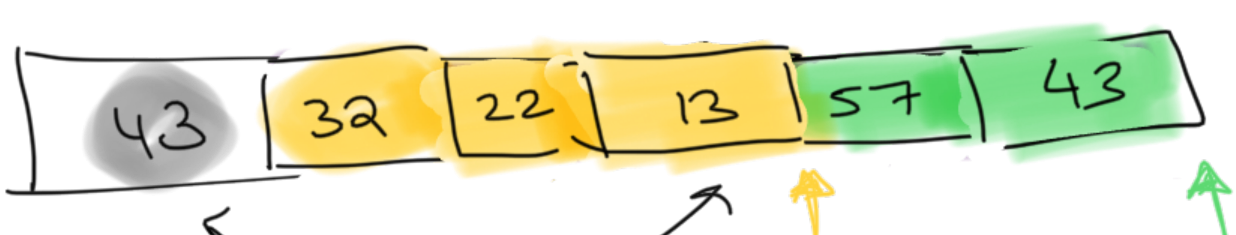
swap with first green

and move pointers

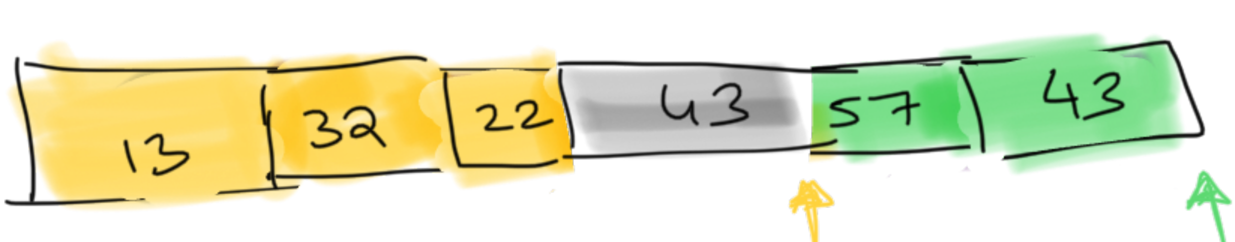


---

13) move pivot in bet L and R

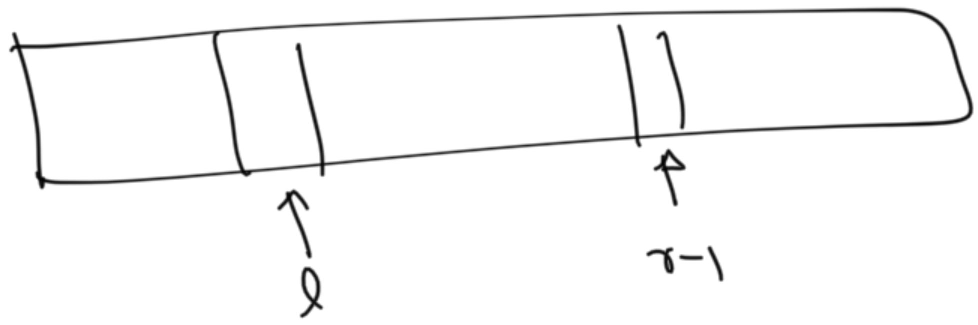


swap with last yellow

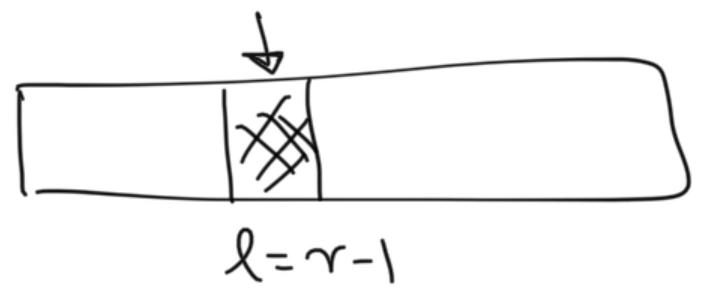


Implementation

→ Quick sort from l to $r-1$ portion of L

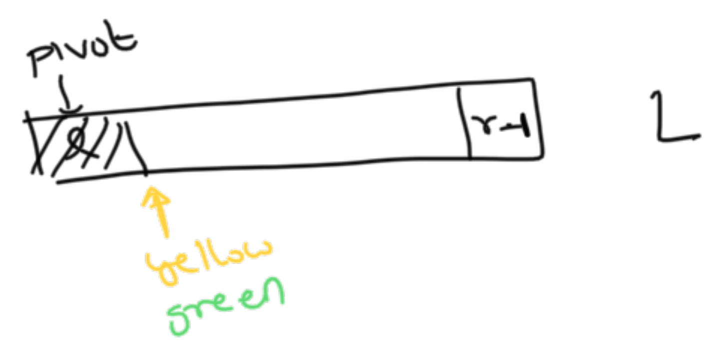


→ Base case



do nothing.

L = list




```
Quick sort (l, r)
{
    if r - l ≤ 1 ;
        return .
    pivot = L[l]
    yellow = l+1
    for l+1 ≤ green < r :
        if L[green] <= pivot :
            \\ swap L[yellow], L[green]
            yellow = yellow + 1

    swap L[l], L[yellow-1]

    quick sort (l, yellow)
    quick sort (yellow+1, r)
}
```

first green

L[yellow] = first green = elt after last yellow

$\underbrace{L[l]}_{\text{pivot}}$, $\underbrace{L[yellow-1]}_{\text{last yellow}}$

Complexity

$n = |L|$

→ Partitioning into $\hat{L}$, $\hat{R}$: $O(n)$,

→ If pivot is median (each time)

$$T(n) = 2T(n/2) + O(n)$$

$$\rightsquigarrow O(n \log n)$$

→ worst case if pivot is NOT median .... ?

$$\text{pivot} = \text{min or max } (L)$$

then

$$\hat{L} = \{\text{pivot}\}, \quad R = L \setminus \{\text{pivot}\} \quad !!$$

so

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

(each time if pivot = min/max)

$$= T(n-2) + n-1 + n$$

$$\vdots$$

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

So if L sorted already, quick sort takes $O(n^2)$ to re-sort it for example.

Amortized analysis

: "on an average", quicksort takes $O(n \log n)$ time

→ Permutations $([1, 2, \dots n]) \rightsquigarrow n!$

→ each input is equally likely ($\frac{1}{n!}$ probability)

→ Expected running time for quick sort is $O(n \log n)$

Randomized Quick sort

→ How to avoid "worst case" scenario in quick sort?

→ Pick pivot randomly! (pick any index in [0, 1, ..., n-1] with uniform probability and choose pivot = L[index])

→ $O(n \log n)$ expected time again

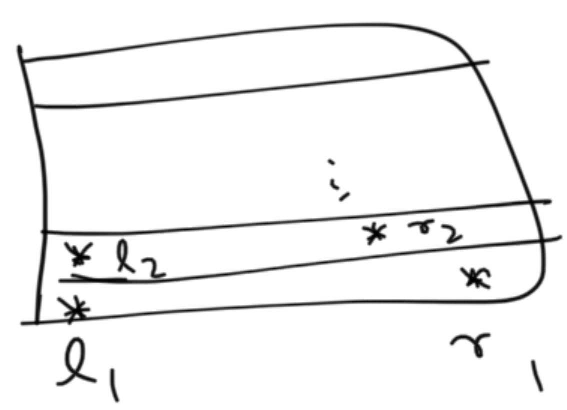
→ Quick sort - usually algo for built in sort

→ usually very fast

→ can be made "iterative" (instead of recursive)

$$\hat{L} \cap \hat{R} = \phi$$

→ maintain explicit stack



of $[l_i, r_i]$ to be sorted.