10 February 2026

Quicksort!

* randomized QS (typically what is implemented) has worst-case runtime of $\Theta(n^2)$ versus other sorting algos w/ worst-case runtime of $\Theta(n \lg n)$
* randomized QS is typically a very fast sorting algo in practice.
* can be done in-place (aka - does not require extra memory)

```
QUICKSORT(A[1..n]):
1:  if (n > 1)
2:      Choose a pivot element A[p]
3:      r ← PARTITION(A, p)
4:      QUICKSORT(A[1..r−1])      ⟨⟨Recurse!⟩⟩
5:      QUICKSORT(A[r+1..n])      ⟨⟨Recurse!⟩⟩
```

```
PARTITION(A[1..n], p):
6:  swap A[p] ↔ A[n]
7:  ℓ ← 0                         ⟨⟨#items < pivot⟩⟩
8:  for i ← 1 to n−1
9:      if A[i] < A[n]
10:         ℓ ← ℓ + 1
11:         swap A[ℓ] ↔ A[i]
12: swap A[n] ↔ A[ℓ + 1]
13: return ℓ + 1
```

Figure 1.8. Quicksort

(From JE, Ch. 1)

Exercise: walk through two examples:

(1) A= | 15 | 10 | 2 | 6 | 7 | 14 | 9 | 1 | 8 | 11 | 12 | 13 | 3 | 4 | 5 |

→ pivot on 2 w/ recursion fairy

→ pivot on 7 w/ recursion fairy

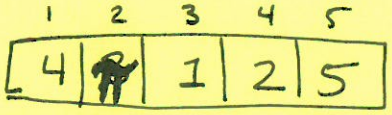
(2) A= | 4 | 3 | 1 | 2 | 5 |

→ pivot on 1 w/out recursion fairy

→ pivot on 3 w/ out recursion fairy

note:
can interpret 2, 7, 1, 3 as p or as A[p].
*intended as A[p]

Example Walk-Through w/out recursion fairy

QuickSort (A = 

) n = 5

variables:

A = 

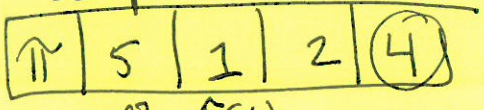
, n = 5

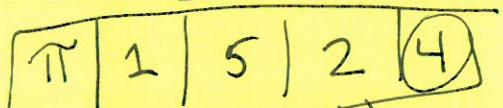
p = 1

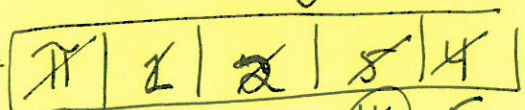
after line 6:

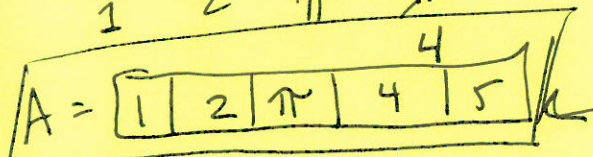
A = 

swap

A = 

A = 

A = 

A = 

l = ~~0 1 2 3~~

i = ~~1 2 3 4~~ 5

only in partition

r = 4

meaning:
A[r] = A[4] is our pivot elt.

Line 1: n > 1 yes; indeed 5 > 1

Line 2: set p = 1

∴ A[p] = A[1] = 4 is our pivot elt.

Line 3: call Partition(A, p=1)

→ Partition(A, p = 1)

Line 6: swap A[p] ↔ A[5]

Line 10': A[i] < A[5]

A[1] < A[5]

5 < 4 NO

10: A[2] < A[5] = 4 ✓

swap A[l] • A[5]

10: A[3] < A[5] = 4 ✓

swap

2 = A[4] < A[5] = 4 ✓

swap

Line 12: swap A[5] • A[l+1]

|| A[4]

return l+1

Back in QS:

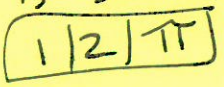
Line 4: Quicksort (

)

recursive call.

after QS Line 4,

A[1...3] is sorted

A[1...3] = 

Line 5: Quicksort A[4+1....5]

|| A[5]

n > 1 is false. Returns right away. (3)

Quicksort (π | 1 | 2 | 4)

These are the 1st 4 elts pres. A. variables:

A = π | 1 | 2 | 4

n = 4

p = 1

Line 1: ✓

Line 2: Pivot on p = 1

A[p=1] = π

Line 3:

Partition (A[1:4], 1)

1st time through for loop

A[1] < A[4] = π

NO

2nd time through

A[2] < π yes

3rd time through

A[3] < π yes

swap A[ℓ=2] and A[i=3]

Line 12: Swap A[4] ⟷ A[3]

return ℓ+1 = 2+1 = 3

ℓ = ~~0~~ ~~1~~ 2

i = ~~1~~ ~~2~~ ~~3~~ 4

r = 3

r is now 3

Line 4: Quicksort (A[1 ... 2])

=

1 | 2

result: A = 1 | 2

Line 5: Quicksort (A[4 ... 4])

=

A = 4

returns sorting A[4].

If pivot poorly chosen, (e.g., largest #)

$$T(n) = T(n-1) + T(0) + \Theta(n)$$
$$= T(n-1) + \Theta(n) \quad \in \Theta(n^2)$$

If wisely chosen, "evenly" splits

$$T(n) = T(n/2) + T(n/2) + \Theta(n)$$
$$= 2T(n/2) + \Theta(n) \quad \in \Theta(n \log n)$$

note: we can always pick a nice pivot in $\Theta(n)$ time.

via Quick select.