# General Principle for Sorting Runtime

$\Theta(N) \leq$ best case

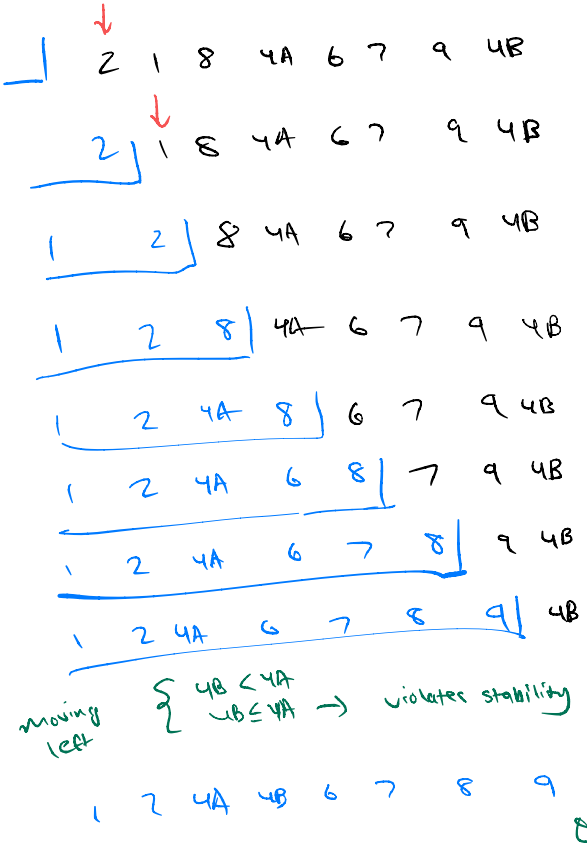
worst case $\leq \Theta(N^2)$ *

* for good sorts

# 1. Step by Step Sorts

2 1 8 4A 6 7 9 4B

## 1. Insertion Sort




Stability:

1A 2 3 1B

1B 1A 2 3

↳ not stable

Inversion:

a b

## Insertion Sort

Runtime:

| | Best | Worst | Stability |
|---|---|---|---|
| | 1 2 3 4 5 | 5 4 3 2 1 | YES |
| | 1 2 3 4 5 | 1 2 3 4 5 | |
| | $\Theta(N)$ | $\Theta(N^2)$ | |

# Selection Sort

every loop:

↳ pick optimal element

iterates through every value and pick "best" number

2 1 8 4A 6 7 9 4B

1 | 2 8 4A 6 7 9 4B

1 2 | 8 4A 6 7 9 4B

1 2 4A | 8 6 7 9 4B

1 2 4A 4B | 8 6 7 9

1 2 4A 4B 6 | 8 7 9

1 2 4A 4B 6 7 | 8 9

1 2 4A 4B 6 7 8 | 9

1 2 4A 4B 6 7 8 9

N elements

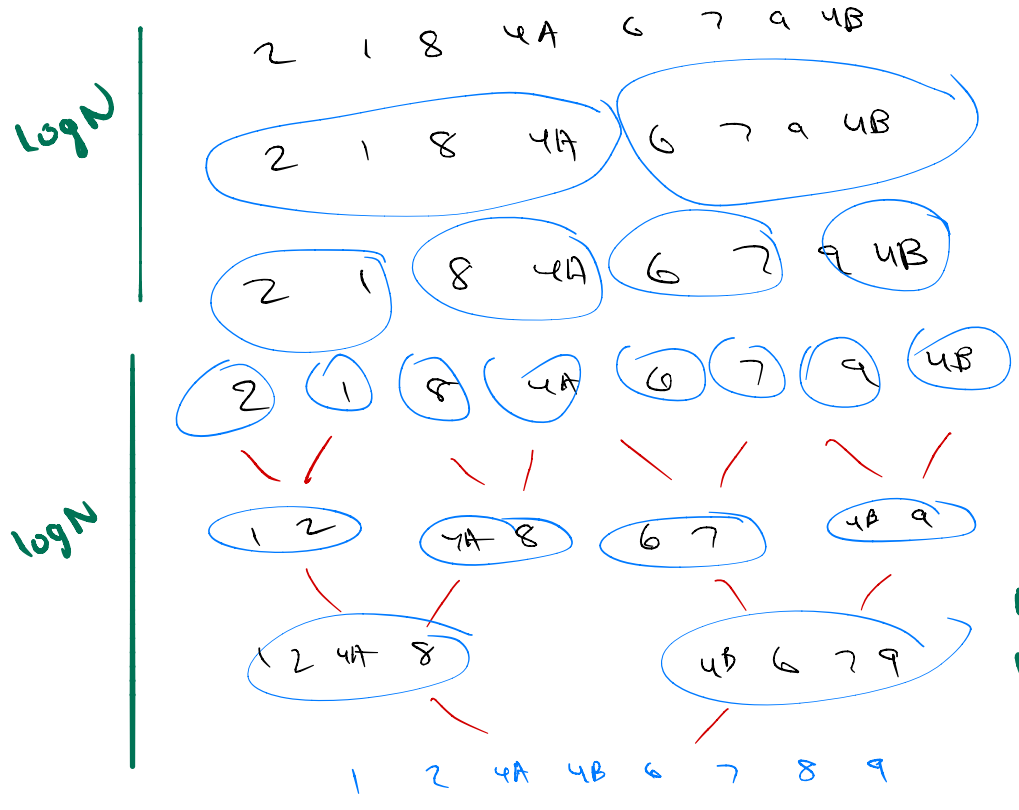
N picks { During each pick: $\Theta(N)$

$\Theta(N \cdot N) = \Theta(N^2)$

**Runtime:**

| Best | Worst | Stability |
|---|---|---|
| $\Theta(N^2)$ | $\Theta(N^2)$ | Depends |

# Merge Sort → recursive




★ at every level, merge() hit every element O(N)

---

merge()

A 4A 7 B 4B 8

defaults to left value first

↳ 4A 4B 7 8

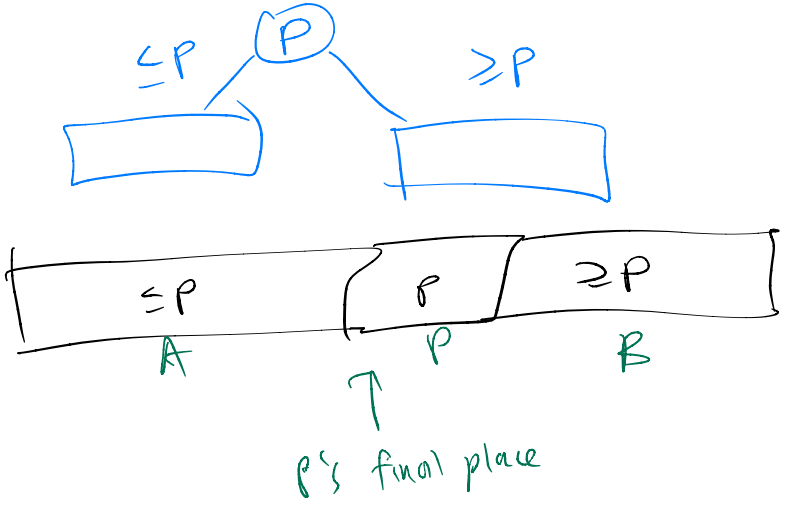
only works if A and B are sorted

Runtime?

| Best | Worst | Stability |
| --- | --- | --- |
| $\Theta(N \log N)$ | $\Theta(N \log N)$ | YES |

# Quicksort → recursive


pivot : choose one element based on a rule




Quicksort(A) → sorted

Quicksort(B) → sorted

p

Final sorted array


Work per level: $\Theta(N)$

↳ "partition" for every element

2 5 1 3 4

2 1 3 5 4

| | Best | Worst | Stability |
|---|---|---|---|
| Runtime: | $\Omega(N \log N)$ | $\Theta(N^2)$ | Depends |

5 4 3 2 1

* fastest sort

# 3. You Choose

1. Almost / already sorted

2. 4 3 2 1

3.
   - Stability
   - picking a bad pivot ⇒ $\Theta(N^2)$
   - highly parallelizable

4. Quicksort — comparing to pivot

   Mergesort — once merged()

   Insertion — sorted elements only