🎣

Quicksort

📘 Quicksort & Sorting Properties — Exam Notes

Desirable Properties of Sorting Algorithmslecture03_quicksort

- **Optimal**:
  - From decision tree proof: any comparison-based sort needs Ω(n log n).
  - Algorithms with Θ(n log n) are called *optimal*.
  - Merge sort & heapsort are always optimal; quicksort is optimal **in expected case**.
- **In-place**:
  - Uses only O(1)–O(log n) extra memory.
  - Insertion sort ✅, heapsort ✅, merge sort ❌ (needs O(n) buffer).
  - Quicksort ✅ (only swaps inside array, plus recursion stack = O(log n) avg).
- **Stable**:
  - Preserves relative order of equal elements.
  - Insertion sort ✅, merge sort ✅, heapsort ❌, quicksort ❌ (partition swaps break order).

Properties Recap Table

Algorithm	Optimal ($\Theta(n \log n)$)	In-Place	Stable
Insertion	❌ ($O(n^2)$ worst)	✅	✅
Merge	✅	❌	✅
Heap	✅	✅	❌
Quicksort	✅ (expected case)	✅	❌

Quicksort — Overview

- **Divide & Conquer** like merge sort, but the work is done in a different phase.
- **Partition step**:
  - Choose a pivot.
  - Rearrange array so all ≤ pivot are left, all > pivot are right.
  - This removes all **cross-pivot inversions** in one step.
- Recursively sort left and right parts until trivial.
- No merge needed — subarrays end up sorted in place.

Why It Workslecture03_quicksort

- Sorting = removing inversions.
- Partitioning guarantees no inversions between left & right of pivot.
- By induction: recursive sorts remove inversions *within* each half.
- Eventually arrays shrink to size 1 → trivially sorted.

Implementation Sketch

```
def partition(xs, lwr, upr):
    pivot = xs[upr-1]
    mid = lwr
    for i in range(lwr, upr):
        if xs[i] <= pivot:
```

```
        xs[mid], xs[i] = xs[i], xs[mid]
        mid += 1
    return mid - 1  # pivot's final index

def quicksort(xs, lwr=0, upr=None):
    if upr is None: upr = len(xs)
    if upr - lwr > 1:
        mid = partition(xs, lwr, upr)
        quicksort(xs, lwr, mid)      # sort left
        quicksort(xs, mid+1, upr)    # sort right
```

Complexity Analysis

- **Partition step**: O(n) (linear scan + swaps).
- **Worst Case** (pivot = smallest or largest each time, e.g. sorted/reversed input):
  - Partition leaves one side empty, other side size n-1.
  - Recurrence: n + (n-1) + ... + 1 = $O(n^2)$.
- **Best Case** (pivot always median):
  - Balanced split each time → recursion depth log n.
  - Each level processes n elements.
  - Total O(n log n).
  - But: finding true median is expensive, so not guaranteed.
- **Expected Case** (random inputs, random pivot):
  - "Good split" means pivot lands not too skewed (e.g. 1:3 – 3:1 ratio).
  - This happens with constant probability (≈ ½).
  - If we got our desired split every time, the recursion would have depth ≈ $\log_{4/3} N$
  - ≈ $2\log_{4/3} N$

- Expected recursion depth still O(log n).
- Total expected complexity = O(n log n).
- ⇒ Quicksort is *optimal on average*.

Memory Usage

- In-place by CLRS definition (swaps inside array).
- Recursion stack:
  - Avg = O(log n).
  - Worst = O(n) if recursion very unbalanced.
  - But if we **always recurse on the smaller half first**, stack depth = O(log n).

Stability

- Quicksort is **not stable**.
- During partition, swapping `x ≤ p` with some `y > p` can leapfrog `y` over equal elements.
- Relative order of equal values can be broken.

Practical Notes

- Quicksort is often **faster in practice** than merge/heap sort due to cache efficiency and low overhead.
- Pivot choice matters:
  - Fixed pivot (like last element) → adversarial worst case on sorted/reversed input.
  - Random pivot or "median-of-3/5" pivot choice greatly improves chance of balanced splits.
- Real-world: many libraries use **Introsort** (quicksort but fallback to heapsort if recursion too deep) to guarantee O(n log n) worst caselecture03_quicksort.

✅ Quick Quicksort Cheat Notes

- *Divide & conquer* but partition first.
- Removes cross-pivot inversions in bulk.
- **Worst = $O(n^2)$** (bad pivot choices).
- **Expected = $O(n \log n)$** (random pivot → balanced on average).
- **In-place**, **not stable**, **optimal on average**.