

Course Algorithms and Data Structures · Lecture 11

Advanced Sorting: Merge Sort and Quick Sort

Textbook: A. Levitin, Sections 5.1, 5.2, 4.5

Divide and conquer

1. Divide

split the problem into smaller problems of the same kind

2. Conquer

solve them recursively

3. Combine

join their solutions

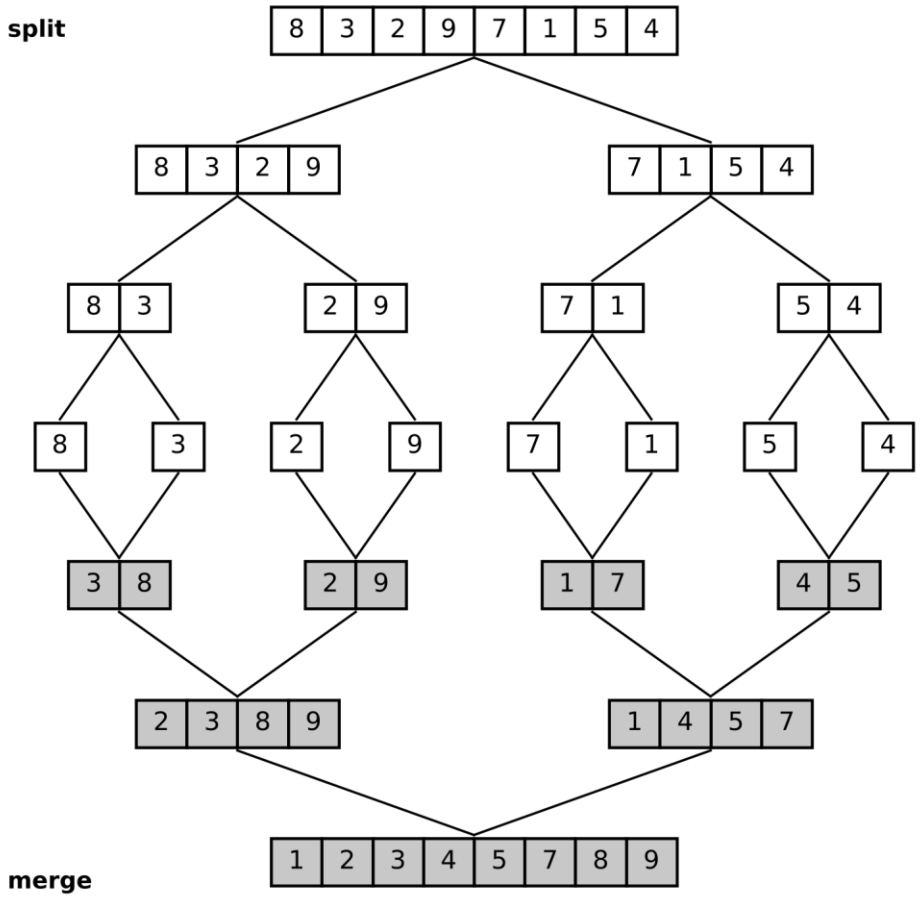
- **Merge sort**: divides trivially, works hard when combining
- **Quick sort**: works hard when dividing, no combining needed

Merge sort

Split the array in half, sort each half recursively, then **merge** the two sorted halves.

```
ALGORITHM Mergesort(A[0..n − 1])
    if n > 1
        copy A[0..⌊n/2⌋ − 1] to B
        copy A[⌊n/2⌋..n − 1] to C
        Mergesort(B)
        Mergesort(C)
        Merge(B, C, A)
```

Merge sort on 8, 3, 2, 9, 7, 1, 5, 4



- Split down to single elements
- Merge back up: grey arrays are sorted
- About $\log_2 n$ levels

Merging two sorted arrays

```
ALGORITHM Merge(B[0..p − 1], C[0..q − 1], A)
    i ← 0;  j ← 0;  k ← 0
    while i < p and j < q do
        if B[i] ≤ C[j]
            A[k] ← B[i];  i ← i + 1
        else
            A[k] ← C[j];  j ← j + 1
        k ← k + 1
    copy the rest of B or C to A
```

Merging 2, 3, 8, 9 and 1, 4, 5, 7

Compare	To A	A so far
2 and 1	1	1
2 and 4	2	1 2
3 and 4	3	1 2 3
8 and 4	4	1 2 3 4
8 and 5	5	1 2 3 4 5
8 and 7	7	1 2 3 4 5 7
C is empty	8, 9	1 2 3 4 5 7 8 9

Each comparison places one element: merging is O(n).

How fast is merge sort?

$$\log_2 n \text{ levels} \times O(n) \text{ per level} = O(n \log n)$$

$$C(n) = 2\,C(n/2) + n,\ C(1) = 0$$

- O(n log n) in **every** case: best, worst, average
- Needs **O(n) extra memory** for B and C
- **Stable**: “≤” takes equal elements from B first

Quick sort

Choose a **pivot**. Partition: smaller elements before it, the others after it. The pivot is now in its final place. Sort both parts recursively.

```
ALGORITHM Quicksort(A[l..r])
    if l < r
        s ← Partition(A[l..r])   //pivot's final position
        Quicksort(A[l..s − 1])
        Quicksort(A[s + 1..r])
```

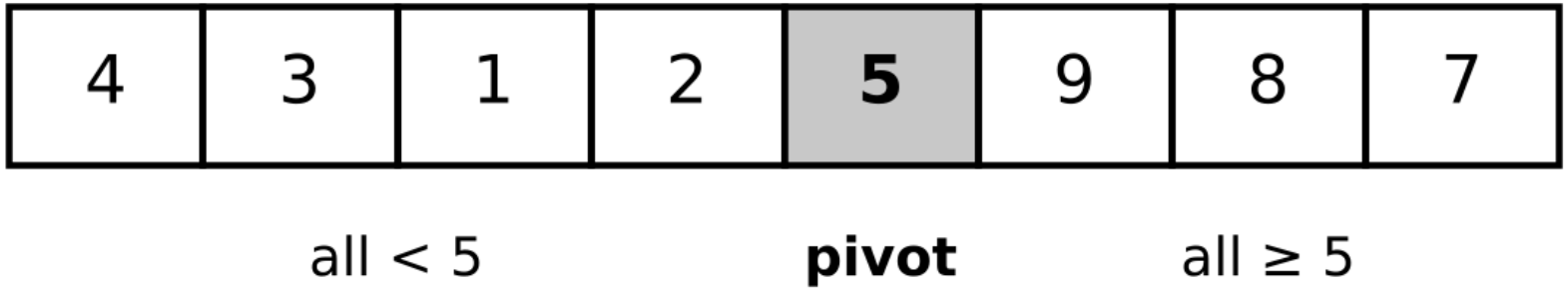
Partition (Lomuto, Levitin 4.5)

```
ALGORITHM Partition(A[l..r])
    p ← A[l]                //the pivot
    s ← l
    for i ← l + 1 to r do
        if A[i] < p
            s ← s + 1
            swap A[s] and A[i]
    swap A[l] and A[s]
    return s
```

Partition of 5, 3, 1, 9, 8, 2, 4, 7 (pivot 5)

i	A[i]	< 5?	Action	s	Array
			start	0	5 3 1 9 8 2 4 7
1	3	yes	s ← 1	1	5 3 1 9 8 2 4 7
2	1	yes	s ← 2	2	5 3 1 9 8 2 4 7
3, 4	9, 8	no		2	5 3 1 9 8 2 4 7
5	2	yes	s ← 3, swap 9, 2	3	5 3 1 2 8 9 4 7
6	4	yes	s ← 4, swap 8, 4	4	5 3 1 2 4 9 8 7
7	7	no		4	5 3 1 2 4 9 8 7
			swap pivot, A[4]	4	4 3 1 2 5 9 8 7

After partition



5 never moves again. Quick sort continues with 4, 3, 1, 2 and 9, 8, 7 separately. Partition is O(n).

How fast is quick sort?

Case	When	Time
Best	the pivot lands in the middle	O(n log n)
Average	random data	O(n log n)
Worst	the pivot is always the smallest: e.g., a sorted array	$O(n^2)$

Fix: a **random** pivot or the **median of three**. In place, not stable, usually the fastest in practice.

Comparison

	Merge sort	Quick sort	Insertion sort
Best	O(n log n)	O(n log n)	O(n)
Average	O(n log n)	O(n log n)	$O(n^2)$
Worst	O(n log n)	$O(n^2)$	$O(n^2)$
Extra memory	O(n)	O(log n) stack	O(1)
Stable	yes	no	yes

Python’s sort mixes merge sort and insertion sort; C++ std::sort is based on quick sort.

Summary

- Divide and conquer: split, solve the parts recursively, combine.
- Merge sort is always O(n log n) but needs O(n) extra memory.
- Quick sort is O(n log n) on average, in place and usually fastest, but $O(n^2)$ in the worst case.
- A random or median-of-three pivot makes the worst case very unlikely.

Review questions

1. What are the three steps of divide and conquer? How do merge sort and quick sort use them?
2. Merge the sorted arrays 1, 6, 9 and 2, 3, 8. Write the result after each comparison.
3. Why is quick sort slow on an already sorted array if the first element is the pivot?

Reading: Levitin, Sections 5.1, 5.2, 4.5; Bhargava, Chapter 4.