

Exercise walk-through

10.2

Divide and conquer

University Computer Science

Divide and conquer: price the combine step

- 1 **Merge** $2T(n/2) + n = n \log n$; binary search is the degenerate case $T(n/2) + 1 = \log n$ - one branch still counts.
- 2 **Quick** Balanced: $n \log n$; sorted input with first-element pivots: $T(n-1) + n = n^2$; measured 310 vs 356 vs worst 2080 comparisons.
- 3 **Subarray** Left best, right best, or the crossing best: the crossing scan is a full pass - $\Theta(n \log n)$; course answer 6.

Recursion structure

the same skeleton, three combine costs

merge sort split in half recurse on the parts $2T(n/2) + n$ $\Theta(n \log n)$	quicksort (balanced) split in half recurse on the parts $2T(n/2) + n$ $\Theta(n \log n)$	binary search split in half recurse on the parts $T(n/2) + 1$ $\Theta(\log n)$
---	---	---

quicksort unbalanced: $T(n) = T(n-1) + n = \Theta(n^2)$ on sorted input — the pivot decides

course measurements: merge 310 comparisons, quicksort 356, worst case 2080 (all on the 64-element array)

one branch is still the pattern: binary search splits, recurses once, combines in $O(1)$

The same skeleton with three combine costs - three different bills.

Exercises 2.1-2.3

2.1 Name the combine cost of merge sort. [3]

2.2 Why does binary search still count as divide and conquer? [3]

2.3 What does the maximum-subarray crossing scan cost? [3]

Exercises 2.1-2.3

2.1 Name the combine cost of merge sort. [3]

2.1

- The merge: $\Theta(n)$ per level ✓.

2.2 Why does binary search still count as divide and conquer? [3]

2.2

- It splits, recurses (once), and combines in $O(1)$ – one branch is still the pattern ✓.

2.3 What does the maximum-subarray crossing scan cost? [3]

2.3

- A full pass over the half on each side: $\Theta(n)$ ✓.

Exercise 3.1

3.1 Maximum subarray on $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$ by divide and conquer.

State the three candidates at the top level (left best, right best, crossing) and compute each. Give the winning subarray and sum. State the recurrence and its solution, and why the crossing candidate forces the log factor. [8]

Exercise 3.1

3.1 Maximum subarray on $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$ by divide and conquer.

State the three candidates at the top level (left best, right best, crossing) and compute each. Give the winning subarray and sum. State the recurrence and its solution, and why the crossing candidate forces the log factor. [8]

3.1

- Split at the midpoint: left $[-2, 1, -3, 4]$, right $[-1, 2, 1, -5, 4]$ ✓.
- Left best: 4 (the single element). Right best: $2 + 1 = 3$ ✓.
- Crossing: best left suffix 4, best right prefix $2 + 1 = 3$, crossing total $4 + (-1) + 2 + 1 = 6$ – winner 6, subarray $[4, -1, 2, 1]$ ✓.
- Recurrence $2T(n/2) + \Theta(n) = \Theta(n \log n)$ – the crossing scan is a full pass, so the per-level work stays linear ✓.

Exercise 3.2

3.2 Predict, then explain, quicksort's comparison counts.

For $n = 64$: the balanced expectation $2n \ln n$ (compute the value), the measured first-element count on the course array (356), and the sorted-input worst $\sum_{k=1}^n (k-1)$ (compute the value). State which recurrence models the worst case and why randomised pivots change the expectation, not the worst case. [7]

Exercise 3.2

3.2 Predict, then explain, quicksort's comparison counts.

For $n = 64$: the balanced expectation $2n \ln n$ (compute the value), the measured first-element count on the course array (356), and the sorted-input worst $\sum_{k=1}^n (k-1)$ (compute the value). State which recurrence models the worst case and why randomised pivots change the expectation, not the worst case. [7]

3.2

- Balanced expectation: $2 \times 64 \times \ln 64 \approx 2 \times 64 \times 4.159 \approx 532$ ✓.
- Measured first-element on the course array: 356 – the array is not adversarial ✓.
- Sorted worst: $\sum_{k=1}^{64} (k-1) = 64 \times 63/2 = 2016$ comparisons; the earlier figure 2080 counts the final pass's extra compares – quote 2016 for the sum formula, 2080 measured ✓.
- Worst-case recurrence: $T(n) = T(n-1) + \Theta(n)$ – each partition peels exactly one element ✓.
- Randomised pivots make every input behave like a random one in expectation – the worst case remains possible but no longer adversarially triggerable ✓.