465 Homework 3

Kyle Renner

July 2025

Problem 1

(a) Sequential Merging Approach

We merge the first two arrays, then merge that result with the third, then the fourth, and so on until all m arrays (each of size n) are merged into a single array.

Each merge between arrays of sizes a and b takes $O(a+b)$ time. Since we're always merging with a new array of size n, and the merged array grows each time, the total time is:

- First merge: $n + n = 2n$ elements $\Rightarrow O(2n)$
- Second merge: $2n + n = 3n$ elements $\Rightarrow O(3n)$
- Third merge: $3n + n = 4n$ elements $\Rightarrow O(4n)$
- $\ldots$
- Final merge: $(m-1)n + n = mn$ elements $\Rightarrow O(mn)$

This results in total time:

$$O(n(2 + 3 + \cdots + m)) = O(n \cdot \frac{m(m+1)}{2} - n) = O(nm^2)$$

(b) Divide-and-Conquer Merge

We can divide the m arrays into two halves recursively, merge each half, and then merge the two results.

This is similar to merge sort:

- Total number of elements across all arrays is mn.
- At each level of recursion, merging all arrays takes $O(mn)$ time.
- There are $\log_2 m$ levels (since we divide the m arrays in half each time).

Therefore, total time is:

$$O(mn \log m)$$

This is significantly faster than the $O(nm^2)$ brute-force approach in part (a).

Problem 2

(a) $T(n) = 8T(n/2) + 100n^3$

Using Master Theorem:

- $a = 8,\ b = 2,\ f(n) = \Theta(n^3)$
- $\log_b a = \log_2 8 = 3$
- $f(n) = \Theta(n^{\log_b a}) \Rightarrow$ Case 2

$$T(n) = \Theta(n^3 \log n)$$

(b) $T(n) = 8T(n/2) + 1000n^{3.5}$

- $a = 8,\ b = 2,\ f(n) = \Theta(n^{3.5})$
- $\log_b a = 3 < 3.5 \Rightarrow$ Case 3

$$T(n) = \Theta(n^{3.5})$$

(c) $T(n) = 16T(n/2) + n \log n$

- $a = 16,\ b = 2,\ f(n) = n \log n$
- $\log_b a = \log_2 16 = 4$
- $f(n) = o(n^4) \Rightarrow$ Case 1

$$T(n) = \Theta(n^4)$$

(d) $T(n) = 2T(n/2) + n \log n$

- $a = 2,\ b = 2,\ \log_b a = 1$
- $f(n) = \Theta(n \log n) = \Theta(n^{\log_b a} \log n) \Rightarrow$ Case 2

$$T(n) = \Theta(n \log^2 n)$$

(e) $T(n) = 8T(n/2) + n^{3.5} \log^2 n$

- $a = 8,\ b = 2,\ \log_b a = 3$
- $f(n) = \Omega(n^{3+\varepsilon})$ for $\varepsilon = 0.5$
- Regularity condition holds

$$T(n) = \Theta(n^{3.5} \log^2 n)$$

(f) $T(n) = T(n/2) + 1.5n$

Unrolling gives:

$$T(n) = T(n/2) + 1.5n = T(n/4) + 1.5n/2 + 1.5n = \cdots \Rightarrow T(n) = \Theta(n)$$

(g) $T(n) = T(3n/5) + T(2n/5) + O(n)$

We guess $T(n) = O(n \log n)$ and prove by induction:
Assume $T(k) \leq c \cdot k \log k$ for all $k < n$.

$$T(n) \leq c \cdot \frac{3n}{5} \log \frac{3n}{5} + c \cdot \frac{2n}{5} \log \frac{2n}{5} + cn$$

That simplifies to:

$$T(n) \leq cn(\frac{3}{5} \log \frac{3}{5} + \frac{2}{5} \log \frac{2}{5} + \log n) + cn$$

The constant terms are bounded, so $T(n) = O(n \log n)$.

(h) $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + 10n$

Let $n = 2^{2^k} \Rightarrow \log \log n = k$
Unrolling gives:

$$T(n) = n^{1/2} \cdot n^{1/4} \cdot n^{1/8} \cdots = n^{\sum 1/2^i} = n^1 = n$$

But we do this for $\log \log n$ levels, each adding $10n$ work:

$$T(n) = O(n \log \log n)$$