Algorithms and Data Structures 2
Exam Notes
Week 1: Divide-and-Conquer

Mads Richardt

1 General Methodology and Theory

Divide-and-Conquer Strategy

1. Divide problem into smaller subproblems (often of equal size).
2. Conquer each subproblem recursively.
3. Combine subproblem solutions into a full solution.

Recurrence Relations

General form for divide-and-conquer running times:

$$T(n) = q \cdot T\left(\frac{n}{b}\right) + f(n),$$

where

- q: number of subproblems,
- b: factor by which input size is reduced,
- $f(n)$: cost to divide and combine.

Solving Recurrences

Recursion Tree Method.

1. Expand recurrence level by level.
2. Compute cost per level.
3. Sum over all levels until base case.

Substitution Method.

1. Guess solution form $T(n) \leq k \cdot g(n)$.
2. Prove by induction:
   - Base case holds.
   - Inductive step: Plug hypothesis into recurrence.

Useful Mathematical Tools

- **Geometric series:** for $x \neq 1$,
$$\sum_{i=0}^{m} x^i = \frac{x^{m+1}-1}{x-1}.$$
For $|x| < 1$,
$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}.$$
- **Logarithm rules:**
$$\log_a b = \frac{\ln b}{\ln a}, \quad \log(ab) = \log a + \log b, \quad \log\frac{a}{b} = \log a - \log b.$$

2 Notes from Slides and Textbook

Mergesort Recurrence

$$T(n) \leq \begin{cases} 2T(n/2) + cn & n > 2, \\ c & n \leq 2. \end{cases}$$

Recursion tree analysis: Each level costs cn. There are $\log_2 n$ levels. Total:

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

Substitution: Guess $T(n) \leq kn \log n$. Show inductively:

$$T(n) \leq 2k\frac{n}{2}\log(n/2) + cn = kn\log n - kn + cn \leq kn \log n.$$

Counting Inversions

- Inversion: pair (i, j) with $i < j$ and $a_i > a_j$.
- Divide-and-conquer algorithm: Sort-and-Count using merge.
- Running time: $O(n \log n)$.

3 Solutions to Problem Set

Exercise 1: Recurrences I

(a) $T(n) \leq 2T(n/4) + cn$.

$$T(n) = 2T\left(\frac{n}{4}\right) + cn.$$

Recursion tree: At level i, 2^i subproblems of size $n/4^i$. Cost per subproblem $\approx c \cdot n/4^i$. Total per level:

$$2^i \cdot \frac{cn}{4^i} = \frac{cn}{2^i}.$$

Summing over $\log_4 n$ levels:

$$T(n) \leq cn \sum_{i=0}^{\log_4 n} \frac{1}{2^i} \leq 2cn = O(n).$$

(b) $T(n) \le 2T(n/4)+c\sqrt{n}$**.** Level i has 2^i subproblems of size $n/4^i$. Cost per subproblem: $c\sqrt{n/4^i} = \frac{c}{2^i}\sqrt{n}$. Total per level:

$$2^i \cdot \frac{c}{2^i}\sqrt{n} = c\sqrt{n}.$$

Depth $\log_4 n$. Total:

$$T(n) = O(\sqrt{n}\log n).$$

Exercise 2: Significant Inversions (KT 4.2)

Use modification of Sort-and-Count:

```
Algorithm CountSignificantInversions(A):
  if length(A) = 1: return (0, A)
  split A into L and R
  (iL, L) := CountSignificantInversions(L)
  (iR, R) := CountSignificantInversions(R)
  (iM, M) := MergeAndCountSignificant(L, R)
  return (iL + iR + iM, M)
```

During merge, when comparing $a_i \in L$ with $a_j \in R$, if $a_i > 2a_j$, then all later elements in L (since sorted) also form significant inversions with a_j. Count efficiently in $O(n)$ per merge. Total complexity: $O(n \log n)$.

Exercise 3: Divide-and-Conquer on Trees (KT 4.6)

Problem: Find a local minimum in a complete binary tree with $n = 2^d - 1$ nodes using $O(\log n)$ probes.

Algorithm:

1. Probe the root.
2. Compare root with its children.
3. Recursively continue into the smaller child.

Because tree height is $\log n$, this uses $O(\log n)$ probes. Correctness: At each step, moving into the smaller neighbor guarantees a local minimum exists along that path.

Exercise 4: Divide-and-Conquer on Grid Graphs (KT 4.7)

Problem: Find a local minimum in an $n \times n$ grid with $O(n)$ probes.

Algorithm:

1. Check middle column for minimum entry x.
2. Compare x with its horizontal neighbors.
3. If x is smaller than both, x is a local minimum.
4. Otherwise recurse into the half-grid containing the smaller neighbor.

Each step reduces size by factor 2, cost $O(n)$ per level, $\log n$ levels. Total: $O(n)$ probes.

Exercise 5: CSES Programming

Missing Number. Sort or use XOR sum trick. XOR all numbers $1, \dots, n$. XOR with input list. Result is missing number. Complexity: $O(n)$.

Distinct Numbers. Insert all into a set. Output set size. Complexity: $O(n \log n)$.

Exercise 6: Recurrences II

(a) $T(n) \le T(3n/4)+cn$**.** *Tree method:* Level i: subproblem size $(3/4)^i n$. Cost per level $\approx c\cdot(3/4)^i n$. Sum over $\log_{4/3} n$ levels:

$$T(n) \le cn \sum_{i=0}^{\log_{4/3} n} \left(\frac{3}{4}\right)^i \le 4cn = O(n).$$

(b) $T(n) \le T(n/2) + T(n/3) + T(n/6) + cn$**.** All subproblem sizes add up to n, so each level cost $\approx cn$. Depth $O(\log n)$. Therefore $T(n) = O(n \log n)$.

4 Summary

- Divide-and-conquer recurrence template: $T(n) = qT(n/b) + f(n)$.
- Geometric sums: essential for solving recurrences.
- Mergesort: $O(n \log n)$.
- Counting inversions: Sort-and-Count in $O(n \log n)$.
- Significant inversions: modify merge, still $O(n \log n)$.
- Local minimum in binary tree: $O(\log n)$ probes.
- Local minimum in grid: $O(n)$ probes.
- Recurrence results:

$$\begin{aligned}
&T(n) = 2T(n/4) + cn \quad \Rightarrow O(n),\\
&T(n) = 2T(n/4) + c\sqrt{n} \quad \Rightarrow O(\sqrt{n}\log n),\\
&T(n) = T(3n/4) + cn \quad \Rightarrow O(n),\\
&T(n) = T(n/2) + T(n/3) + T(n/6) + cn \quad \Rightarrow O(n \log n).
\end{aligned}$$