5 Divide & Conquer

10 November 2025

Prof. Dr. Sebastian Wild

CS566 (Wintersemester 2025–26)
Philipps-Universität Marburg

version `2025-11-10 15:58`

Learning Outcomes

Unit 5: ***Divide & Conquer***

1. Know the steps of the Divide & Conquer paradigm.
2. Be able to solve simple Divide & Conquer recurrences.
3. Be able to design and analyze new algorithms using the Divide & Conquer paradigm.
4. Know the performance characteristics of selection-by-rank algorithms.
5. Know the divide and conquer approaches for integer multiplication, matrix multiplication, finding majority elements, and the closest-pair-of-points problem.

Outline

Divide and conquer

Divide and conquer *idiom* (Latin: *divide et impera*)
to make a group of people disagree and fight with one another
so that they will not join together against one (Merriam-Webster Dictionary)

$\rightsquigarrow$ in politics & algorithms, many independent, small problems are better than one big one!

Divide-and-conquer algorithms:

1. Break problem into smaller, independent subproblems. (Divide!)
2. Recursively solve all subproblems. (Conquer!)
3. Assemble solution for original problem from solutions for subproblems.

Divide and conquer

Divide and conquer *idiom* (Latin: *divide et impera*)
to make a group of people disagree and fight with one another
so that they will not join together against one (Merriam-Webster Dictionary)

$\leadsto$ in politics & algorithms, many independent, small problems are better than one big one!

Divide-and-conquer algorithms:

1. Break problem into smaller, independent subproblems. (Divide!)
2. Recursively solve all subproblems. (Conquer!)
3. Assemble solution for original problem from solutions for subproblems.

Examples:

- Mergesort
- Quicksort
- Binary search
- (arguably) Tower of Hanoi

Clicker Question

Have you seen the *Master Method* before?

- **A** Sure, could apply it blindfolded
- **B** Vaguely remember
- **C** Never heard of it

→ sli.do/cs566

5.1 Divide & Conquer Recurrences

Back-of-the-envelope analysis

- before working out the details of a D&C idea,
  it is often useful to get a quick indication of the resulting performance
  - don't want to waste time on something that's not competitive in the end anyways!
- since D&C is naturally recursive, running time often not obvious
  instead: given by a recursive equation

Back-of-the-envelope analysis

- before working out the details of a D&C idea,
  it is often useful to get a quick indication of the resulting performance
  - don't want to waste time on something that's not competitive in the end anyways!
- since D&C is naturally recursive, running time often not obvious
  instead: given by a recursive equation
- unfortunately, rigorous analysis often tricky
  - Remember mergesort?

$$C(n) = \begin{cases} 0 & n \le 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2n & n \ge 2 \end{cases}$$

$$\rightsquigarrow C(n) = 2n\lfloor \lg(n) \rfloor + 2n - 4 \cdot 2^{\lfloor \lg(n) \rfloor} \quad 😱$$
$$= \Theta(n \log n) \quad 😌$$

Back-of-the-envelope analysis

- before working out the details of a D&C idea,
  it is often useful to get a quick indication of the resulting performance
  - don't want to waste time on something that's not competitive in the end anyways!
- since D&C is naturally recursive, running time often not obvious
  instead: given by a recursive equation
- unfortunately, rigorous analysis often tricky
  - Remember mergesort?

$$C(n) = \begin{cases} 0 & n \le 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2n & n \ge 2 \end{cases}$$

$$\rightsquigarrow C(n) = 2n\lfloor \lg(n) \rfloor + 2n - 4 \cdot 2^{\lfloor \lg(n) \rfloor} \text{ 😱}$$
$$= \Theta(n \log n) \text{ 😌}$$

- the following method works for many typical cases to give the right **order of growth**

The Master Method

Mergesort

- Assume a stereotypical D&C algorithm
  - a recursive calls on (for some constant $a > 0$)
  - subproblems of size n/b (for some constant $b > 1$)
  - with non-recursive "conquer" effort $f(n)$ (for some function $f : \mathbb{R} \to \mathbb{R}$)
  - base case effort d (some constant $d > 0$)

$a = 2$

$b = 2$

$f(n) = 2 \cdot n$

$n = 2 \quad d = 2$

$(n = 1 \rightsquigarrow d = 0)$

The Master Method

- Assume a stereotypical D&C algorithm
  - a recursive calls on (for some constant $a > 0$)
  - subproblems of size n/b (for some constant $b > 1$)
  - with non-recursive "conquer" effort $f(n)$ (for some function $f : \mathbb{R} \to \mathbb{R}$)
  - base case effort d (some constant $d > 0$)

$\rightsquigarrow$ running time $T(n)$ satisfies

$$\boxed{T(n) = \begin{cases} a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \\ d & n \le 1 \end{cases}}$$

n_0 also possible

The Master Method

- Assume a stereotypical D&C algorithm
  - a recursive calls on (for some constant $a > 0$)
  - subproblems of size n/b (for some constant $b > 1$)
  - with non-recursive "conquer" effort $f(n)$ (for some function $f : \mathbb{R} \to \mathbb{R}$)
  - base case effort d (some constant $d > 0$)

$\rightsquigarrow$ running time $T(n)$ satisfies

$$\boxed{T(n) \;=\; \begin{cases} a \cdot T\left(\frac{n}{b}\right) \;+\; f(n) & n > 1 \\ d & n \le 1 \end{cases}}$$

Theorem 5.1 (Master Theorem)

With $c := \log_b(a)$, we have for the above recurrence:

(a) $T(n) = \Theta(n^c)$ if $f(n) = O(n^{c-\varepsilon})$ for constant $\varepsilon > 0$.

(b) $T(n) = \Theta(n^c \log n)$ if $f(n) = \Theta(n^c)$.

(c) $T(n) = \Theta(f(n))$ if $f(n) = \Omega(n^{c+\varepsilon})$ for constant $\varepsilon > 0$ **and** f satisfies the regularity condition $\exists n_0, \alpha < 1\ \forall n \ge n_0 \;:\; a \cdot f\left(\frac{n}{b}\right) \le \alpha f(n)$.

Example: Mergesort

$a = b = 2$ $\qquad$ $f(n) = 2n$

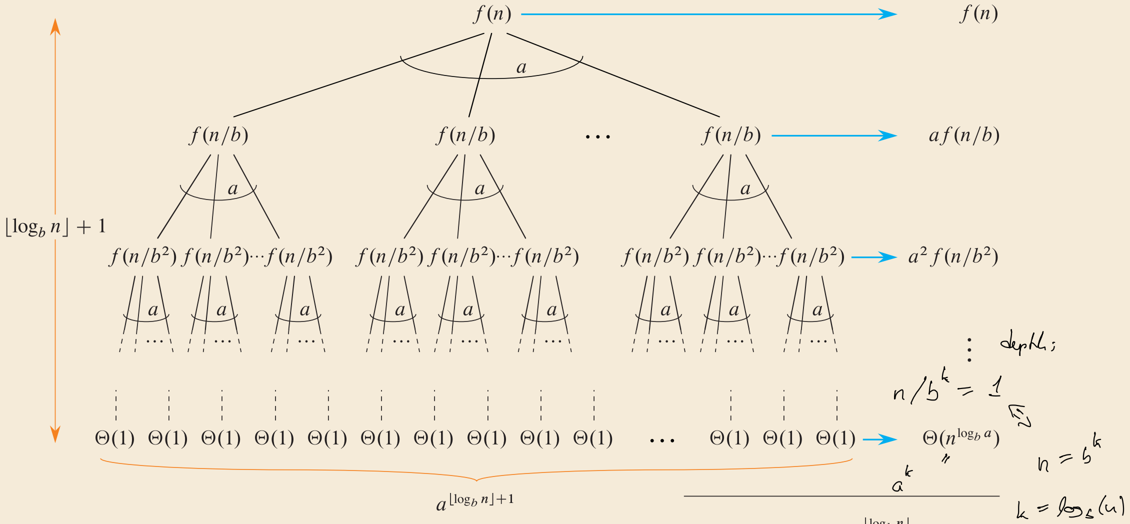
$c = \log_2(2) = 1$

$f(n) = \Theta(n^1) \rightsquigarrow$ case (b)

$\Rightarrow$ sort $\Theta(n \log n)$

MT

Master Theorem – Intuition & Proof Idea



Total: $\Theta(n^{\log_b a}) + \sum_{j=0}^{\lfloor \log_b n \rfloor} a^j f(n/b^j)$

Figure 4.3 of Cormen et al. *Introduction to Algorithms* 4th ed.

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

$$= a\left(a\,T\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b}\right)\right) + f(n)$$

$$\vdots$$

$$= a^k \cdot T(1) + \sum_{j=0}^{k} a^j f\left(\frac{n}{b^j}\right)$$

$k = \log_b(n)$

$$= a^{\log_b(n)} \cdot d + \sum_{j=0}^{\log_b(n)} a^j f\left(\frac{n}{b^j}\right)$$

$$= n^{\log_b(a)} \cdot d + \sum_{j=0}^{\log_b(n)} a^j f\left(\frac{n}{b^j}\right)$$

$$a^{\log_b(n)} = e^{\ln(a)\cdot \ln(n)/\ln(b)}$$

$$= n^{\ln(a)/\ln(b)} = n^{\log_b(a)}$$

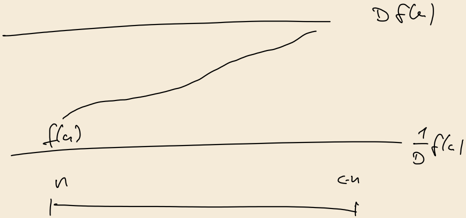
$\boxed{\text{proof not in exam}}$

When it’s fine to ignore floors and ceilings

The ***polynomial-growth condition***

- $f : \mathbb{R}_{>0} \to \mathbb{R}$ satisfies the *polynomial-growth condition* if

$$\exists n_0 \; \forall C \geq 1 \; \exists D > 1 \quad \forall n \geq n_0 \; \forall c \in [1, C] \;\; : \;\; \frac{1}{D} f(n) \;\leq\; f(cn) \;\leq\; D f(n)$$



When it's fine to ignore floors and ceilings

The ***polynomial-growth condition***

- $f : \mathbb{R}_{>0} \to \mathbb{R}$ satisfies the *polynomial-growth condition* if

$$\exists n_0 \; \forall C \ge 1 \; \exists D > 1 \quad \forall n \ge n_0 \; \forall c \in [1, C] \;\; : \;\; \frac{1}{D} f(n) \;\le\; f(cn) \;\le\; D f(n)$$

- intuitively: increasing n by up to a factor C (and anywhere in between!) changes the function value by at most a factor $D = D(C)$ (for sufficiently large n)

- examples: $f(n) = \Theta(n^{\alpha} \log^{\beta}(n) \log\log^{\gamma}(n))$ for constants α, β, γ zero allowed
  $\rightsquigarrow$ f satisfies the polynomial-growth condition

When it's fine to ignore floors and ceilings

The ***polynomial-growth condition***

- $f : \mathbb{R}_{>0} \to \mathbb{R}$ satisfies the *polynomial-growth condition* if

$$\exists n_0 \; \forall C \ge 1 \; \exists D > 1 \quad \forall n \ge n_0 \; \forall c \in [1, C] \;\; : \;\; \frac{1}{D} f(n) \; \le \; f(cn) \; \le \; D f(n)$$

- intuitively: increasing n by up to a factor C (and anywhere in between!) changes the function value by at most a factor $D = D(C)$ (for sufficiently large n)

- examples: $f(n) = \Theta(n^{\alpha} \log^{\beta}(n) \log\log^{\gamma}(n))$ for constants α, β, γ zero allowed
  $\rightsquigarrow$ f satisfies the polynomial-growth condition

Lemma 5.2 (Polynomial-growth master method)

If the toll function $f(n)$ satisfies the polynomial-growth condition, then the Θ-class of the solution of a D&C recurrence remains the same when ignoring floors and ceilings on subproblem sizes.

A Rigorous and Stronger Meta Theorem

$\notin$ exam

Theorem 5.3 (Roura's Discrete Master Theorem)

Let $T(n)$ be recursively defined as

$$T(n) = \begin{cases} b_n & 0 \le n < n_0, \\ f(n) + \displaystyle\sum_{d=1}^{D} a_d \cdot T\left(\frac{n}{b_d} + r_{n,d}\right) & n \ge n_0, \end{cases}$$

where $D \in \mathbb{N}$, $a_d > 0$, $b_d > 1$, for $d = 1, \ldots, D$ are constants, functions $r_{n,d}$ satisfy $|r_{n,d}| = O(1)$ as $n \to \infty$, and function $f(n)$ satisfies $f(n) \sim B \cdot n^{\alpha}(\ln n)^{\gamma}$ for constants $B > 0$, α, γ.
Set $H = 1 - \sum_{d=1}^{D} a_d\,(1/b_d)^{\alpha}$; then we have:

(a) If $H < 0$, then $T(n) = O(n^{\tilde{\alpha}})$, for $\tilde{\alpha}$ the unique value of α that would make $H = 0$.

(b) If $H = 0$ and $\gamma > -1$, then $T(n) \sim f(n)\ln(n)/\tilde{H}$ with constant $\tilde{H} = (\gamma + 1)\sum_{d=1}^{D} a_d\, b_d^{-\alpha} \ln(b_d)$.

(c) If $H = 0$ and $\gamma = -1$, then $T(n) \sim f(n)\ln(n)\ln(\ln(n))/\hat{H}$ with constant $\hat{H} = \sum_{d=1}^{D} a_d\, b_d^{-\alpha} \ln(b_d)$.

(d) If $H = 0$ and $\gamma < -1$, then $T(n) = O(n^{\alpha})$.

(e) If $H > 0$, then $T(n) \sim f(n)/H$.