Chapter 1. Introduction

We are given computational/decision problems:

- Compute maximum of n numbers,
- Sorting n items in nondecreasing order,
- Compute product of 2 binary integers,
- Verify whether a given integer is prime.

For example sorting can be formally def:

Input A sequence of n numbers $a_1, a_2, \ldots, a_n$

Output A permutation $a'_1, a'_2, \ldots, a'_n$ of $a_1, \ldots, a_n$ s.t. $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Similarly we can def. primality as:

Input. A binary integer $m = b_{n-1} b_{n-2} \ldots b_0$

Question. Is m prime?

We design algorithms to solve problems. An algorithm is a well.def. deterministic computational procedure that takes inputs and produces outputs

There are 2 issues concerning algorithms: correctness and efficiency.

An algor. is correct if it halts and produces a correct output for every input instance.

In the following we discuss:

- Efficiency of algor.
- Asymptotic notations
- Recurrence relations

Efficiency of Algorithms.

Execution of algor. requires resources: computing time and storage space

We focus on computing time

Questions: (1) How to measure computing time?

(2) When is an algorithm better (faster) than another one?

(3) Is a given algorithm optimal?

Note: We deal with algorithmically solvable problems only.

How to measure computing time of an algor.?

An algorithm has to be implemented on a machine model such as Random Access machine.

Computing time depends on size of input. the larger an input the more time is needed.

We associate with an input instance a <u>size</u> and measure computing time as function of input size.

Consider a problem P:

The size of an input instance p is denoted by $\|p\|$ (which is a positive integer)

- Size of sorting is number of items to be sorted
- Size of maximum problem is number of elements in the input
- Size of integer multiplication is sum of lengths of two binary integers in input
- Size of primality problem is length of bin. representation of input integer.

Let A be an algorithm.

$T_A(p)$ = computing time of A on input instance p

Worst-Case Time Complexity

The worst-case time complexity of A on inputs of size n is:

$$T_A(n) := \sup \{ T_A(p) \mid \|p\| = n \}$$

Goal. Determine $T_A(n)$ for some important algorithms.

We confine ourselves with estimating

- **Upper bound:** $T_A(n) \leq f(n)$ for all but finitely many n.
  for some function $f(n)$. For example,
  $T_A(n) \leq cn^2$, where c is some const.

- **Lower bound:** $T_A(n) \geq g(n)$
  for some function $g(n)$. For example,
  $T_A(n) \geq d \cdot n \cdot \log n$ for all $n \geq n_0$
  where d, n_0 are some constants.

Let A_1, A_2 be two algor. that solve the same problem.

A_1 is *faster* than A_2 if $T_{A_1}(n) \leq T_{A_2}(n), \forall n$

A_1 is asymptotically faster than A_2 if

$$\lim_{n \to \infty} \frac{T_{A_1}(n)}{T_{A_2}(n)} = 0$$

(Example: If $T_{A_1}(n) \leq c \log n$
$T_{A_2}(n) \geq dn$ $\forall n \geq n_0$,
then A_1 is asymptotically faster than A_2)

Average-Case Time Complexity.

Observation: Running times of an algor. on different input instances of same size may vary significantly.

Example: Insertion Sort

```
Insertion_Sort (A)
    for i = 2 to A.length
        key = A[i]
        // Now insert A[i] into
          right place in A[1..i-1].
        j = i-1
        while j > 0 ∧ key < A[j]
            A[j+1] := A[j]
            j = j-1
        A[j+1] = key
```

Fact. Insertion_Sort takes (for some c_1, c_2)
- $\leq c_1 n$ steps on $A[1] < A[2] < \cdots < A[n]$
- $\geq c_2 n^2$ steps on $A[1] > A[2] > \cdots > A[n]$

The <u>expected time complexity</u> $T_A^{AV}(n)$ of algorithm A is def. by:

$$T_A^{av}(n) = \left(\sum_{\substack{p \in P \\ \|p\| = n}} T_A(p)\right) \Big/ \text{Card}\{p \in P \mid \|p\| = n\}$$

<u>Fact</u>. $T_A^{av}(n) \leq T_A(n)$

<u>Ex</u>. As seen later, Quicksort is $O(n \log n)$ on average and $O(n^2)$ in the worst case.

Complexity of Problems

There are potentially infinitely many algor. for a given problem.

<u>Question:</u> When is an algor. A for a probl. P optimal?

- A probl. can be <u>solved within $f(n)$</u> steps if $\exists$ algor. A s.t. $T_A(n) \leq f(n)$, $\forall n$.
- A problem P <u>requires at least $g(n)$</u> steps if every algor. A for P has time complexity $T_A(n) \geq g(n)$ for all sufficiently large n. (i.e., $\forall n \geq n_0$ where n_0 is some constant.)

An algor. A is optimal for problem P if $T_A(n)$ and the lower bound $g(n)$ have the same order of magnitude.

Order of Growth.

$\mathbb{N} = \{0,1,2,\dots\}$ Let $f : \mathbb{N} \to \mathbb{N}$ be a function:

$$O(f(n)) = \{ g(n) \mid \exists c>0 : \exists n_0 \in \mathbb{N} : g(n) \leq c \cdot f(n) \quad \forall n \geq n_0 \}$$

$$\Omega(f(n)) = \{ g(n) \mid \exists c>0 : \exists n_0 : g(n) \geq c \cdot f(n) \quad \forall n \geq n_0 \}$$

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

$$= \{ g(n) \mid \exists c>0 : \exists n_0 : \tfrac{1}{c} f(n) \leq g(n) \leq c \cdot f(n) \quad \forall n \geq n_0 \}$$

Thus, we use:

O-notation in determining upper bounds

Ω-notation " " lower bounds

Θ-notation " " exact growth rate

Remarks. (1) Instead of $n^2+5n \in n^2+O(n) \subseteq O(n^2)$ we write $n^2+5n = n^2+O(n) = O(n^2)$

(2) Expressions containing O-notation denote sets of functions. They are read from left to right only.

(3) Thus, $s_1 = s_2 = \dots = s_k$ represents sequence of larger and larger sets of functions.

Further Asymptotic Notations

Fact. Let $f(n) = O(s(n))$ and $g(n) = O(r(n))$:

(1) $f(n) + g(n) = O(s(n) + r(n))$

(2) $f(n) \cdot g(n) = O(s(n) \cdot r(n))$ □

A function $f(n)$ is **monotonically increasing** if $f(m) \geq f(n)$ $\forall m \geq n$.

Prop. Let $c > 0$, $a > 1$ be constant and $f(n)$ be a monotonically increasing function. Then $f(n)^c = O(a^{f(n)})$ □

Example. (1) $f(n) = \log n$: $(\log n)^c = O(a^{\log n}) = O(n^{\log a})$

(2) $f(n) = n$: $n^c = O(a^n)$ $a > 1$ □

Let $f(n), g(n)$ be s.t. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Then we write $f(n) = o(g(n))$

"$f(n)$ is little oh of $g(n)$"

In this case it holds: $O(f(n)) \subsetneq O(g(n))$

Examples. (1) $(\log n)^c = o(n)$ $\forall c > 0$

(2) $n^c = o(a^n)$ $\forall c > 0$ $\forall a > 1$ □

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, we write

$f(n) = \omega(g(n))$

"$f(n)$ is little omega of $g(n)$"

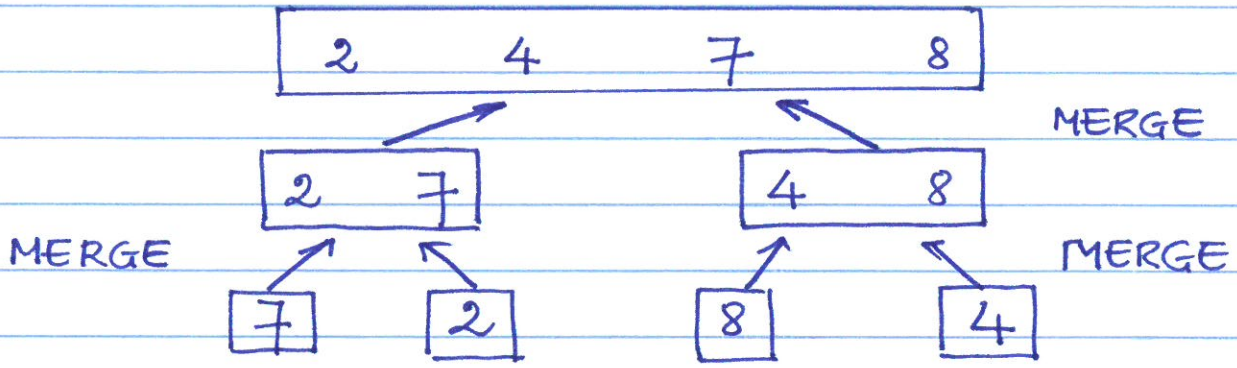
Recurrence Relations.

Example: MERGE-SORT

This is an example of divide-and-conquer

Idea:
- Split array of size n into subarrays of size $n/2$
- Sort 2 subarrays
- Merge sorted subarrays to produce sorted array.

For example,




```
MERGE-SORT (A, p, r)
if p<r then
    q = ⌊(p+r)/2⌋ ;
    MERGE-SORT (A, p, q)
    MERGE-SORT (A, q+1, r)
    MERGE (A, p, q, r)
```

where: MERGE (A, p, q, r) is procedure to merge sorted $A[p..q]$ and $A[q+1..r]$

MERGE (A, p, q, r)

$n_1 = q - p + 1$; $n_2 = r - q$

create new arrays $L[1..n_1+1]$, $R[1..n_2+1]$

$L[1..n_1] = A[p..q]$; $L[n_1+1] = \infty$;

$R[1..n_2] = A[q+1..r]$; $R[n_2+1] = \infty$;

$i = 1$; $j = 1$;

for $k = p$ to r

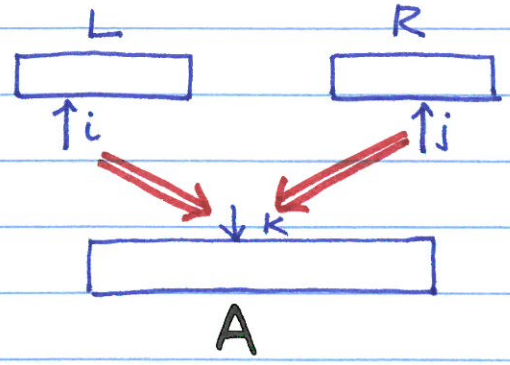
  if $L[i] \leq R[j]$

    $A[k] = L[i]$

    $i = i + 1$

  else $A[k] = R[j]$

    $j = j + 1$



Complexity of MERGE_SORT

Merging n elements takes $\Theta(n)$ steps.

Thus, running time of MERGE_SORT is:

$$T(n) = 2T(n/2) + \Theta(n)$$

This recurrence relation has solution:

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

Question: How to solve recurrences?

Some methods for solving recurrences:

- Substitution method
- Recursion tree method
- Master Theorem

Substitution Method

- Guess a solution & verify.

Example: $t(n) = 2t(\lfloor \frac{n}{2} \rfloor) + n$

Guess: $t(n) = O(n \lg n)$

Claim: $\exists c > 0 : \exists n_0 : t(n) \leq cn\lg(n) \; \forall n \geq n_0$

Pf. By induction on n.

Basis: $n = 2, 3$: ✓

Ind. Step:

IH: Assume it holds true for all m:

$$2 \leq m < n \; .$$

We show Claim holds true for n:

$$\begin{aligned} t(n) &= 2t(\lfloor \tfrac{n}{2} \rfloor) + n \\ &\leq 2\left[c\lfloor \tfrac{n}{2} \rfloor \cdot \lg(\lfloor \tfrac{n}{2} \rfloor)\right] + n \\ &\leq 2c\,\tfrac{n}{2} \cdot \lg(\tfrac{n}{2}) + n \\ &= cn\lg n - cn + n \\ &\leq cn\lg n \qquad \text{for } c > 1 \end{aligned}$$

□

Example: $t(n) = t(\lfloor \frac{n}{2} \rfloor) + t(\lceil \frac{n}{2} \rceil) + 1$

Guess: $t(n) = O(n)$

Claim: $\exists\, c > 0 : \exists\, n_0 : t(n) \leq cn \quad \forall n > n_0$

Pf.

Ind. Step:

IH: Suppose Claim holds true for

$$2 \leq m < n.$$

Noting $\lfloor \frac{n}{2} \rfloor \leq \lceil \frac{n}{2} \rceil < n$, we have:

$$t(n) = t(\lfloor \tfrac{n}{2} \rfloor) + t(\lceil \tfrac{n}{2} \rceil) + 1$$
$$\leq c(\lfloor \tfrac{n}{2} \rfloor) + c(\lceil \tfrac{n}{2} \rceil) + 1$$
$$= c \cdot n + 1 \not\leq cn \qquad \square$$

New Guess: $t(n) \leq cn - b$ for some $b > 0$

Claim: $\exists\, b, c : \exists\, n_0 : t(n) \leq cn - b \quad \forall n > n_0$

Pf.

Basis: ✓

Ind. Step:

IH: Suppose Claim holds true for all $2 \leq m < n$.

We show Claim holds true for n

$$t(n) = t(\lfloor \tfrac{n}{2} \rfloor) + t(\lceil \tfrac{n}{2} \rceil) + 1$$
$$\leq c \cdot (\lfloor \tfrac{n}{2} \rfloor) - b + c \cdot (\lceil \tfrac{n}{2} \rceil) - b + 1$$
$$= c \cdot n - 2b + 1$$
$$\leq c \cdot n - b \quad \text{for } b \geq 1.$$

Now simply choose c and b s.t. the basis ($n = 2, 3$?) works. □

Changing Variables:

Example: $t(n) = 2t(\sqrt{n}) + \lg n$

Renaming $m = \lg n$, we have:

$$t(2^m) = 2t(2^{m/2}) + m.$$

Letting $s(m) = t(2^m)$ we obtain:

$$s(m) = 2s(\tfrac{m}{2}) + m$$

Thus, $s(m) = O(m \cdot \lg m)$.

Therefore:

$$t(n) = t(2^m)$$
$$= s(m)$$
$$= O(m \lg m)$$
$$= O(\lg n \cdot \lg \lg n)$$ □

Remark: Algebraic manipulation helps!

The iteration method

Example: $t(n) = 3t(\lfloor \frac{n}{4} \rfloor) + n$

We have: $t(n)$

$$= n + 3\left(\lfloor \frac{n}{4} \rfloor + 3t\left(\lfloor \frac{\lfloor \frac{n}{4} \rfloor}{4} \rfloor\right)\right)$$

$$= n + 3\left(\lfloor \frac{n}{4} \rfloor\right) + 3^2\left(t\lfloor \frac{n}{4^2} \rfloor\right)$$

$$= n + 3\left(\lfloor \frac{n}{4} \rfloor\right) + 3^2\left(\lfloor \frac{n}{4^2} \rfloor\right) + \cdots + 3^{\log_4 n - 1}\left(\lfloor \frac{n}{4^{\log_4 n - 1}} \rfloor\right) + 3^{\log_4 n} \cdot t\left(\lfloor \frac{n}{4^{\log_4 n}} \rfloor\right)$$

(last term is when $i = \log_4 n$)

$$\leq n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \cdots + \left(\frac{3}{4}\right)^{\log_4 n - 1} n + \Theta\left(3^{\log_4 n}\right)$$

$$\leq n \cdot \underbrace{\sum_{k=0}^{\infty}\left(\frac{3}{4}\right)^k}_{= O(1)} + \Theta\left(n^{\log_4 3}\right)$$

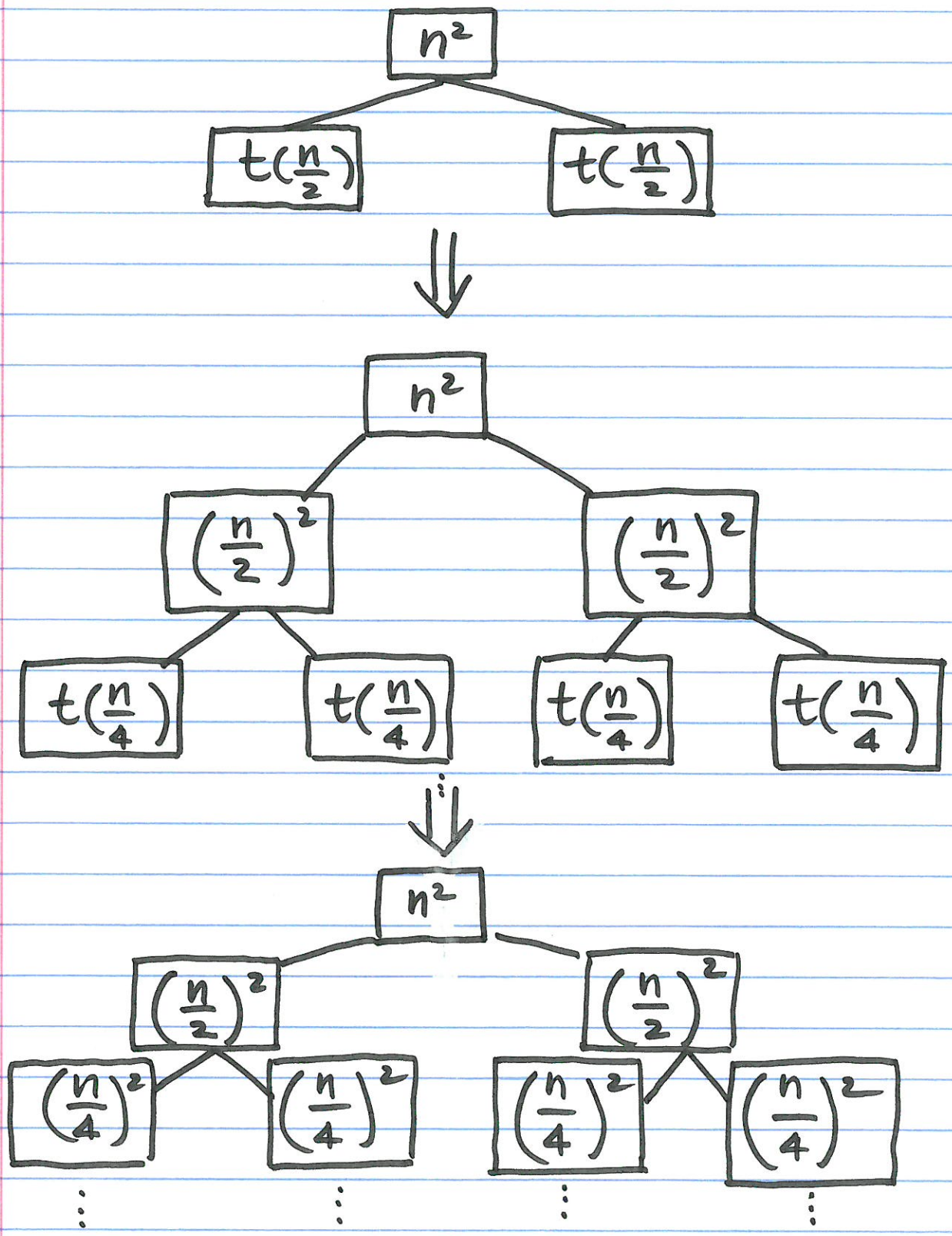
$$= O(n) + o(n)$$

$$= O(n)$$

□

Recursion Tree

Example: $t(n) = 2t\left(\frac{n}{2}\right) + n^2$



Height of tree $= \lg n$.

Thus,

$$t(n) \leq n^2\left(1 + 2\left(\frac{1}{2}\right)^2 + 4\left(\frac{1}{4}\right)^2 + \cdots\right)$$

$$= n^2\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right)$$

$$= \Theta(n^2) \qquad \square$$

The Master Theorem

Let: $a \geq 1$ and $b > 1$ be constants,

$f(n)$ be a function,

$T(n)$ be def. by

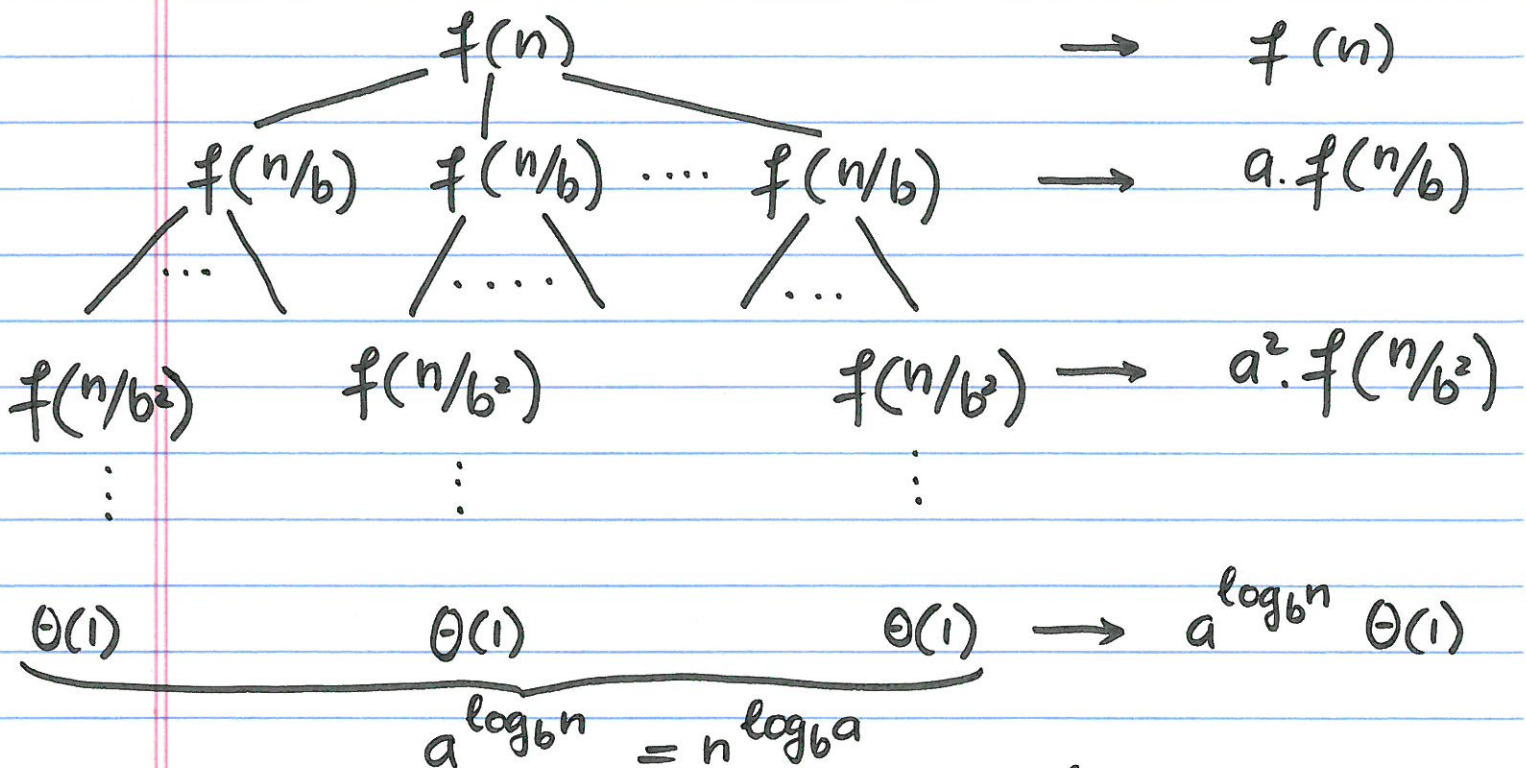
$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n).$$

(1) If $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0$,
then $T(n) = \Theta(n^{\log_b a})$

(2) If $f(n) = \Theta(n^{\log_b a})$
then $T(n) = \Theta(n^{\log_b a} \lg n)$

(3) If $f(n) = \Theta(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$,
and $a f\left(\frac{n}{b}\right) \leq c f(n)$ for some $c < 1$
and sufficiently large n,
then $T(n) = \Theta(f(n))$ $\square$

Proof Idea. Consider recursion tree



Thus:

$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i)$$

(→1.17!)

Lemma. For $g(n) = \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i)$:

(1) If $f(n) = O(n^{\log_b a - \varepsilon})$,
then $g(n) = O(n^{\log_b a})$

(2) If $f(n) = \Theta(n^{\log_b a})$,
then $g(n) = \Theta(n^{\log_b a} \cdot \lg n)$

(3) If $a f(n/b) \leq c f(n)$ for some $c < 1$
and sufficiently large n,
then $g(n) = \Theta(f(n))$ □

$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

Consider $\sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right) =: g(n)$

Question. What is $g(n)$?

(a) Suppose that $f(n) = c \cdot n^{\log_b a}$. Then

$$a f\left(\frac{n}{b}\right) = a \cdot c \cdot \left(\frac{n}{b}\right)^{\log_b a} = a \cdot c \cdot \frac{n^{\log_b a}}{b^{\log_b a}}$$

$$= ac \frac{n^{\log_b a}}{a} = c \cdot n^{\log_b a}$$

and $a^2 f\left(\frac{n}{b^2}\right) = c \cdot n^{\log_b a}$

$$\vdots$$

$$a^i f\left(\frac{n}{b^i}\right) = c \cdot n^{\log_b a}$$

Thus, $g(n) = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right) = \log_b n \cdot c \cdot n^{\log_b a}$

(b) What if $f(n) = \Theta(n^{\log_b a + \varepsilon})$?

$(= \omega(n^{\log_b a}))$

Assumption. $a f\left(\frac{n}{b}\right) \le c f(n)$ for some $c < 1$ and sufficiently large n.

Then $a^2 f\left(\frac{n}{b^2}\right) = a\left(a f\left(\frac{n/b}{b}\right)\right)$

$$\le a \cdot c f(n/b) \le c \underbrace{a f(n/b)}_{c f(n)}$$

$$\le c^2 f(n)$$

$$\Rightarrow g(n) = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right) \le \sum_{i=0}^{\log_b n-1} c^i f(n)$$

$$= f(n) \cdot O(1) \quad \text{since } \sum c^i = O(1) \text{ for } c < 1$$

(c) If $f(n) = O(n^{\log_b a - \varepsilon})$, then

$$g(n) = O\left(\sum_{i=0}^{\log_b n - 1} a^i \left(\frac{n}{b^i}\right)^{\log_b a - \varepsilon}\right)$$

Now,

$$a^i \left(\frac{n}{b^i}\right)^{\log_b a - \varepsilon}$$

$$= \frac{a^i}{(b^i)^{\log_b a}} \cdot \frac{n^{\log_b a - \varepsilon}}{(b^i)^{-\varepsilon}}$$

$$= n^{\log_b a - \varepsilon} \cdot \frac{a^i}{(b^{\log_b a})^i} \cdot (b^\varepsilon)^i$$

$$= n^{\log_b a - \varepsilon} \cdot \frac{a^i}{a^i} \quad (b^\varepsilon)^i$$

Thus,

$$g(n) = n^{\log_b a - \varepsilon} \cdot O\left(\underbrace{\sum_{i=0}^{\log_b n - 1} (b^\varepsilon)^i}\right)$$

$$= n^{\log_b a - \varepsilon} \cdot O(n^\varepsilon) \qquad \frac{(b^\varepsilon)^{\log_b n} - 1}{b^\varepsilon - 1}$$

$$= O(n^{\log_b a}) \qquad \| \qquad O(n^\varepsilon)$$

Example: As in Merge Sort let

$$T(n) = 2T(\tfrac{n}{2}) + \Theta(n)$$

Thus: $a=2$, $b=2$, $\log_b a = 1$, $f(n)=\Theta(n)$

Case (2) $\Rightarrow$ $T(n) = O(n \lg n)$ □

Example: $T(n) = 3T(\tfrac{n}{2}) + \Theta(n)$

Thus: $a=3$, $b=2$, $\log_b a = \log_2 3 > 1$

Since $f(n) = \Theta(n) = O(n^{\log_2 3 - \varepsilon})$, Case (1) $\Rightarrow$

$$T(n) = \Theta(n^{\log_2 3})$$

Example: $T(n) = 9T(\tfrac{n}{3}) + \Theta(n)$

Here: $a=9$, $b=3$, $\log_3 9 = 2$.

Since $f(n) = \Theta(n) = O(n^{2-\varepsilon})$, Case (1)

$\Rightarrow$ $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$

Example: $T(n) = T(\tfrac{2}{3}n) + \Theta(1)$

$a=1$, $b=\tfrac{3}{2}$, $\log_b a = 0$.

$f(n) = \Theta(1) = \Theta(n^{\log_b a}) \Rightarrow$ Case (2):

$$T(n) = \Theta(\lg n)$$

Example: $T(n) = 3T(\frac{n}{4}) + n\lg n$

Here: $a=3, b=4, \log_b a = \log_4 3 < 1$.

$f(n) = \Theta(n\lg n) = \Omega(n^{\log_4 3 + \varepsilon})$

$a f(n/b) = 3(\frac{n}{4})\lg(\frac{n}{4})$

$\leq (\frac{3}{4}) n\lg n$

$= c \cdot f(n)$, $c = \frac{3}{4} < 1$.

Case (3): $T(n) = \Theta(n\lg n)$

Example. $T(n) = 2T(\frac{n}{2}) + n\lg n$

We have: $a = b = 2$, $\log_b a = 1$

$f(n) = n\lg n \neq \Omega(n^{1+\varepsilon}) \quad \forall \varepsilon > 0$

$\neq O(n^{1-\varepsilon})$

$\neq \Theta(n)$

$\Rightarrow$ Master Theorem doesn't apply.

Claim. $T(n) = \Theta(n\lg^2 n)$

Pf. From rec. tree (p.17):

$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i \frac{n}{b^i} \lg_b \frac{n}{b^i}$$

$$T(n) = \Theta(n) + \sum_{i=0}^{\lg n - 1} n(\lg n - i)$$

$$= \Theta(n) + n \sum_{i=0}^{\lg n - 1} (\lg n - i)$$

$$= \Theta(n(\lg n)^2)$$

Summary:

- Notion of algor.
- Complexity of algor.
- Worst_case & average_case
- Complexity of problems
- Order of growth
- Asymptotic notations
- Recurrence relations
  - Substitution
  - Iteration / Recursion Tree
  - Master Theorem
- Insertion Sort
- Merge Sort