ALGORITHMS – COMTEK3,CCT3 & ESD3
Growth of Functions and Asymptotic Analysis

Ramoni Adeogun
Associate Professor & Head AI for Communications
Wireless Communication Networks Section (WCN)
Department of Electronic Systems
Email: ra@es.aau.dk

AALBORG UNIVERSITET

Outline

- Growth of function and asymptotic analysis
- Solving recurrences
- Summary and key takeaways

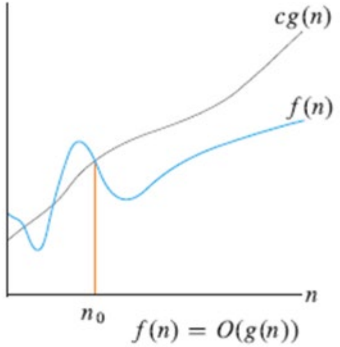
Asymptotic Notations

O-notation (upper bounds)

- O-notation: provides **asymptotic upper bound** on a function

$$O(g(n)) = \{ f(n) : \text{there exist positive constants } c, \text{and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n0 \}$$

- A function $f(n)$ **belongs to** the set $O(g(n))$ if there exists a positive constant c such that $f(n) \le f(n)$ for sufficiently large n



- Example:
  - $4n^2 + 100n + 500 = O(n^2)$; $(c = 19, n_0 = 19)$

Witnesses. Can we find other pairs?

- Is $n^3 - 100n^2 = O(n^2)$?

$\boldsymbol{\Omega}$-notation (lower bounds)

- Ω-notation: provides **asymptotic lower bound** on a function

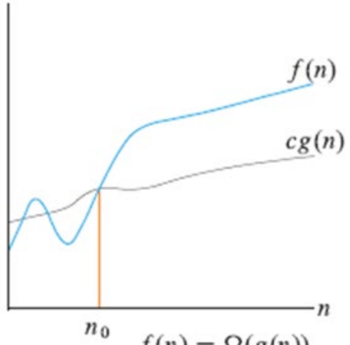
$$\boldsymbol{\Omega}(g(n)) = \{f(n) : \text{there exist positive constants } c, \text{and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n0 \}$$

- A function $f(n)$ **belongs to** the set $\boldsymbol{\Omega}(g(n))$ if there exists a positive constant c such that $cg(n) \leq cg(n)$ for sufficiently large n

- Example:
  - $4n^2 + 100n + 500 = \boldsymbol{\Omega}(n^2)$; $(c = 4, n_0 = 1)$

Witnesses. Can we find other pairs?

- Is $\sqrt{n} = \Omega(\log(n))$? What are the witnesses?



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

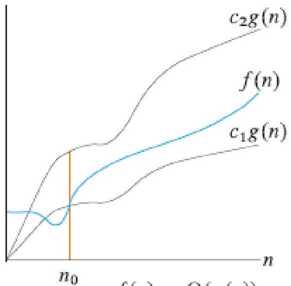
Θ-notation (tight bounds)

- Θ (big theta)-notation: provides asymptotically **tight** bounds

$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n0 \}$$

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

- **Theorem**: For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.



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

- Examples:
  - $\frac{1}{2}n^2 - 2n = \Theta(n^2)$; (… … … … … … … . .)

o-notation

- o(little-oh)-notation: provides **asymptotic upper bound** that is not tight

$$\mathrm{o}(g(n)) = \{ f(n) : \text{for any positive constant } c > 0, \text{there exists a constant } n_0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n0 \}$$

- $\boldsymbol{O}$- and o-notations are similar. The key difference is that:
  - $f(n) = O(g(n))$ holds for some constant $c > 0$ but $f(n) = o(g(n))$ holds for all constants $c > 0$

- Example:
  - $2n = o(n^2)$; $(n_0 = \qquad)$

- Is $2n^2 = o(n^2)$?

o-notation

- ω-notation: provides **asymptotic lower bound** that is not tight
- **Analogy**: ω-notation is to Ω-notation as o-notation is to O-notation

$$\omega(g(n)) \;=\; \{\, f(n) : \text{for any positive constant } c > 0, \text{there exists a constant } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq \; n0 \,\}$$

- **Definition*:** $f(n) \in \omega(g(n))$ if and only if $g(n) \in o(f(n))$

- Example:
  - $\frac{n^2}{2} = \omega(n);$

- Is $\frac{\mathrm{n}^2}{2} = \omega(n^2)$?

Comparison of Functions

- Relational properties:
  - Transitivity:
    - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \rightarrow f(n) = \Theta(h(n))$
    - Same of $O, \Omega, o,$ and ω
  - Reflexivity:
    - $f(n) = \Theta(f(n))$
    - Same for O and Ω
  - Symmetry:
    - $f(n) = \Theta(g(n))$ if and only if g$(n) = \Theta(f(n))$
  - Transpose symmetry:
    - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$
    - $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$
- Comparisons:
  - $f(n)$ is asymptotically smaller than $g(n)$ if $f(n) = o(g(n))$
  - $f(n)$ is asymptotically larger than $g(n)$ if $f(n) = \omega(g(n))$

Solving Recurrences

Recurrence

- We use a **recurrence** to characterize the running time of a divide-and-conquer algorithm. Solving the recurrence gives the asymptotic running time.

- A recurrence is a function defined in terms of
  - one or more base cases, and
  - itself, with smaller arguments.

- **Examples:**
  - $T(n) = \begin{cases} 1, & if\ n = 1 \\ T(n-1) + 1, & if\ n > 1 \end{cases}$ Solution: $T(n) = n$
  - $T(n) = \begin{cases} 1, & if\ n = 1 \\ 2T(n/2) + n, & if\ n \geq 1 \end{cases}$ Solution: $T(n) = n \log_2(n)$

Solving Recurrences

- Methods for solving recurrences
  - **Substitution method**
    - Guess the form of a bound
      - Verify by mathematical induction
        - Solve for constants
  - **Recursion-tree method:**
    - Model recurrence as a tree with nodes representing costs
      - Determine the cost at each level
        - Add them up
  - **Master method:**
    - Provides bounds for recurrence of the form: $T(n) = aT\left(\frac{n}{b}\right) + f(n); a > 0 \text{ and } b > 1$
  - Akra-Bazzi method*:
    - A general method for solving recurrences involving use of calculus

Substitution Method

The most general method for solving recurrences:

1. **Guess** the form of a bound
2. **Verify** by mathematical **induction**
3. **Solve** for constants

Example: Solve the recurrence: $$T(n) = \begin{cases} \Theta(1), & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & if\ n \geq 1 \end{cases}$$

Proof:

1. Guess: $T(n) = n\log_2(n) + n$

Substitution Method

2. Induction:

Base case: $n = 1 \rightarrow n\log_2(n) + n = 1 = T(n)$

Inductive step: Inductive hypothesis: $T(k) = k\log_2(k) + k \quad \forall k < n$

- **Substitution**: Substitute the inductive hypothesis into the recurrence:

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n \\
&= 2\left(\frac{n}{2}\log_2\left(\frac{n}{2}\right) + \frac{n}{2}\right) + n \\
&= n\log_2\left(\frac{n}{2}\right) + n + n \\
&= n\log_2(n) - n\log_2(2) + n + n \\
&= n\log_2(n) - n + n + n \\
&= n\log_2(n) + n
\end{aligned}
$$

Substitution Method

- Generally, we use asymptotic notation:
  - We would write $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$
  - We assume $T(n) = O(1)$ for sufficiently small n.
  - We express the solution by the asymptotic notation: $T(n) = \Theta(n \log_2(n))$.
  - When we want an asymptotic solution to a recurrence, we don't worry about the base cases in our proofs.
  - When we want an exact solution, then we have to deal with base cases.

- Note:
  - Name the constant in the additive term.
  - Show the upper (O) and lower (Ω) bounds separately.
    - Might need to use different constants for each.

Substitution Method - Example

Consider the recurrence: $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$.

Upper bound:

- We write $T(n) \leq 2T\left(\frac{n}{2}\right) + cn$
- Guess: $T(n) \leq dn\log_2(n)$ for some positive constant d.
- Substitution:

$$
\begin{aligned}
T(n) &\leq 2T\left(\frac{n}{2}\right) + cn \\
&= 2\left(d\frac{n}{2}\log_2\left(\frac{n}{2}\right) + \frac{n}{2}\right) + cn \\
&= dn\log_2\left(\frac{n}{2}\right) + cn \\
&= dn\log_2(n) - dn\log_2(2) + cn \\
&= dn\log_2(n) - dn + cn \\
&\leq dn\log_2(n) \qquad if\ \ -dn + cn \leq 0, \quad d \geq c
\end{aligned}
$$

Therefore, $T(n) = O(n\log_2(n))$

Substitution Method - Exampe

Lower bound:

- We write $T(n) \geq 2T\left(\frac{n}{2}\right) + cn$
- Guess: $T(n) \geq dn\log_2(n)$ for some positive constant d.
- Substitution:

$$T(n) \geq 2T\left(\frac{n}{2}\right) + cn$$
$$= 2(d\frac{n}{2}\log_2\left(\frac{n}{2}\right) + \frac{n}{2}) + cn$$
$$= dn\log_2\left(\frac{n}{2}\right) + cn$$
$$= dn\log_2(n) - dn\log_2(2) + cn$$
$$= dn\log_2(n) - dn + cn$$
$$\leq dn\log_2(n) \qquad if\ \ -dn + cn \geq 0, \quad d \leq c$$

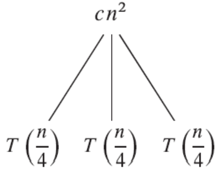
Therefore, $T(n) = \Omega(n\log_2(n))$ and $\boldsymbol{T(n) = \Theta(n\,log_2(n))}$

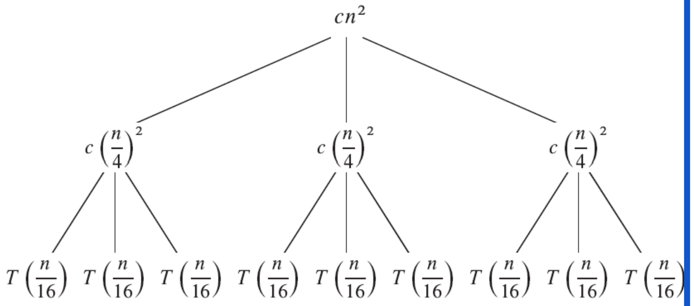
Recursion Tree Method

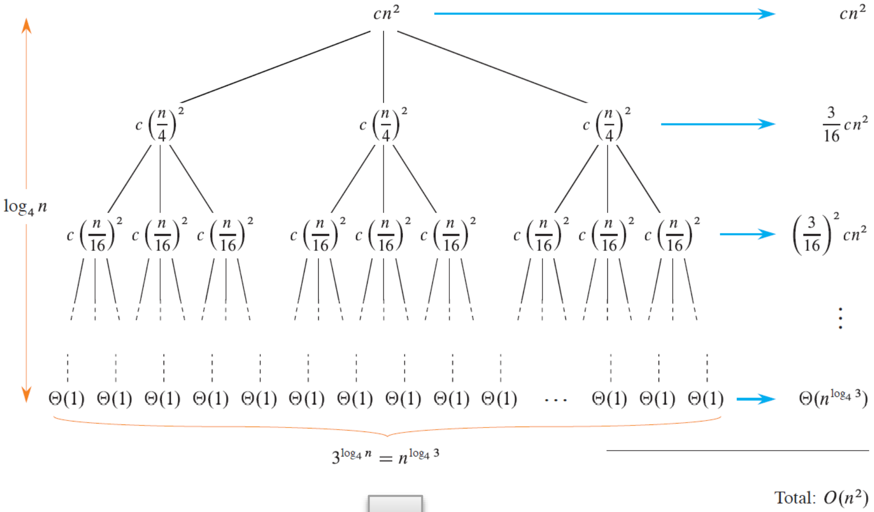
- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be **unreliable.**
- The recursion-tree method **promotes intuition**, however.
- The recursion tree method is good for generating guesses for the substitution method.
  - If you are meticulous when drawing out a recursion tree and summing the costs, however, you can use a recursion tree as a direct proof of a solution to a recurrence

Recursion Tree Method - Example

- Recurrence: $T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n^2)$
- We draw recursion tree for $T(n) = 3T\left(\frac{n}{4}\right) + cn^2$







Recursion Tree Method - Example

- We add up the costs over all levels to determine the cost for the entire tree:

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4(n)} cn^2 + \Theta\left(n^{\log_4(3)}\right) \\
&= \sum_{i=0}^{n^{\log_4(n)}} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4(3)}\right) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4(3)}\right) \\
&= \frac{1}{1-\left(\frac{3}{16}\right)} cn^2 + \Theta\left(n^{\log_4(3)}\right) \\
&= \frac{16}{13} cn^2 + \Theta\left(n^{\log_4(3)}\right) \\
&= O(n^2)
\end{aligned}
$$

Exercise

- Use substitution method to verify that $T(n) = O(n^2)$ is an upper bound for the recurrence $T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n)$

The Master Method

- The master method applies to recurrences of the form
$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$
where $a \geq 1$, $b > 1$, and f is asymptotically positive.

- The master method depends upon the **master theorem.**

The Master Theorem

Let

- $a > 0$ and $b > 1$ be constants, and
- let $f(n)$ be a driving function that is defined and nonnegative on all sufficiently large reals.

Define the recurrence $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ on $n \in \mathrm{N}$. The asymptotic behaviour of T(n) can be characterized as follows (compare $f(n)$ with the **watershed** function):

1. $f(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0 \;\rightarrow\; T(n) = \Theta\left(n^{\log_b(a)}\right)$

2. $f(n) = O\left(n^{\log_b(a)} \log^k(n)\right)$ for some constant $k \geq 0 \rightarrow T(n) = \Theta\left(n^{\log_b(a)} \log^{k+1}(n)\right)$

3. $f(n) = O\left(n^{\log_b(a)+\epsilon}\right)$ for some constant $\epsilon > 0$ **and** $f(n)$ satifies the ***regularity condition*** $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1 \;\rightarrow\; T(n) = \Theta(f(n))$

Using the Master Method

- To use the master method:
  - Determine which case (if any) of the master theorem applies
  - Write down the answer

- Example:
  - $T(n) = 4T\left(\frac{n}{2}\right) + n$
    $a = 4, b = 2 \rightarrow n^2 = n^{\log_b(a)}; f(n) = n$
    Case 1: $f(n) = O(n^{2-\epsilon})$ with $\epsilon = 1$
    Therefore: $T(n) = \Theta(n^2)$.

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

Summary and Key Take-aways

- **Asymptotic analysis** allows us to describe behavior of functions in the limit
  - Describe growth of functions
  - Abstract lower order terms and constant factors
- Asymptotic notations
  - $O \approx \leq$
  - $\Omega \approx \geq$
  - $\Theta \approx =$
  - $o \approx <$
  - $\omega \approx >$
- Solving recurrences:
  - **Substitution method**
  - **Recursion tree method**
  - **Master method**
  - Aka-Bazi method

- **Next lecture**: Divide and conquer algorithms