# Recursion and Structural Induction

COMP2711 Discrete Mathematical Tools

Mingxun Zhou (mingxunz@ust.hk)

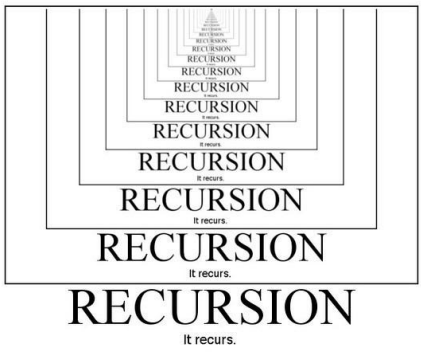



Image source

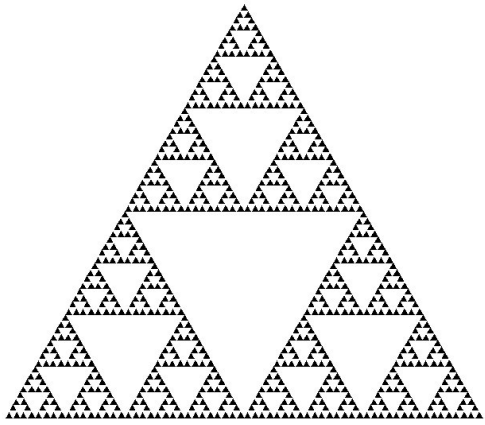


Image source: University of Toronto

# Learning objectives

1. **Recursive definitions**
   Define an object using base cases and construction rules.

2. **Structural induction**
   Use those rules to prove a property of every object.

3. **Solving recurrences**
   Expand a recursive equation and verify the result by induction.

# Contents

1. **Recursive definitions**

2. Structural induction

3. Solving recurrences

# Recursive definitions

MUST UNDERSTAND

## Recursive definition

**Base cases:** specify the starting values or objects.

**Recursive rules:** build new values or objects from smaller ones.

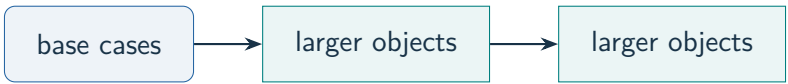
# Recursive definitions

**MUST UNDERSTAND**

## Recursive definition

**Base cases:** specify the starting values or objects.

**Recursive rules:** build new values or objects from smaller ones.



# Case study 1: A recursive sequence

**Recursive definition**

$a_0 = 3, \qquad a_n = 2a_{n-1} + 3 \quad (n \geq 1).$

$$3,\ 9,\ 21,\ 45,\ 93, \ldots$$

# Case study 1: A recursive sequence

**Recursive definition**

$a_0 = 3, \qquad a_n = 2a_{n-1} + 3 \quad (n \geq 1).$

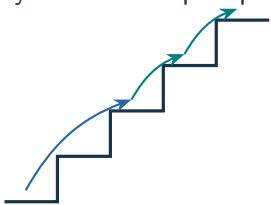
$$3,\ 9,\ 21,\ 45,\ 93, \ldots$$

The same sequence has the explicit formula $a_n = 6 \times 2^n - 3$.

**Sometimes it is not easy to see the explicit formula.**

# Case study 2: Counting stair climbs

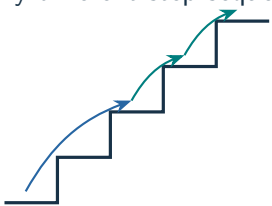
You may climb **1 or 2 stairs** per step.
How many different step sequences climb exactly $n$ stairs?



example: $2, 1, 1$

# Case study 2: Counting stair climbs

You may climb **1 or 2 stairs** per step.
How many different step sequences climb exactly $n$ stairs?



example: $2, 1, 1$

For $n = 4$:

$$1111, \quad 112, \quad 121, \quad 211, \quad 22.$$

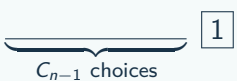
There are five ways.

# Case study 2: Split by the last step

Let $C_n$ count climbs of exactly $n$ stairs.

## Last step has size 1

A climb of $n-1$ stairs, then 1.

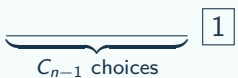


# Case study 2: Split by the last step

Let $C_n$ count climbs of exactly $n$ stairs.

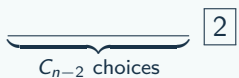
## Last step has size 1

A climb of $n-1$ stairs, then 1.




## Last step has size 2

A climb of $n-2$ stairs, then 2.

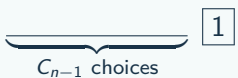


# Case study 2: Split by the last step

Let $C_n$ count climbs of exactly $n$ stairs.

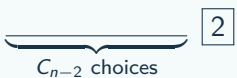
## Last step has size 1

A climb of $n-1$ stairs, then 1.




## Last step has size 2

A climb of $n-2$ stairs, then 2.




$$C_n = C_{n-1} + C_{n-2} \quad (n \geq 2), \qquad C_0 = 1, \quad C_1 = 1.$$

$C_0 = 1$ counts the empty sequence.

# Contents

1. Recursive definitions

2. **Structural induction**

3. Solving recurrences

# What makes an expression valid?

**Valid**

$$(x + y) \times z$$

**Invalid**

$$(x+) \times z$$

# What makes an expression valid?

**Valid**

$$(x + y) \times z$$

**Invalid**

$$(x+) \times z$$

**How can we use mathematical rules to describe all valid expressions?**

# Recursive sets: Arithmetic expressions

## Valid arithmetic expressions

**Base:** every variable $x, y, z, \ldots$ is a valid expression.

# Recursive sets: Arithmetic expressions

## Valid arithmetic expressions

**Base:** every variable $x, y, z, \ldots$ is a valid expression.

**Parentheses:** if $E$ is valid, then $(E)$ is valid.

# Recursive sets: Arithmetic expressions

## Valid arithmetic expressions

**Base:** every variable $x, y, z, \ldots$ is a valid expression.

**Parentheses:** if $E$ is valid, then $(E)$ is valid.

**Operations:** if $E$ and $F$ are valid, then

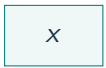
$$E + F, \qquad E - F, \qquad E \times F, \qquad E \div F$$

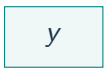
are valid.

Only expressions built by finitely many applications of these rules are included.

# Case study 3: Build an arithmetic expression

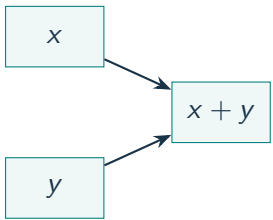
Construct $(x + y) \times z$ using the rules.



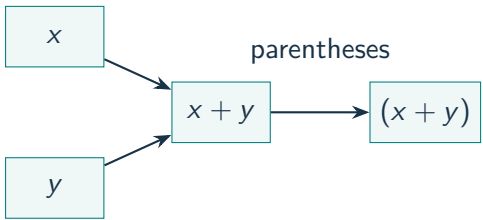
# Case study 3: Build an arithmetic expression

Construct $(x + y) \times z$ using the rules.



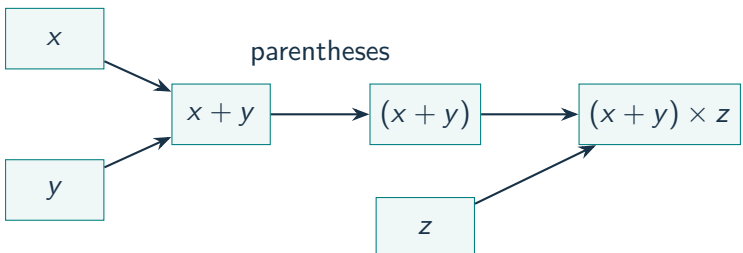
# Case study 3: Build an arithmetic expression

Construct $(x + y) \times z$ using the rules.



# Case study 3: Build an arithmetic expression

Construct $(x + y) \times z$ using the rules.



# Structural induction

**MUST UNDERSTAND**

To prove a property of every generated object

1. Prove it for every **base object**.
2. For each **construction rule**, assume it for the input objects and prove it for the resulting object.

# Structural induction

**MUST UNDERSTAND**

To prove a property of every generated object

1. Prove it for every **base object**.
2. For each **construction rule**, assume it for the input objects and prove it for the resulting object.

| **Construction** | **Proof case** |
|---|---|
| A variable $x$ | Prove $P(x)$. |
| Wrap $E$ in parentheses | $P(E) \Rightarrow P((E))$. |
| Combine $E$ and $F$ | $P(E), P(F) \Rightarrow P(E \circ F)$. |

$\circ$ is any of $+$, $-$, $\times$, $\div$.

# Case study 3: Count the parentheses

**To prove**

Every valid expression has equally many opening and closing parentheses.

Let $L(E)$ and $R(E)$ count opening and closing parentheses in $E$.

# Case study 3: Count the parentheses

**To prove**

Every valid expression has equally many opening and closing parentheses.

Let $L(E)$ and $R(E)$ count opening and closing parentheses in $E$.

**Base:** a variable has $L(x) = R(x) = 0$.

# Case study 3: Count the parentheses

**To prove**

Every valid expression has equally many opening and closing parentheses.

Let $L(E)$ and $R(E)$ count opening and closing parentheses in $E$.

**Base:** a variable has $L(x) = R(x) = 0$.

**Parentheses rule**

Assume $L(E) = R(E)$. Wrapping $E$ adds one of each:

$$L((E)) = L(E) + 1 = R(E) + 1 = R((E)).$$

# Case study 3: Combine two expressions

**Inductive hypothesis**

$L(E) = R(E), \qquad L(F) = R(F).$

For any operation $\circ \in \{+, -, \times, \div\}$, the counts add:

$$L(E \circ F)$$

# Case study 3: Combine two expressions

**Inductive hypothesis**

$L(E) = R(E), \qquad L(F) = R(F).$

For any operation $\circ \in \{+, -, \times, \div\}$, the counts add:

$$L(E \circ F) = L(E) + L(F)$$

# Case study 3: Combine two expressions

**Inductive hypothesis**

$L(E) = R(E), \qquad L(F) = R(F).$

For any operation $\circ \in \{+, -, \times, \div\}$, the counts add:

$$\begin{aligned} L(E \circ F) &= L(E) + L(F) \\ &= \underbrace{R(E) + R(F)}_{\text{inductive hypotheses}} \end{aligned}$$

# Case study 3: Combine two expressions

**Inductive hypothesis**

$L(E) = R(E), \qquad L(F) = R(F).$

For any operation $\circ \in \{+, -, \times, \div\}$, the counts add:

$$\begin{aligned} L(E \circ F) &= L(E) + L(F) \\ &= \underbrace{R(E) + R(F)}_{\text{inductive hypotheses}} \\ &= R(E \circ F). \end{aligned}$$

# Case study 3: Check validity recursively

**Algorithm 1** $\mathrm{Valid}(s)$

**Input:** a sequence of symbols $s$ (spaces ignored).
**Output:** **true** if $s$ is valid; **false** otherwise.

1: **if** $s$ is a single variable **then**
2: **return true**

# Case study 3: Check validity recursively

**Algorithm 1** $\mathrm{Valid}(s)$

**Input:** a sequence of symbols $s$ (spaces ignored).
**Output:** **true** if $s$ is valid; **false** otherwise.

1: **if** $s$ is a single variable **then**
2: **return true**
3: **if** $s = (E)$ with one matching outer pair **then**
4: **return** $\mathrm{Valid}(E)$

# Case study 3: Check validity recursively

**Algorithm 1** $\mathrm{Valid}(s)$

**Input:** a sequence of symbols $s$ (spaces ignored).
**Output:** **true** if $s$ is valid; **false** otherwise.

1: **if** $s$ is a single variable **then**
2: **return true**
3: **if** $s = (E)$ with one matching outer pair **then**
4: **return** $\mathrm{Valid}(E)$
5: **for** each split $s = E \circ F$ with $\circ \in \{+, -, \times, \div\}$ **do**
6: **if** $\mathrm{Valid}(E) \wedge \mathrm{Valid}(F)$ **then**
7: **return true**

# Case study 3: Check validity recursively

**Algorithm 1** $\mathrm{Valid}(s)$

**Input:** a sequence of symbols $s$ (spaces ignored).
**Output:** **true** if $s$ is valid; **false** otherwise.

1: **if** $s$ is a single variable **then**
2: **return true**
3: **if** $s = (E)$ with one matching outer pair **then**
4: **return** $\mathrm{Valid}(E)$
5: **for** each split $s = E \circ F$ with $\circ \in \{+, -, \times, \div\}$ **do**
6: **if** $\mathrm{Valid}(E) \wedge \mathrm{Valid}(F)$ **then**
7: **return true**
8: **return false**

# Case study 4: Full binary trees

A **leaf** has no children. In a **full binary tree**, every other vertex has exactly two children.

## Recursive definition

**Base**



one leaf

# Case study 4: Full binary trees

A **leaf** has no children. In a **full binary tree**, every other vertex has exactly two children.

## Recursive definition

**Base**



one leaf

**Rule: join two full binary trees under a new root.**



# Case study 4: Full binary trees

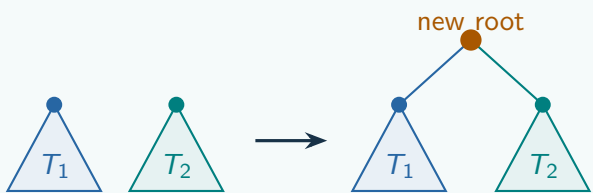
A **leaf** has no children. In a **full binary tree**, every other vertex has exactly two children.

## Recursive definition

**Base**




**Rule: join two full binary trees under a new root.**



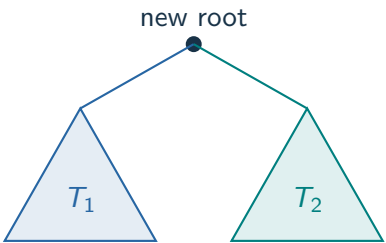

Include only trees built by finitely many applications of these rules.

# Case study 4: Size and height

$v(T)$: number of vertices. $h(T)$: longest root-to-leaf path, in **edges**.

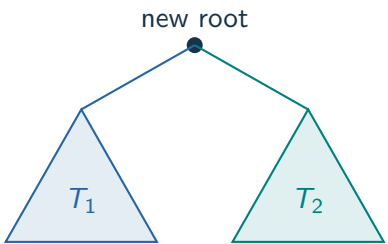
**A single leaf:**

$$v(T) = 1, \qquad h(T) = 0.$$



# Case study 4: Size and height

$v(T)$: number of vertices. $h(T)$: longest root-to-leaf path, in **edges**.




**A single leaf:**

$$v(T) = 1, \qquad h(T) = 0.$$

**A new root with subtrees:**

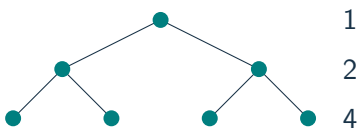
$$v(T) = 1 + v(T_1) + v(T_2),$$

$$h(T) = 1 + \max\{h(T_1), h(T_2)\}.$$

# Case study 4: A bound on tree size

**To prove**

Every full binary tree $T$ satisfies $v(T) \leq 2^{h(T)+1} - 1$.

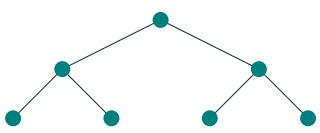



$h = 2, \quad v = 7 = 2^3 - 1$

# Case study 4: A bound on tree size

**To prove**

Every full binary tree $T$ satisfies $v(T) \leq 2^{h(T)+1} - 1$.




$h = 2, \quad v = 7 = 2^3 - 1$

**Base:** a single leaf.

$$v = 1, \qquad h = 0,$$

$$1 = 2^{0+1} - 1.$$

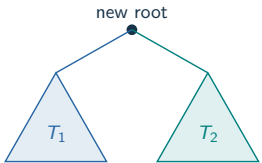
# Case study 4: Apply both subtree hypotheses

## Inductive hypothesis

$v(T_1) \leq 2^{h(T_1)+1} - 1, \qquad v(T_2) \leq 2^{h(T_2)+1} - 1.$

Put $h = h(T)$. Then $h(T_1), h(T_2) \leq h - 1$.

$$v(T) = 1 + v(T_1) + v(T_2)$$

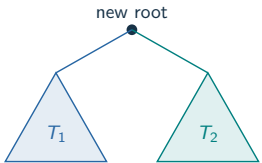


# Case study 4: Apply both subtree hypotheses

**Inductive hypothesis**

$v(T_1) \leq 2^{h(T_1)+1} - 1, \qquad v(T_2) \leq 2^{h(T_2)+1} - 1.$

Put $h = h(T)$. Then $h(T_1), h(T_2) \leq h - 1$.



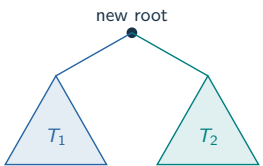

$$\begin{aligned} v(T) &= 1 + v(T_1) + v(T_2) \\ &\leq 1 + (2^{h(T_1)+1} - 1) \\ &\quad + (2^{h(T_2)+1} - 1) \end{aligned}$$

# Case study 4: Apply both subtree hypotheses

**Inductive hypothesis**

$v(T_1) \leq 2^{h(T_1)+1} - 1, \qquad v(T_2) \leq 2^{h(T_2)+1} - 1.$

Put $h = h(T)$. Then $h(T_1), h(T_2) \leq h - 1$.



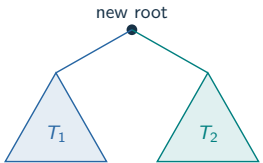

$$\begin{aligned} v(T) &= 1 + v(T_1) + v(T_2) \\ &\leq 1 + (2^{h(T_1)+1} - 1) \\ &\quad + (2^{h(T_2)+1} - 1) \\ &\leq 1 + (2^h - 1) + (2^h - 1) \end{aligned}$$

# Case study 4: Apply both subtree hypotheses

## Inductive hypothesis

$v(T_1) \le 2^{h(T_1)+1} - 1, \qquad v(T_2) \le 2^{h(T_2)+1} - 1.$

Put $h = h(T)$. Then $h(T_1), h(T_2) \le h - 1$.



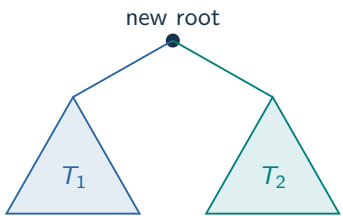

$$\begin{aligned} v(T) &= 1 + v(T_1) + v(T_2) \\ &\le 1 + (2^{h(T_1)+1} - 1) \\ &\quad + (2^{h(T_2)+1} - 1) \\ &\le 1 + (2^h - 1) + (2^h - 1) \\ &= 2^{h+1} - 1. \end{aligned}$$

# Case study 4: Exercise on leaves

Let $L(T)$ count leaves and $I(T)$ count internal vertices.

**To prove**

For every full binary tree $T$, $L(T) = I(T) + 1$.




**Start with:**

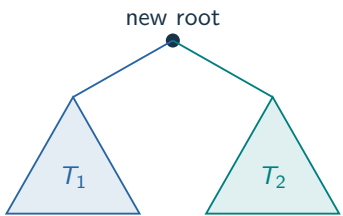
$$L(\text{leaf}) = 1, \qquad I(\text{leaf}) = 0.$$

# Case study 4: Exercise on leaves

Let $L(T)$ count leaves and $I(T)$ count internal vertices.

**To prove**

For every full binary tree $T$, $L(T) = I(T) + 1$.




**Start with:**

$$L(\text{leaf}) = 1, \qquad I(\text{leaf}) = 0.$$

$$L(T) = L(T_1) + L(T_2),$$
$$I(T) = 1 + I(T_1) + I(T_2).$$

# Contents

1. Recursive definitions

2. Structural induction

3. **Solving recurrences**

# Recurrences and closed forms

**MUST UNDERSTAND**

**Recurrence relation**

An equation that expresses a term using earlier terms.

# Recurrences and closed forms

MUST UNDERSTAND

**Recurrence relation**

An equation that expresses a term using earlier terms.

**Recursive definition**

$s_0 = 0, \qquad s_n = s_{n-1} + 2n - 1 \quad (n \geq 1).$

**Closed form**

$$s_n = n^2.$$

An expression directly in terms of $n$.

# Recurrences and closed forms

MUST UNDERSTAND

**Recurrence relation**

An equation that expresses a term using earlier terms.

**Recursive definition**

$s_0 = 0, \qquad s_n = s_{n-1} + 2n - 1 \quad (n \geq 1).$

**Closed form**

$s_n = n^2.$

An expression directly in terms of $n$.

**Method:** expand to the base, sum the terms, verify by induction.

# Case study 5: Solve a recurrence

**Find a closed form**

$a_0 = 3, \qquad a_n = 2a_{n-1} + 3 \quad (n \geq 1).$

# Case study 5: Solve a recurrence

**Find a closed form**

$a_0 = 3, \qquad a_n = 2a_{n-1} + 3 \quad (n \geq 1).$

$$\begin{aligned} a_n &= 2a_{n-1} + 3 \\ &= 2^2 a_{n-2} + 3(2+1) \\ &= \cdots = 2^n a_0 + 3(2^{n-1} + \cdots + 2 + 1). \end{aligned}$$

# Case study 5: Solve a recurrence

**Find a closed form**

$a_0 = 3, \qquad a_n = 2a_{n-1} + 3 \quad (n \geq 1).$

$$\begin{aligned} a_n &= 2a_{n-1} + 3 \\ &= 2^2 a_{n-2} + 3(2+1) \\ &= \cdots = 2^n a_0 + 3(2^{n-1} + \cdots + 2 + 1). \end{aligned}$$

Using the geometric sum,

$$a_n = 3 \times 2^n + 3(2^n - 1) = 6 \times 2^n - 3.$$

# Case study 5: Verify by induction

**To prove**

The recurrence $a_0 = 3$, $a_n = 2a_{n-1} + 3$ satisfies

$$a_n = 6 \times 2^n - 3 \qquad (n \geq 0).$$

**Base:** $6 \times 2^0 - 3 = 3 = a_0$.

# Case study 5: Verify by induction

**To prove**

The recurrence $a_0 = 3$, $a_n = 2a_{n-1} + 3$ satisfies

$$a_n = 6 \times 2^n - 3 \qquad (n \geq 0).$$

**Base:** $6 \times 2^0 - 3 = 3 = a_0$.

**Step:** assume $a_k = 6 \times 2^k - 3$.

$$a_{k+1} = 2a_k + 3 = 2(6 \times 2^k - 3) + 3 = 6 \times 2^{k+1} - 3.$$

# Case study 6: Recursive binary search

Find `target` in a sorted array $A$; return its index, or $-1$ if absent.

BinarySearch($A$, `target`, `low`, `high`)

```
1: if low > high return −1                          ▷ empty range
2: mid ← ⌊(low + high)/2⌋
3: if A[mid] = target return mid
4: if target < A[mid] then
5:     return BinarySearch(A, target, low, mid − 1)
6: return BinarySearch(A, target, mid + 1, high)
```

Initial call: BinarySearch($A$, `target`, $1$, $|A|$).

# Case study 6: Binary search intuition

Only **one half** is searched. Each call does constant extra work.

$$n \xrightarrow{+1} n/2 \xrightarrow{+1} n/4 \longrightarrow \cdots \longrightarrow 1$$

# Case study 6: Binary search intuition

Only **one half** is searched. Each call does constant extra work.

$$n \xrightarrow{+1} n/2 \xrightarrow{+1} n/4 \longrightarrow \cdots \longrightarrow 1$$

## Worst-case running time

$$\underbrace{\Theta(1)}_{\text{work per level}} \times \underbrace{\Theta(\log n)}_{\text{number of levels}} = \Theta(\log n).$$

# Case study 6: Binary search statement and proof

## Claim for the simplified cost model

For $n = 2^k$, $k \geq 0$, let $T_{\text{binary}}(1) = 1$ and $T_{\text{binary}}(n) = T_{\text{binary}}(n/2) + 1$ for $n > 1$. Then

$$T_{\text{binary}}(n) = 1 + \log_2 n = \Theta(\log n).$$

# Case study 6: Binary search statement and proof

## Claim for the simplified cost model

For $n = 2^k$, $k \geq 0$, let $T_{\text{binary}}(1) = 1$ and $T_{\text{binary}}(n) = T_{\text{binary}}(n/2) + 1$ for $n > 1$. Then

$$T_{\text{binary}}(n) = 1 + \log_2 n = \Theta(\log n).$$

**Proof:** after $j$ expansions, $0 \leq j \leq k$,

$$T_{\text{binary}}(n) = T_{\text{binary}}(n/2^j) + j.$$

# Case study 6: Binary search statement and proof

## Claim for the simplified cost model

For $n = 2^k$, $k \geq 0$, let $T_{\text{binary}}(1) = 1$ and $T_{\text{binary}}(n) = T_{\text{binary}}(n/2) + 1$ for $n > 1$. Then

$$T_{\text{binary}}(n) = 1 + \log_2 n = \Theta(\log n).$$

**Proof:** after $j$ expansions, $0 \leq j \leq k$,

$$T_{\text{binary}}(n) = T_{\text{binary}}(n/2^j) + j.$$

At $j = k = \log_2 n$, the remaining input is 1:

$$T_{\text{binary}}(n) = \underbrace{T_{\text{binary}}(1)}_{1} + \log_2 n = 1 + \log_2 n.$$

# Case study 7: Recursive merge sort

Return a sorted copy of $A$.

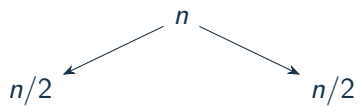
**MergeSort($A$)**

1: $n \leftarrow |A|$
2: **if** $n \leq 1$ **return** $A$ ▷ already sorted
3: $\texttt{mid} \leftarrow \lfloor n/2 \rfloor$
4: $\texttt{sortedLeft} \leftarrow \text{MergeSort}(A[1 \ldots \texttt{mid}])$
5: $\texttt{sortedRight} \leftarrow \text{MergeSort}(A[\texttt{mid} + 1 \ldots n])$
6: **return** $\text{Merge}(\texttt{sortedLeft}, \texttt{sortedRight})$ ▷ combine

Merge combines two sorted lists in linear time.

# Case study 7: Merge sort intuition

**Both halves** are sorted. Splitting and merging take linear work.

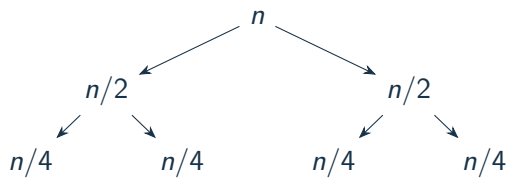



work: $n$

work: $2(n/2) = n$

# Case study 7: Merge sort intuition

**Both halves** are sorted. Splitting and merging take linear work.



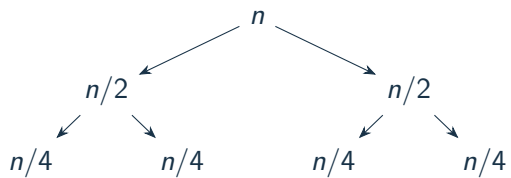

work: $n$

work: $2(n/2) = n$

work: $4(n/4) = n$

# Case study 7: Merge sort intuition

**Both halves** are sorted. Splitting and merging take linear work.




work: $n$

work: $2(n/2) = n$

work: $4(n/4) = n$

$$\underbrace{\Theta(n)}_{\text{work per level}} \times \underbrace{\Theta(\log n)}_{\text{number of levels}} = \Theta(n \log n).$$

# Case study 7: Merge sort statement and proof

## Running time of merge sort

For powers of two, $T_{\text{merge}}(1) = \Theta(1)$ and

$$T_{\text{merge}}(n) = 2\,T_{\text{merge}}(n/2) + \Theta(n).$$

Then $T_{\text{merge}}(n) = \Theta(n \log n)$.

# Case study 7: Merge sort statement and proof

## Running time of merge sort

For powers of two, $T_{\text{merge}}(1) = \Theta(1)$ and

$$T_{\text{merge}}(n) = 2\,T_{\text{merge}}(n/2) + \Theta(n).$$

Then $T_{\text{merge}}(n) = \Theta(n \log n)$.

**Proof:** level $j$ has $2^j$ subproblems, each of size $n/2^j$.

Total work: $\quad 2^j \Theta(n/2^j) = \Theta(n)$.

# Case study 7: Merge sort statement and proof

## Running time of merge sort

For powers of two, $T_{\text{merge}}(1) = \Theta(1)$ and

$$T_{\text{merge}}(n) = 2\,T_{\text{merge}}(n/2) + \Theta(n).$$

Then $T_{\text{merge}}(n) = \Theta(n \log n)$.

**Proof:** level $j$ has $2^j$ subproblems, each of size $n/2^j$.

$$\text{Total work:} \qquad 2^j \Theta(n/2^j) = \Theta(n).$$

There are $\log_2 n$ internal levels and $n$ leaves:

$$T_{\text{merge}}(n) = \underbrace{\Theta(n \log n)}_{\text{internal levels}} + \underbrace{\Theta(n)}_{\text{leaves}} = \Theta(n \log n).$$

# From two examples to a general recurrence

**Binary search:** 1 per level.

**Merge sort:** $n$ per level.

# From two examples to a general recurrence

**Binary search:** 1 per level. **Merge sort:** $n$ per level.

$$T(n) = \underbrace{aT(n/b)}_{a \text{ calls, each of size } n/b} + \underbrace{cn^d}_{\text{work outside the calls}}.$$

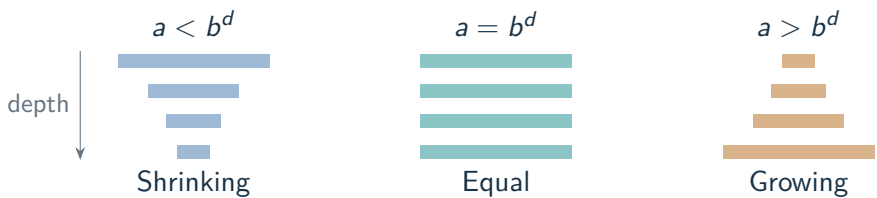
Work at level $j$ : $a^j c(n/b^j)^d = cn^d \left(\frac{a}{b^d}\right)^j$.

# From two examples to a general recurrence

**Binary search:** 1 per level. **Merge sort:** $n$ per level.

$$T(n) = \underbrace{aT(n/b)}_{a \text{ calls, each of size } n/b} + \underbrace{cn^d}_{\text{work outside the calls}}.$$

Work at level $j$ : $a^j c(n/b^j)^d = cn^d \left(\frac{a}{b^d}\right)^j$.



# The Master Theorem

## Polynomial work outside the recursive calls

$$T(n) = aT(n/b) + cn^d, \qquad T(1) > 0.$$

Fixed integers $a \geq 1$, $b \geq 2$; constants $c > 0$, $d \geq 0$; inputs $n = b^k$.

| Compare | Work is concentrated | Result |
|---|---|---|
| $a < b^d$ | Near the root | $\Theta(n^d)$ |
| $a = b^d$ | Evenly across levels | $\Theta(n^d \log n)$ |
| $a > b^d$ | Near the leaves | $\Theta(n^{\log_b a})$ |

# The Master Theorem

## Polynomial work outside the recursive calls

$$T(n) = aT(n/b) + cn^d, \qquad T(1) > 0.$$

Fixed integers $a \geq 1$, $b \geq 2$; constants $c > 0$, $d \geq 0$; inputs $n = b^k$.

| Compare | Work is concentrated | Result |
|---|---|---|
| $a < b^d$ | Near the root | $\Theta(n^d)$ |
| $a = b^d$ | Evenly across levels | $\Theta(n^d \log n)$ |
| $a > b^d$ | Near the leaves | $\Theta(n^{\log_b a})$ |

**Binary search:** $(a, b, d) = (1, 2, 0) \Rightarrow \Theta(\log n)$.

**Merge sort:** $(a, b, d) = (2, 2, 1) \Rightarrow \Theta(n \log n)$.

# Summary

1. **Recursive definitions**
   Bases and construction rules determine the objects.

2. **Structural induction**
   Prove the property for the bases and preserve it under each rule.

3. **Solving recurrences**
   Expand to the base; add the work; verify by induction.

# Review exercises

1. Give a recursive definition of $q_n = 2^n - 1$ for $n \geq 0$.

2. Starting from 3 and closing under addition, prove that the generated set is exactly the positive multiples of 3.

3. For $n = 3^k$, solve and verify
$$S(1) = 2, \qquad S(n) = 3S(n/3) + n.$$

4. For $n = 3^k$, solve
$$R(1) = 1, \qquad R(n) = 9R(n/3) + n^2.$$