# Design and Analysis of Algorithm
## Series Summation and Recurrence Relation

# Outline

## Mathematics of Algorithm Complexity

Algorithm typically consists of loop and iteration structure

- complexity $\rightsquigarrow$ series summation

Method of calculating series summation

- general term formula $\rightsquigarrow$ exact result
- estimate the upper bound of summation $\rightsquigarrow$ approximate result

---

Algorithm may consist of recursive structure

- complexity $\rightsquigarrow$ recurrence relation

Methods of solving recurrence relation

- recurrence relation is simple: direct iteration + substitution iteration
- recurrence relation is complex: simplification + recursion tree
- general case: master theorem

# Concepts of Sequences and Series

Sequence: an ordered list of numbers; the numbers in this ordered list are called the "terms" of the sequence.

## Concepts of Sequences and Series

Sequence: an ordered list of numbers; the numbers in this ordered list are called the “terms” of the sequence.

Series: the sum all the terms of a sequence; the resulting value, are called the “sum” or the “summation”.

## Concepts of Sequences and Series

Sequence: an ordered list of numbers; the numbers in this ordered list are called the "terms" of the sequence.

Series: the sum all the terms of a sequence; the resulting value, are called the "sum" or the "summation".

Example. $1, 2, 3, 4$ is a sequence, with terms "$1$", "$2$", "$3$", "$4$"; the corresponding series is the sum "$1 + 2 + 3 + 4$", and the value of the series is $10$.

Next, we first recall three classical sequences.

# Arithmetic Sequence

Arithmetic Sequence

$$a, a+d, \ldots, a+(n-1)d$$
$$a_i = a + (i-1)d$$
$$\text{common difference} = d \neq 0$$

Arithmetic Series

$$S(n) = \sum_{i=1}^{n} a_i = \frac{n(a_1 + a_n)}{2} = \frac{n(2a + (n-1)d)}{2}$$

# Geometric Sequence

Geometric Sequence

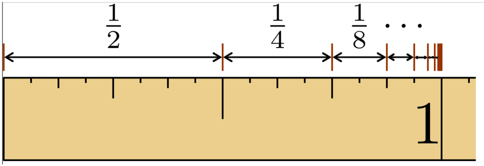
$$a, ar, \ldots, ar^{n-1}$$
$$a_i = ar^{i-1}$$
$$\text{common ratio} = r \neq 1$$

Geometric Series

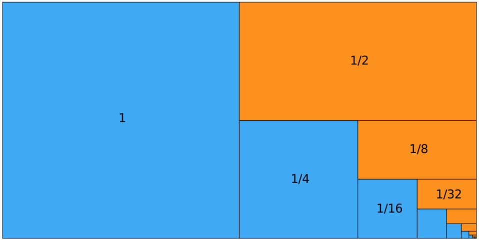
$$S(n) = \sum_{i=1}^{n} ar^{i-1} = a + ar + ar^2 + \cdots + ar^{n-1}$$
$$rS(n) = \sum_{i=1}^{n} ar^{i} = ar + ar^2 + ar^3 + \cdots + ar^{n}$$
$$\Rightarrow S(n) - rS(n) = a - ar^n \Rightarrow$$
$$S(n) = a\left(\frac{1-r^n}{1-r}\right) \qquad \lim_{n\to\infty} S(n) = \frac{a}{1-r}, |r| < 1$$

## Visualization of Geometric Series

$$S(n) = \frac{1}{2} + \frac{1}{4} + \dots$$




$$S(n) = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots$$



## Applications of Geometric Sequence

Geometric Series are among the simplest examples of infinite series with finite sums (although not all of them have this property).

Geometric series are used throughout mathematics, have important applications in physics, engineering, biology, economics, computer science, queueing theory, and finance.

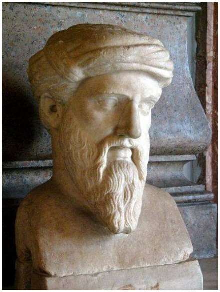
- Repeating decimals (e.g., $0.77777\cdots$) is rational
- Fractal geometry
- Zeno's paradoxes
- Economics: the present value of an annuity
- number of total Bitcoins $\leq 2.1 \times 10^8$

# Harmonic Sequence

Harmonic Sequence (whose inverse forms an arithmetic sequence)

$$1, \frac{1}{2}, \ldots, \frac{1}{n}$$

$$a_i = \frac{1}{i}$$



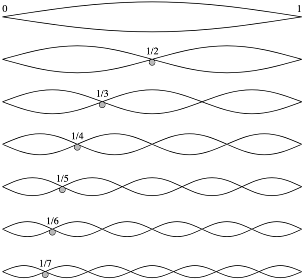
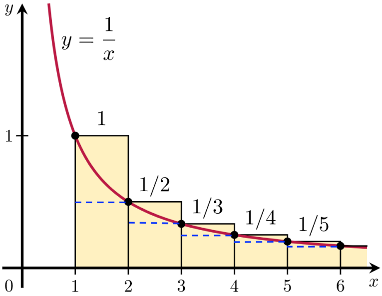


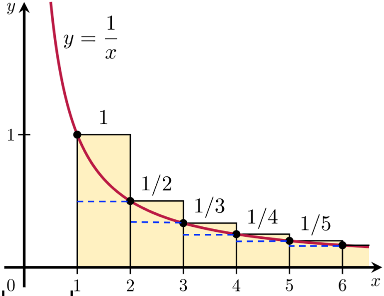

Figure: Pythagoras

# Calculation of Harmonic Series: Integral Test




$$S(n) = \Theta(\ln n)$$

# Calculation of Harmonic Series: Integral Test

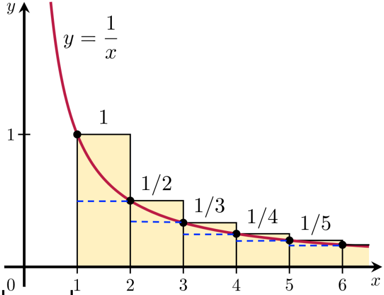



$$S(n) = \Theta(\ln n)$$

Lower bound

$$S(n) = \sum_{i=1}^{n} \frac{1}{i} > \int_{i=1}^{n+1} \frac{1}{x} dx = \ln(n+1)$$

# Calculation of Harmonic Series: Integral Test




$$S(n) = \Theta(\ln n)$$

Lower bound

$$S(n) = \sum_{i=1}^{n} \frac{1}{i} > \int_{i=1}^{n+1} \frac{1}{x} dx = \ln(n+1)$$

Upper bound

$$S(n) = \sum_{i=1}^{n} \frac{1}{i} = 1 + \left( \frac{1}{2} + \cdots + \frac{1}{n} \right) < 1 + \int_{i=1}^{n} \frac{1}{x} dx = \ln n + 1$$

## Interesting Properties of These Sequences

The middle term is the “mean” of its two neighbors

## Interesting Properties of These Sequences

The middle term is the “mean” of its two neighbors

- arithmetic sequences

$$\text{arithmetic mean:} a_{i+1} = \frac{a_i + a_{i+2}}{2}$$

## Interesting Properties of These Sequences

The middle term is the "mean" of its two neighbors

- arithmetic sequences

$$\text{arithmetic mean:} a_{i+1} = \frac{a_i + a_{i+2}}{2}$$

- geometric sequences

$$\text{geometric mean:} a_{i+1} = \sqrt{a_i \cdot a_{i+2}}$$

## Interesting Properties of These Sequences

The middle term is the "mean" of its two neighbors

- arithmetic sequences

$$\text{arithmetic mean:} a_{i+1} = \frac{a_i + a_{i+2}}{2}$$

- geometric sequences

$$\text{geometric mean:} a_{i+1} = \sqrt{a_i \cdot a_{i+2}}$$

- harmonic sequences

$$\text{harmonic mean:} a_{i+1} = \frac{2}{\frac{1}{a_i} + \frac{1}{a_{i+2}}}$$

## Exact Series Summation

$$
\begin{aligned}
\sum_{i=1}^{n} i2^{i-1} &= \sum_{i=1}^{n} i(2^i - 2^{i-1}) && \text{//split terms} \\
&= \sum_{i=1}^{n} i2^i - \sum_{i=1}^{n} i2^{i-1} \\
&= \sum_{i=1}^{n} i2^i - \sum_{i=0}^{n-1} (i+1)2^i && \text{//substitute subscripts} \\
&= \sum_{i=1}^{n} i2^i - \sum_{i=0}^{n-1} i2^i - \boxed{\sum_{i=0}^{n-1} 2^i} && \text{//split terms} \\
&= n2^n - (2^n - 1) = (n-1)2^n + 1 && \text{//geometric series}
\end{aligned}
$$

# Approximate Series Summation

## Amplification method

1. $\sum_{i=1}^{n} a_i \leq n a_{\max}$ (coarse)
2. Assume $\exists\ 0 < r < 1$, s.t. $\forall k \geq 0$ the inequality $a_{i+1}/a_i \leq r$ holds, we can amplify them to geometric series

$$\sum_{i=0}^{n} a_i \leq \sum_{i=0}^{n} a_0 r^i = a_0 \frac{1 - r^{n+1}}{1 - r}$$

## Example of Amplification Method

Estimate the upper bound of $\sum_{i=1}^{n} \frac{i}{3^i}$

Solution.

$$a_i = \frac{i}{3^i}, a_{i+1} = \frac{i+1}{3^{i+1}} \Rightarrow$$
$$\frac{a_{i+1}}{a_i} = \frac{1}{3}\frac{i+1}{i} \leq \frac{2}{3}$$

Apply the amplification method, we have:

$$\sum_{i=1}^{n} \frac{i}{3^i} < \sum_{i=1}^{\infty} \frac{1}{3}\left(\frac{2}{3}\right)^{i-1} = \frac{1}{3}\frac{1}{1-\frac{2}{3}} = 1$$

# Binary Search Algorithm

---

**Algorithm 1:** BinarySearch$(A, l, r, x)$

---

**Input:** $A[l, r]$, target element $x$

**Output:** $j$

$l \leftarrow 1$, $r \leftarrow n$;

**while** $l \leq r$ **do**

    $m \leftarrow \lfloor (l + r)/2 \rfloor$;

    **if** $A[m] = x$ **then return** $m$; $//x$ is the median

    **else if** $A[m] > x$ **then** $r \leftarrow m - 1$;

    **else** $l \leftarrow m + 1$;

**end**

**return** $0$

---

# Demo of Binary Search

1st compare: $3.5 < 4$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | | | 3.5 | | | |

2nd compare: $3.5 > 2$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 3.5 | | | | | |

3rd compare: $3.5 > 3$

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | | 3.5 | | | | |

# Input Size $n$

Ideal case. $n = 2^k - 1$

## Input Size $n$

Ideal case. $n = 2^k - 1$

Q. Why we call $n = 2^k - 1$ as ideal case?

## Input Size $n$

Ideal case. $n = 2^k - 1$

Q. Why we call $n = 2^k - 1$ as ideal case?

A. Because the size of sub-problem is still of the form $2^i - 1$

# Input Size $n$

Ideal case. $n = 2^k - 1$

Q. Why we call $n = 2^k - 1$ as ideal case?

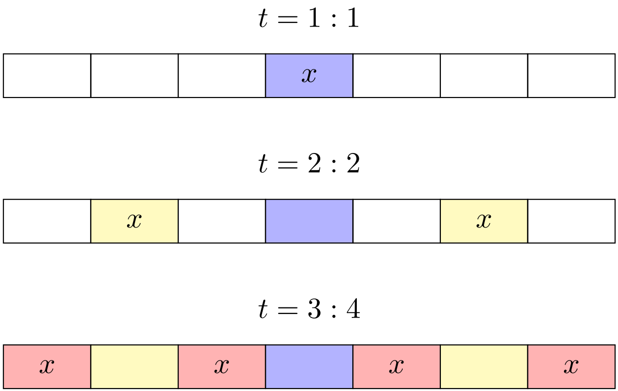
A. Because the size of sub-problem is still of the form $2^i - 1$

---

There are $2n + 1$ possibilities of $x$:

- $x$ in the array: $n$
- $x$ not in the array: fall into $n + 1$ intervals

## Number of input $x$ that requires $t$ times compare ($n = 7, k = 3$)




- for $t \in [k-1]$, # possible input elements that requires $t$ times compares is $2^{t-1}$
- for $t = k$, # possible input elements that requires $k$ times compares is $2^{k-1} + (n+1)$

## Average-case Complexity of Binary Search

Let $n = 2^k - 1$, assume $x$ appears at each position with the same probability:

$$
\begin{aligned}
T(n) &= \frac{1}{2n+1}\left(\sum_{t=1}^{k-1} t2^{t-1} + k(2^{k-1} + n + 1)\right) \\
&= \frac{1}{2n+1}\left(\boxed{\sum_{t=1}^{k-1} t2^{t-1}} + k(2^{k-1} + 2^k)\right) \\
&= \frac{1}{2n+1}\left((k-2)2^{k-1} + 1 + k2^k + k2^{k-1}\right) \quad \text{//use previous result} \\
&= \frac{k2^k - 2^k + 1 + k2^k}{2n+1} \\
&= \frac{(2k-1)2^k + 1}{2^{k+1} - 1} \approx k - \frac{1}{2} = \Theta(\log n)
\end{aligned}
$$

# Outline

# Motivation

Recursion and Iteration are two commonly used programming paradigm

Common: Solution to a problem is obtained by combining solutions to subproblems of smaller size.

In this case, time complexity functions can be expressed as recurrence relations.

*How to solve recurrence relations?*

# Recurrence Relation

**Definition 1 (Recurrence Relation)**

Let $a_0, a_1, \ldots, a_n$ be a sequence, shorthand as $\{a_n\}$. A recurrence relation defines each term of a sequence using preceding term(s), and always state the initial term of the sequence.

*Recurrence relation captures the dependence of a term to its preceding terms.*

Solution. Given recurrence relation for a sequence $\{a_n\}$ together with some initial values, compute the general term formula of $a_n$.

- general term formula: a function of $n$, without involvement of other terms

## Example of Recurrence Relation: Fibonacci Number

Fibonacci number:
$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$

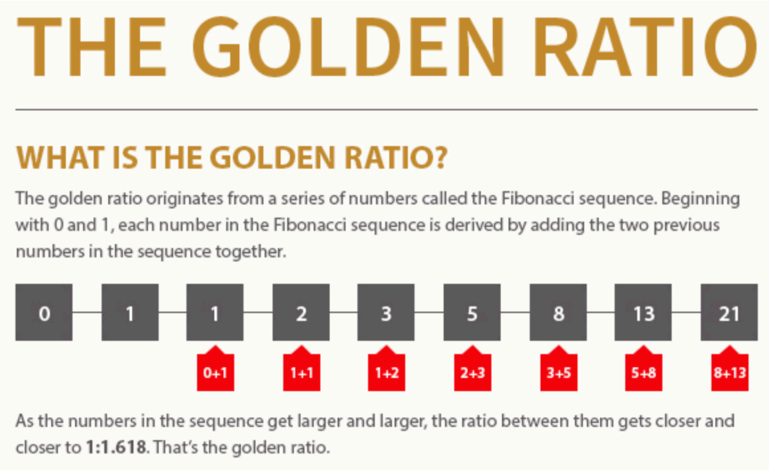
Recurrence relation:
$f_n = f_{n-1} + f_{n-2}$

Initial value: $f_0 = 1,\ f_1 = 1$

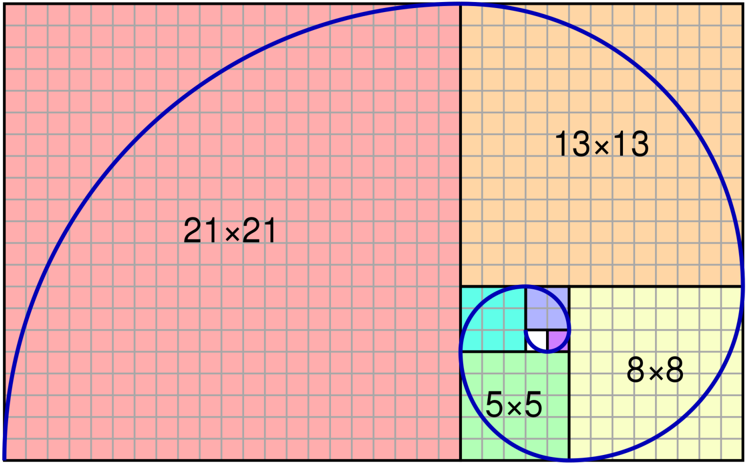
Figure: Fibonacci

$$\boxed{f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}}$$

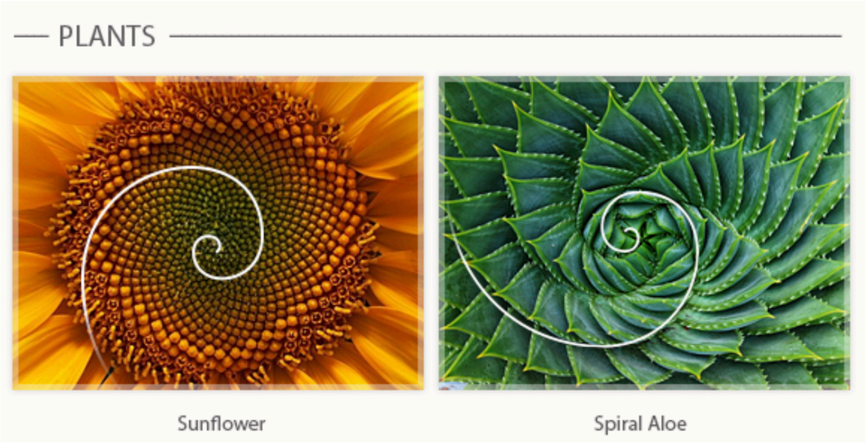
# The Nature of Design: the Fibonacci Number and the Golden Ratio



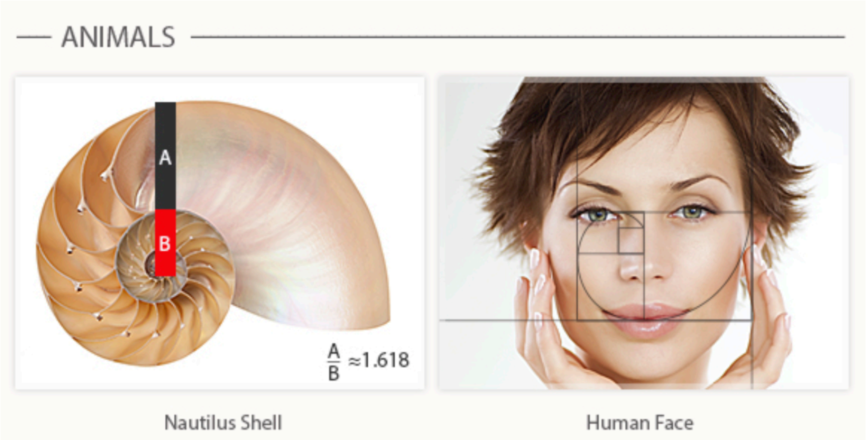
# Visualizing the Golden Ratio



# The Golden Ratio is Everywhere (Plant)



# The Golden Ratio is Everywhere (Animal)



# The Golden Ratio is Everywhere (Animal)

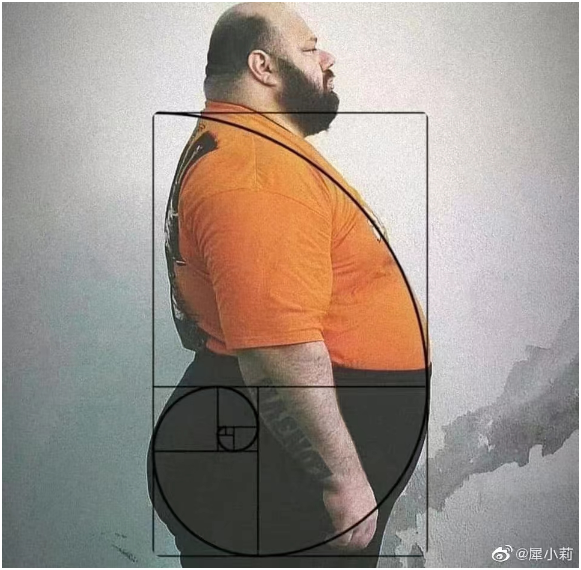
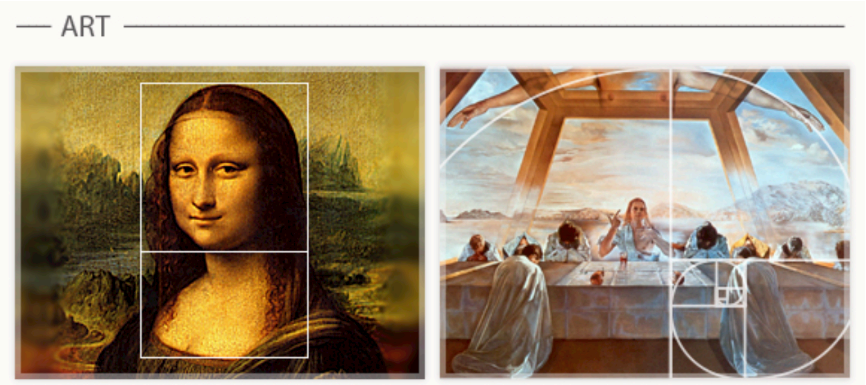



Figure: 男人四十一枝花

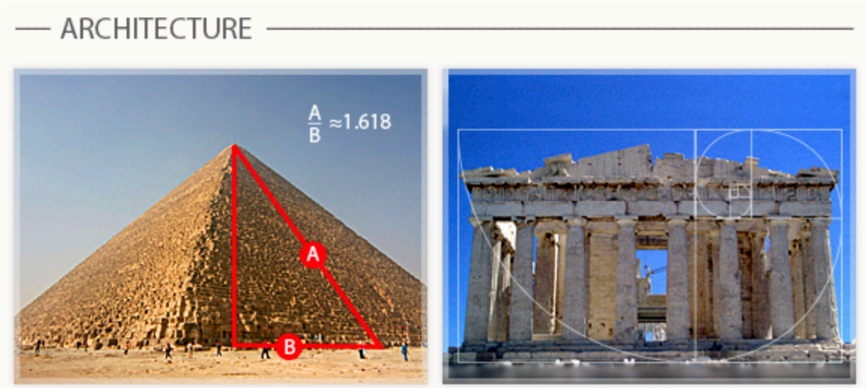
# The Golden Ratio is Everywhere (Art)




Da Vinci's Mona Lisa

Dali's Sacrament of the Last Supper

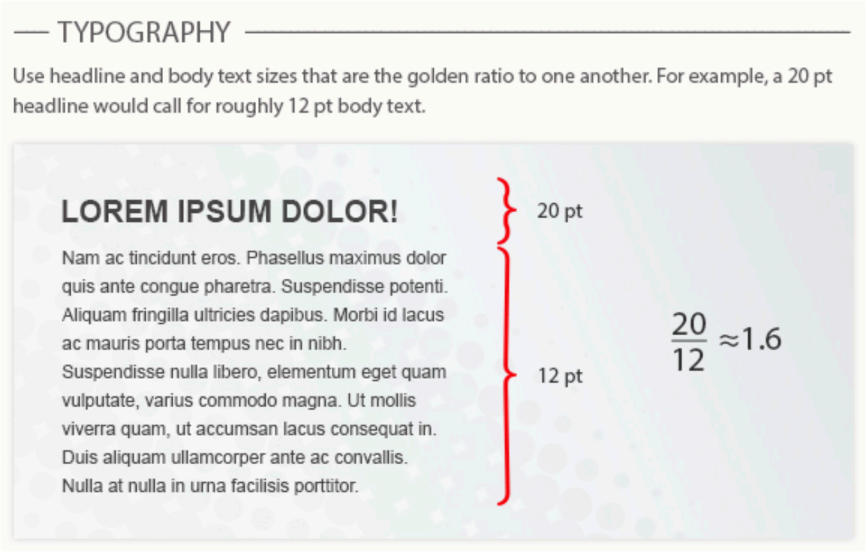
# The Golden Ratio is Everywhere (Architecture)




Great Pyramid of Giza

Parthenon

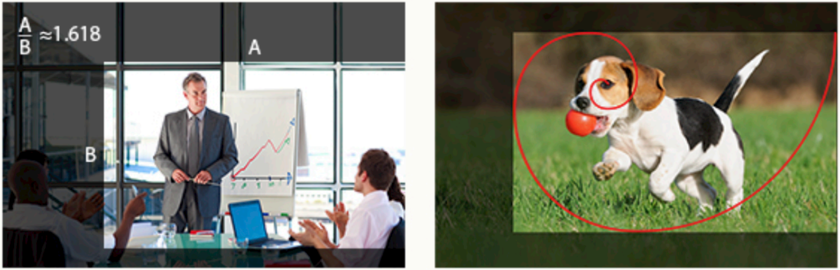
# The Golden Ratio is Everywhere (Typography)



# The Golden Ratio is Everywhere (Sizing/Cropping Images)

## SIZING/CROPPING IMAGES

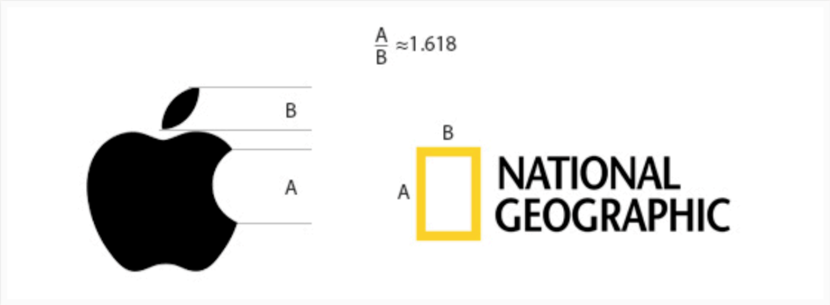
Use the golden ratio as your guide for image proportions and for drawing focus to the most important elements.



# The Golden Ratio is Everywhere (Shapes and Symbols)

## SHAPES AND SYMBOLS

Use the golden ratio to add interest to vector-based shapes. Many major companies, for example, have used golden proportions in the design of their logos.



Next, we introduce three methods for solving recurrence relation.

# Outline

## Steps of Direct Iteration

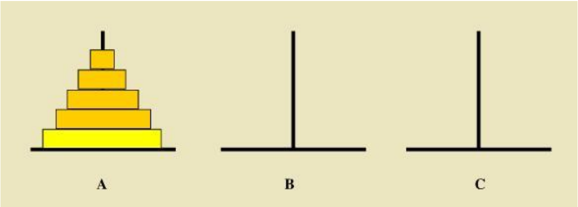
When the recurrence relation is simple, i.e., $F(n)$ only relies on $F(n-1)$, we use direct iteration.

1. Continuously substitute the "right part" of formula with the "right right part"
2. After each substitution, a new term emerged in the series as $n$ decreases
3. Stop substitution until reaching the initial values
4. Calculate the series with the initial values
5. Use mathematical induction to check the correctness of the solution

Remark. Mathematical induction is useful for testing if your guess is correct. When the correctness is evident, it is not necessary.

## Example: Hanoi Tower Problem

Origin. *When creating the world, Brahma also built three diamond rods and 64 golden disks of different sizes. At the beginning, the disks place in ascending order of size on one rod, the smallest at the top, thus making a conical shape. Brahmin priests have been moving these disks from one rod to another rod in accordance with the immutable rules of Brahma since that time. When the last move is completed, the world will end.*



## Problem as a Puzzle

Problem Abstract. There are three rods (labeled as $A, B, C$) with $n$ of disks of different sizes. At the beginning, the disks in a neat stack in ascending order of size on rod $A$. The objective of figure out the minimum number of moves $T(n)$ required to move the entire stack to rod $C$, obeying the following rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.

## Problem as a Puzzle

Problem Abstract. There are three rods (labeled as $A, B, C$) with $n$ of disks of different sizes. At the beginning, the disks in a neat stack in ascending order of size on rod $A$. The objective of figure out the minimum number of moves $T(n)$ required to move the entire stack to rod $C$, obeying the following rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.

Example. $n = 1$, $T(1) = 1$; $n = 2$, $T(2) = 3$; $n = 3$, $T(3) = 7$;

## Problem as a Puzzle

Problem Abstract. There are three rods (labeled as $A, B, C$) with $n$ of disks of different sizes. At the beginning, the disks in a neat stack in ascending order of size on rod $A$. The objective of figure out the minimum number of moves $T(n)$ required to move the entire stack to rod $C$, obeying the following rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.

Example. $n = 1$, $T(1) = 1$; $n = 2$, $T(2) = 3$; $n = 3$, $T(3) = 7$;

$$\text{general form } T(n) = ?$$

## Recursive Algorithm for Hanoi Tower

**Algorithm 2:** Hanoi$(A, C, n)$ // move $n$ disks from $A$ to $C$

```
Input: A(n), B(0), C(0)
Output: A(0), B(0), C(n)
if n = 1 then move (A, C); //one disk from A to C
else
    Hanoi(A, B, n − 1) // use C as swap tower;
    move (A, C);
    Hanoi(B, C, n − 1) // use A as swap tower;
end
```

Let $T(n)$ be the number of moves required to move $n$ disks

- $T(n) = 2T(n-1) + 1$
- $T(1) = 1$

# Complexity Analysis: Direct Iteration

$$\left.\begin{array}{c} T(n) = 2T(n-1) + 1 \\ T(1) = 1 \end{array}\right\} \Rightarrow T(n) = 2^n - 1$$

## Complexity Analysis: Direct Iteration

$$\left.\begin{array}{c} T(n) = 2T(n-1) + 1 \\ T(1) = 1 \end{array}\right\} \Rightarrow T(n) = 2^n - 1$$

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 2^2 T(n-2) + 2 + 1 \\ &= \ldots \\ &= 2^{n-1} T(1) + \underline{2^{n-2} + \cdots + 2 + 1} // \text{reaching the initial terms} \\ &= 2^{n-1} \cdot 1 + 2^{n-1} - 1 // \text{substitute with initial values} \\ &= 2^n - 1 \end{aligned}$$

## More about Hanoi Tower

Is there a better algorithm?

# More about Hanoi Tower

Is there a better algorithm?

No! Tower of Hanoi is an intractable problem, no polynomial time algorithm is known.

## More about Hanoi Tower

Is there a better algorithm?

No! Tower of Hanoi is an intractable problem, no polynomial time algorithm is known.

---

Q. $1$ move/s, how many times needed to move $64$ disks?

# More about Hanoi Tower

Is there a better algorithm?

No! Tower of Hanoi is an intractable problem, no polynomial time algorithm is known.

---

Q. $1$ move/s, how many times needed to move $64$ disks?

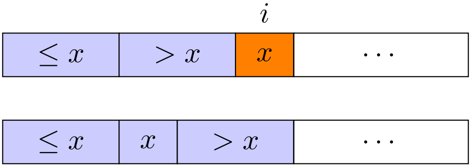
A. $500$ billion years! Bad news for algorithm but good news to the world!

## Example: Iterated Algorithm for Insertion Sort

**Algorithm 3:** InsertionSort$(A, n)$

**Input:** unsorted $A[n]$
**Output:** $A[n]$ in ascending order

**for** $j \leftarrow 2$ *to* $n$ **do**
    $x \leftarrow A[j]$;
    $i \leftarrow j - 1$ //insert $A[j]$ to $A[1 \ldots j-1]$;
    **while** $i > 0$ *and* $x < A[i]$ **do**
        $A[i+1] \leftarrow A[i]$;
        $i \leftarrow i - 1$;
    **end**
    $A[i+1] \leftarrow x$;
**end**



## Worse-case Complexity

Basic computer step. element compare

Input size. $n$

$$\left.\begin{array}{c} W(n) = W(n-1) + (n-1) \\ W(1) = 0 \end{array}\right\} \Rightarrow W(n) = n(n-1)/2$$

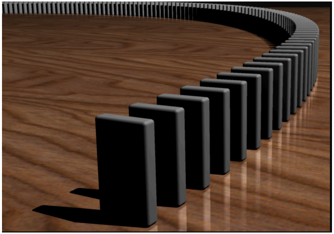
- When inserting the $i$-th element, algorithm compares it with the first $i-1$ sorted elements; the maximum number of compare is $i-1$.

## Solve Recurrence Relation by Direct Iteration

$$
\begin{aligned}
W(n) &= W(n-1) + n - 1 \\
&= (W(n-2) + n - 2) + n - 1 \\
&= W(n-2) + n - 2 + n - 1 \\
&= \ldots \\
&= W(1) + \underline{1 + 2 + \cdots + (n-2) + (n-1)} \text{//reach the initial term} \\
&= 0 + \underline{1 + 2 + \cdots + (n-2) + (n-1)} \text{//substitute with initial value} \\
&= n(n-1)/2
\end{aligned}
$$

# Mathematical Induction (date back to 370 BC, Plato's Parmenides)

Mathematical induction is a mathematical proof technique $\Rightarrow$ prove that a property $P(n)$ holds for every natural number $n \in \mathbb{N}$.



*Mathematical induction proves that we can climb as high as we like on a ladder, by proving that we can climb onto the bottom rung (the basis) and that from each rung we can climb up to the next one (the step).* — Concrete Mathematics

## Template of Mathematical Induction

The method of induction requires two facts to be proved.

Induction basis: Prove the property holds for number $0$.

Induction step

1. Prove that if the property holds for one natural number $n$, then it holds for the next natural number $n+1$

$$P(0) = 1$$
$$\forall n, P(n) = 1 \Rightarrow P(n+1) = 1$$
$$n = 0, P(0) \Rightarrow P(1); n = 1, P(1) \Rightarrow P(2) \ldots$$

2. Prove that the the property holds for all natural number $k < n$, then it also holds for $n$.

$$P(0) = 1$$
$$\forall k < n, P(k) = 1 \Rightarrow P(n) = 1$$
$$n = 1, P(0) = 1 \Rightarrow P(1) = 1;$$
$$n = 2, P(0) = 1 \wedge P(1) = 1 \Rightarrow P(2) = 1 \ldots$$

## Comparsion Between Two Types of Mathematics Induction

Induction basis is same: $P(0) = 1$

Induction step is different

Logic reasoning:

1. Type 1 induction: $P(0) = 1 \Rightarrow P(1) = 1 \Rightarrow P(2) = 1$
2. Type 2 induction:
   $P(0) = 1 \Rightarrow P(0) = 1 \wedge P(1) = 1 \Rightarrow P(0) = 1 \wedge P(1) = 1 \wedge P(2) = 1 \Rightarrow P(0) = 1 \wedge P(1) = 1 \wedge P(2) = 1 \wedge P(3) = 1$

Think. The intutions are same, when to apply which?

- Type 1 (loose coupling): the property of next number only depends on its nearest preceding
- Type 2 (tight coupling): the property of next number depends on all its precedings

# Remarks on Mathematic Induction

These two steps establish the property $P(n) = 1$ for every natural number $n = 0, 1, 2, 3$.

## Remarks on Mathematic Induction

These two steps establish the property $P(n) = 1$ for every natural number $n = 0, 1, 2, 3$.

- The base case does not necessarily begin with $n = 0$. It can begin with any natural number $n_0$, establishing the truth of $P(n) = 1$ holds for all $n \geq n_0$.

# Remarks on Mathematic Induction

These two steps establish the property $P(n) = 1$ for every natural number $n = 0, 1, 2, 3$.

- The base case does not necessarily begin with $n = 0$. It can begin with any natural number $n_0$, establishing the truth of $P(n) = 1$ holds for all $n \geq n_0$.
- The method can be extended to structural induction ⇒prove statements about more general well-founded structures, such as trees (widely used in mathematical logic and computer science).

## Verify Correctness of Solution: Mathematical Induction

Proposition. $W(n) = n(n-1)/2$ is the general term formula for recurrence relation

$$\begin{cases} W(n) = W(n-1) + n - 1 \\ W(1) = 0 \end{cases}$$

# Verify Correctness of Solution: Mathematical Induction

Proposition. $W(n) = n(n-1)/2$ is the general term formula for recurrence relation

$$\begin{cases} W(n) = W(n-1) + n - 1 \\ W(1) = 0 \end{cases}$$

Method: Mathematical Induction

1. Basis: $n = 1$, $W(1) = 1 \times (1-1)/2 = 0$
2. Induction step: $P(n) = 1 \Rightarrow P(n+1) = 1$:

$$\begin{aligned} W(n+1) &= W(n) + n \\ &= \underline{n(n-1)/2} + n \quad \text{//premise} \\ &= n((n-1)/2 + 1) = n(n+1)/2 \end{aligned}$$

## Variant of Direct Iteration: Substitution-then-Iteration

When $n$ itself is a function of another variable, say, $k$, and $k$ decreases 1 after each iteration, we first substitute $n$ by the function of $k$, then apply the iteration approach over $k$.

1. Transform the recursive formula about $n$ to recursive formula about $k$
2. Iterate over $k$
3. Transform the general term formula about $k$ back to general term formula about $n$

# MergeSort Algorithm

**Algorithm 4:** MergeSort$(A, n)$

**Input:** unsorted $A[n]$

**Output:** sorted $A[n]$ in ascending order

$l \leftarrow 1$, $r \leftarrow n$;
**if** $l < r$ **then**
    $k \leftarrow \lfloor (l + r)/2 \rfloor$;
    MergeSort$(A, l, k)$;
    MergeSort$(A, k + 1, r)$;
    Merge$(A, p, k, r)$
**end**

## Example of Substitution-and-Iteration (1/2)

Assume $n = 2^k$, the recurrence relation is:

$$\begin{cases} W(n) = 2W(n/2) + n - 1 \\ W(1) = 0 \end{cases}$$

- $n - 1$ is the cost of merge

Substitution: $n \rightarrow 2^k$

$$\begin{cases} W(2^k) = 2W(2^{k-1}) + 2^k - 1 \\ W(2^0 = 1) = 0 \end{cases}$$

## Example of Substitution-and-Iteration (2/2)

$W(n)$

$= 2W(2^{k-1}) + 2^k - 1$ //substitute and iterate on $k$

$= 2(2W(2^{k-2}) + 2^{k-1} - 1) + 2^k - 1$ //1st round iteration

$= 2^2W(2^{k-2}) + 2^k - 2 + 2^k - 1$

//2nd round iteration

$= 2^2(2W(2^{k-3}) + 2^{k-2} - 1) + 2^k - 2 + 2^k - 1$

$= \dots$

$= 2^kW(2^0 = 1) + k2^k - (2^{k-1} + 2^{k-2} + \cdots + 2 + 1)$

$= 0 + k2^k - 2^k + 1$

$= n \log n - n + 1$ //substitute back

## Outline

# Simplification-then-Iteration

## Motivation

- Basic approach for solving recurrence relation is iteration
- When the original recurrence relation is complex, we need simplification
  - transform high order equation to one order equation (reduce dependence), then substitute

# Example of QuickSort

## Recap of QuickSort

Suppose the elements in $A[n]$ are distinct, set $l \leftarrow 1$, $r \leftarrow n$, partition $A[l \ldots r]$ with the first element $A[1] = x$, such that

- elements less than $x$ are stored in $A[l \ldots k-1]$
- elements greater than $x$ are stored in $A[k+1 \ldots r]$
- $A[1]$ is placed in $A[k]$

sort $A[l \ldots k-1]$ and $A[k+1 \ldots r]$ recursively

## Overall complexity.

- complexity of subproblems
- complexity of partition

## Input and Subproblem Size

According to the final position of the first element $x$ in the resulting sorted array, we can break input to $n$ cases

| final position of $x$ | size of subproblem-1 | size of subproblem-2 |
|---|---|---|
| $1$ | $0$ | $n-1$ |
| $2$ | $1$ | $n-2$ |
| $3$ | $2$ | $n-3$ |
| $\dots$ | $\dots$ | $\dots$ |
| $n-1$ | $n-2$ | $1$ |
| $n$ | $n-1$ | $0$ |

- For each input, the number of compares required for partition is exactly $n-1$ (think why?)

# Summation of Complexity

$$T(0) + T(n-1) + n - 1$$
$$T(1) + T(n-2) + n - 1$$
$$T(2) + T(n-3) + n - 1$$
$$\dots$$
$$T(n-1) + T(0) + n - 1$$

Summation: $2(T(1) + \cdots + T(n-1)) + n(n-1)$

# Average Complexity of QuickSort

Assumption. The first element $x$ finally appears at each position with equal probability:

$$T(n) = \frac{2}{n}\sum_{i=1}^{n-1} T(i) + O(n), n \geq 2$$
$$T(1) = 0$$
$$T(0) = 0$$

Observation and Idea

- The recurrence relation is complex: $n$-th term depends on all preceding terms $\rightsquigarrow$ direct iteration would be very complex
- Idea: Simplify the complex equation, then iterate

## Simplification via Subtraction

Rewrite and iterate once to obtain two recurrence relations, then try to simplify the terms of the right side.

$$T(n) = \frac{2}{n}\sum_{i=1}^{n-1} T(i) + n - 1$$

$$nT(n) = \boxed{2\sum_{i=1}^{n-1} T(i)} + n(n-1)$$

$$(n-1)T(n-1) = \boxed{2\sum_{i=1}^{n-2} T(i)} + (n-1)(n-2)$$

Subtraction

$$nT(n) - \underline{(n-1)T(n-1)} = \underline{2T(n-1)} + 2(n-1)$$

Simplification

$$nT(n) = (n+1)T(n-1) + \Theta(n)$$

Rewrite

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{\Theta(n)}{n(n+1)} = \frac{T(n-1)}{n} + \frac{\Theta(1)}{n+1}$$

## Iteration

$$
\begin{aligned}
\frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + \frac{\Theta(1)}{n+1} = \dots \\
&= \Theta(1)\left(\frac{1}{n+1} + \frac{1}{n} + \cdots + \frac{1}{3}\right) + \frac{T(1)}{2} \quad \text{//reach the initial term} \\
&= \Theta(1)\left(\frac{1}{n+1} + \frac{1}{n} + \cdots + \frac{1}{3}\right) \quad \text{//substitute with initial value} \\
&= \Theta(\ln n)
\end{aligned}
$$

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

# Outline

## Concept of Recursion Tree

When $F(n)$ relies on several non-consecutive preceding terms, we could try solving the recurrence relation using recursion tree.

- Recursion tree is the model of recursive computation, also the iteration of recurrence relation
- The generation of recursion tree is same as that of recursion process
- The nodes on the recursion tree is exactly the terms in the series of recursion
- The summation of all nodes (including the internal and leaf nodes) on the recursion tree is the solution to the recurrence relation

# Representation of Iteration in Recursion Tree

Recursion tree is the model of recursion $\Rightarrow$ closely related to solving for recurrence relation

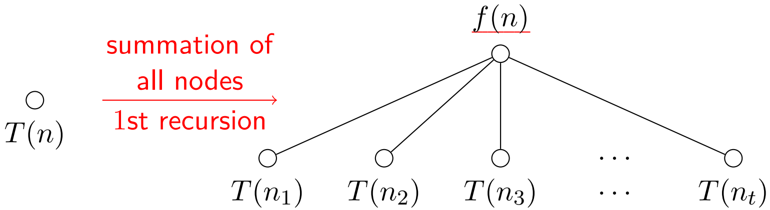
Assume the recurrence relation is as below:

$$T(n) = T(n_1) + \cdots + T(n_t) + f(n), |n_1|, \ldots, |n_t| < |m|$$

- $T(n_1), \ldots, T(n_t)$: function items
- $f(n)$: dividing cost + merging cost

*How to represent $T(n)$ on the recursion tree?*

## Visualization of Recursion Tree

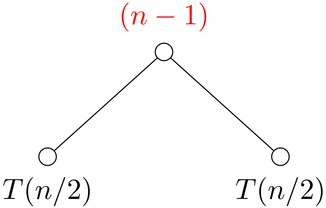



root node is the combine and divide cost
each leaf node is a function term

## Example of 2-Level Recursion Tree

Recurrence relation for MergeSort

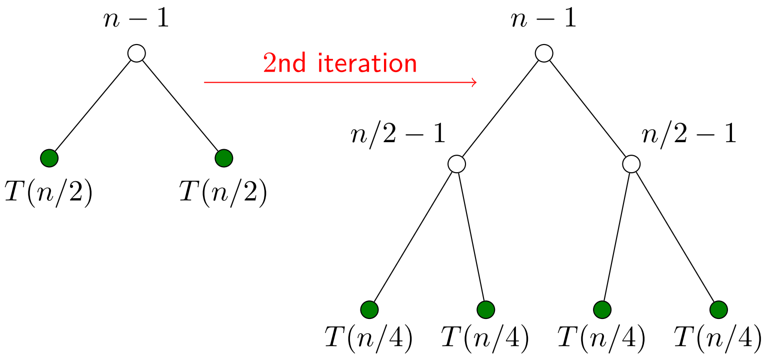
$$\begin{cases} T(n) = 2T(n/2) + (n-1) \\ T(2) = 1 \\ T(1) = 0 \end{cases}$$



## The Generation Rules of Recursion Tree

1. At the very beginning, there is only the root node in the recursion tree, whose value is $T(n)$
2. Repeat the following steps:
   - represent the function term $T(n)$ in the leaf node as a 2-level subtree
   - replace the leaf node with this subtree
3. Continue the generation of recursion tree until there is no function term in the tree.
   - Reaching the leaf nodes — initial values

## Demo of Balanced Recursion Tree Generation



## The Whole Recursion Tree

$$n-1 \qquad\qquad \boxed{n-1}$$

$$\frac{n}{2}-1 \qquad \frac{n}{2}-1 \qquad\qquad \boxed{n-2}$$

$$\frac{n}{4}-1 \qquad \frac{n}{4}-1 \qquad \frac{n}{4}-1 \qquad \frac{n}{4}-1 \qquad\qquad \boxed{n-4}$$

$$\cdots\cdots$$

$$1 \qquad \cdots\cdots \qquad 1 \qquad\qquad \boxed{n-2^{k-1}}$$

$$0 \quad 0 \qquad \cdots\cdots \qquad 0 \quad 0$$

# Calculate the Sum of Recursion Tree (balanced setting)

$$\begin{cases} T(n) = 2T(n/2) + n - 1, n = 2^k \\ T(1) = 0 \end{cases}$$

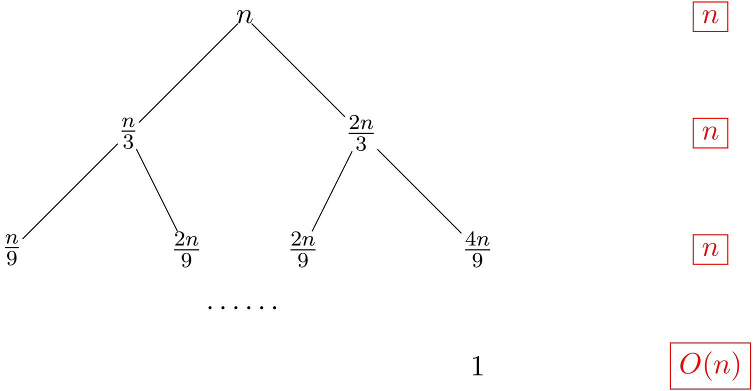
---

$$\begin{aligned} T(n) &= \overbrace{(n-1)}^{\text{0-level}} + \overbrace{(n-2)}^{\text{1-level}} + \cdots + \overbrace{(n-2^{k-1})}^{(k-1)\text{-level}} \\ &= kn - (2^k - 1) \quad //k = \log n \\ &= n\log n - n + 1 \end{aligned}$$

## Application of Recursion Tree (unbalanced setting)

Compute the general term formula of

$$T(n) = T(n/3) + T(2n/3) + n$$




The rates that different routes reach the initial value are different

- the left route is fastest – estimate the lower bound
- the right route is slowest – estimate the upper bound

## Calculate the Sum of Recursion Tree (unbalanced setting)

Recurrence relation: $T(n) = T(n/3) + T(2n/3) + n$

The depth of recursion tree is $k$, the sum of each level is $O(n)$

- Estimate the longest route to calculate for the upper bound

$$n\left(\frac{2}{3}\right)^k = 1 \Rightarrow \left(\frac{3}{2}\right)^k = n \Rightarrow k = \log_{3/2} n$$

$$T(n) < \log_{3/2} n \times n = O(n \log n)$$

---

- Estimate the shortest route to calculate the lower bound

$$T(n) > \log_3 n \times n = \Omega(n \log n)$$

Putting all the above together, $T(n) = \Theta(n \log n)$

## Remark

For the sake of simplicity, the leaf nodes that represent initial values are not included in the summation.

- The initial values usually cannot be represented by $f(n)$.
- The initial values are usually constants, such as $0$ or $1$, and thus they can be easily calculated separately.

# Outline

## Application of Master Theorem

Solving recurrence relation

$$T(n) = aT(n/b) + f(n)$$

- $a$: the number of subproblems after dividing
- $n/b$: the size of subproblems
- $f(n)$: the cost of dividing and merging subproblems

---

Examples

- binary search: $T(n) = T(n/2) + 1$
- merge sort: $T(n) = 2T(n/2) + n - 1$

## Master Theorem

Let $a \geq 1$, $b \geq 1$ be constants, $T(n)$ and $f(n)$ be functions, and

$$T(n) = aT(n/b) + f(n)$$

1. if $\exists \varepsilon > 0$ s.t. $f(n) = O(n^{(\log_b a) - \varepsilon})$, then:

$$T(n) = \Theta(n^{\log_b a})$$

2. if $f(n) = \Theta(n^{\log_b a})$, then:

$$T(n) = \Theta(n^{\log_b a} \log n)$$

3. if $\exists \varepsilon > 0$ s.t. $f(n) = \Omega(n^{(\log_b a) + \varepsilon})$, and $\exists r < 1$ s.t. for all $n$ (can be relaxed to for sufficiently large $n$) the inequality $af(n/b) \leq rf(n)$ holds, then:

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

*How to prove the master theorem?*

## Direct Iteration

$$T(n) = aT(n/b) + f(n)$$

For the sake of convenience, let $n = b^k$

$$\begin{aligned} T(n) &= aT\left(\frac{n}{b}\right) + f(n) \\ &= a\left(aT\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b}\right)\right) + f(n) \\ &= a^2T\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) + f(n) \\ &= \ldots \end{aligned}$$

## Result of Iteration

$$= a^k T\left(\frac{n}{b^k}\right) + a^{k-1} f\left(\frac{n}{b^{k-1}}\right) + \cdots + af\left(\frac{n}{b}\right) + f(n)$$

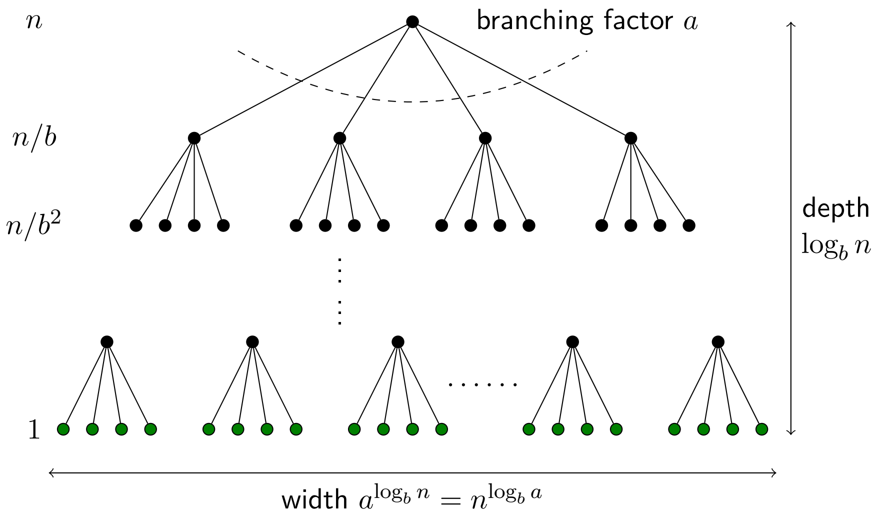
$$= a^k T(1) + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right) \quad \text{//reach the initial term}$$

$$= \underline{c_1 n^{\log_b a}} + \sum_{j=0}^{k-1} a^j f\left(\frac{n}{b^j}\right) \quad \text{//assume } T(1) = c_1$$

$$\boxed{k = \log_b n = \log_b a \cdot \log_a n}$$

- the first term is the total costs of all base subproblems
- the second term is the total costs of all dividing and merging steps

# Corresponding Recursion Tree



## Explaining the Meaning of Recursion Tree

- $a$: branching factor
- $b$: the size of subproblems decreases by a factor of $b$ with each level of recursion
- $k = \log_b n$: reaches the base case after $k$ levels, the height of the recursion tree

the $j$th level of the tree is made up of $a^j$ subproblems, each of size $n/b^j$

## Case 1

$$\exists \varepsilon > 0 \text{ s.t. } f(n) = O(n^{(\log_b a)-\varepsilon})$$

$$T(n)$$

$$= c_1 n^{\log_b a} + O\left(\sum_{j=0}^{(\log_b n)-1} a^j \left(\frac{n}{b^j}\right)^{(\log_b a)-\varepsilon}\right)$$ //substitute with premise

$$= c_1 n^{\log_b a} + O\left(n^{(\log_b a)-\varepsilon} \sum_{j=0}^{(\log_b n)-1} \frac{a^j}{\left(b^{(\log_b a)-\varepsilon}\right)^j}\right)$$

//move sum irrelevant terms outside

## Case 1: Continue to Simplify

$$\boxed{\frac{1}{\left(b^{\log_b a-\varepsilon}\right)^j}=\frac{b^{\varepsilon j}}{\left(b^{\log_b a}\right)^j}=\frac{b^{\varepsilon j}}{a^j}}$$

$$= c_1 n^{\log_b a} + O\left(n^{(\log_b a)-\varepsilon}\sum_{j=0}^{\log_b n-1}\frac{a^j}{\left(b^{(\log_b a)-\varepsilon}\right)^j}\right)$$ //simplify

$$= c_1 n^{\log_b a} + O\left(n^{(\log_b a)-\varepsilon}\boxed{\sum_{j=0}^{\log_b n-1}(b^{\varepsilon})^j}\right)$$ //geometric series

$$= c_1 n^{\log_b a} + O\left(n^{(\log_b a)-\varepsilon}\frac{b^{\varepsilon\log_b n}-1}{b^{\varepsilon}-1}\right)$$ //ignore the constants

$$= c_1 n^{\log_b a} + O\left(n^{(\log_b a)-\varepsilon}n^{\varepsilon}\right)=\Theta(n^{\log_b a})$$

## Case 2

$$f(n) = \Theta(n^{\log_b a})$$

$$
\begin{aligned}
&T(n) \\
&= c_1 n^{\log_b a} + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right) \\
&= c_1 n^{\log_b a} + \Theta\left(\sum_{j=0}^{(\log_b n)-1} a^j \boxed{\left(\frac{n}{b^j}\right)^{\log_b a}}\right) \qquad \text{//substitute with premise}
\end{aligned}
$$

//move sum irrelevant terms outside

$$
\begin{aligned}
&= c_1 n^{\log_b a} + \Theta\left(n^{\log_b a} \sum_{j=0}^{(\log_b n)-1} \frac{a^j}{a^j}\right) \\
&= c_1 n^{\log_b a} + \Theta(n^{\log_b a} \log_b n) = \Theta(n^{\log_b a} \log n)
\end{aligned}
$$

## Case 3

$$\exists \varepsilon > 0, f(n) = \Omega(n^{(\log_b a)+\varepsilon}) \tag{1}$$
$$a f(n/b) \leq r f(n) \tag{2}$$

Repeatedly apply Condition (2)

$$a^j f\left(\frac{n}{b^j}\right) \leq a^{j-1} r f\left(\frac{n}{b^{j-1}}\right) \leq \cdots \leq r^j f(n)$$

$$\begin{aligned} T(n) &= c_1 n^{\log_b a} + \sum_{j=0}^{(\log_b n)-1} a^j f\left(\frac{n}{b^j}\right) \\ &= c_1 n^{\log_b a} + f(n) + \sum_{j=1}^{(\log_b n)-1} a^j f\left(\frac{n}{b^j}\right) \end{aligned}$$

## Case 3 (continue)

$$
\begin{aligned}
T(n) &\leq c_1 n^{\log_b a} + f(n) + \sum_{j=1}^{(\log_b n)-2} \boxed{r^j f(n)} \\
&= c_1 n^{\log_b a} + f(n) + f(n) r \frac{1 - r^{(\log_b n)-2}}{1 - r} \quad \text{//geometric series: } r < 1 \\
&= c_1 n^{\log_b a} + \Theta(f(n))
\end{aligned}
$$

$$\text{Condition 1} \Rightarrow \text{order}(f(n)) \geq \text{order}(n^{\log_b a})$$

Therefore, we have:

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

- Recap: Condition 2 is used to prove the coefficient of $f(n)$ is upper bounded by a constant.

## Simplified Form of Master Theorem


Define $h(n) = n^{\log_b a}$, we re-state master theorem as below:

$$T(n) = \begin{cases} \Theta(h(n)) & \text{if } f(n) = o(h(n)) \\ \Theta(h(n)\log n) & \text{if } f(n) = \Theta(h(n)) \\ \Theta(f(n)) & \text{if } f(n) = \omega(h(n)) \\ & \wedge \exists\, r < 1 \text{ s.t. } af(n/b) < rf(n) \end{cases}$$

## Outline

## Example 1 of Solving Recurrence Relation

Compute the general term formula of recurrence relation:

$$T(n) = 9T(n/3) + n$$

Applying the master theorem

- $a = 9$, $b = 3$, $h(n) = n^2$, $\varepsilon = 1$;
- $f(n) = n$, $n^{\log_3 9} = n^2$, $f(n) = O(n^{\log_3 9 - 1}) = o(n^2)$

Master theorem (case 1) $\Rightarrow T(n) = \Theta(n^2)$

## Example 2 of Solving Recurrence Relation

Compute the general term formula of recurrence relation:

$$T(n) = T(2n/3) + 1$$

Applying the master theorem

- $a = 1$, $b = 3/2$, $h(n) = n^{\log_b a} = n^0 = 1$;
- $f(n) = 1$, $n^{\log_{3/2} 1} = n^0 = 1$, $f(n) = \Theta(1)$

Master theorem (case 2) $\Rightarrow T(n) = \Theta(\log n)$

# Example 3 of Solving Recurrence Relation

Compute the general term formula of recurrence relation:

$$T(n) = 3T(n/4) + n \log n$$

Applying the master theorem

- $a = 3$, $b = 4$, $h(n) = n^{\log_4 3}$;
- $f(n) = n \log n = \Omega(n^{\log_4 3+\varepsilon}) \approx \Omega(n^{0.793+\varepsilon})$, choose $\varepsilon = 0.2$
- Check addition condition $af(n/b) \leq rf(n)$ holds for all $n$.
  - Test $f(n) = n \log n \Rightarrow af(n/b) = 3(n/4)\log(n/4) \leq rn \log n$ holds for some $r < 1$.
  - Choose $r = 3/4 < 1$, this inequality holds for all $n$.

Master theorem (case 3) $\Rightarrow T(n) = \Theta(f(n)) = \Theta(n \log n)$

Binary search. $T(n) = T(n/2) + 1$, $T(1) = 1$

- $a = 1$, $b = 2$, $h(n) = n^{\log_2 1} = 1$, $f(n) = 1$

Master theorem (case 2) $\Rightarrow T(n) = \Theta(\log n)$

---

Merge sort. $T(n) = 2T(n/2) + (n - 1)$, $T(1) = 0$

- $a = 2$, $b = 2$, $h(n) = n^{\log_2 2} = n$, $f(n) = n - 1$

Master theorem (case 2) $\Rightarrow T(n) = \Theta(n \log n)$

# Cases that Master Theorem is not Applicable

Example. Compute the general term formula of

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

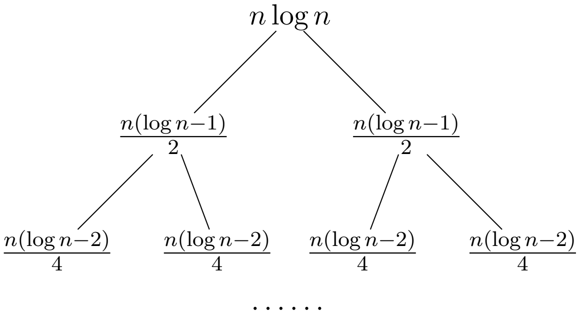
Apply master theorem. $a = b = 2$, $h(n) = n^{\log_b a} = n$, $f(n) = n \log n$

---

Only case 3 is possible, but $\nexists\ r < 1$ to make $af(n/b) \leq rf(n)$ holds for all $n$.

$$\begin{aligned} af(n/b) - rf(n) \quad &= 2(n/2)\log(n/2) - rn \log n \\ &= n(\log n - 1) - rn \log n \\ &= (1-r)n \log n - n > 0 \text{ if } r < 1 \end{aligned}$$

# Solving via Recursion Tree




$$n \log n$$

$$n(\log n - 1)$$

$$n(\log n - 2)$$

$$n(\log n - k + 1)$$

# Summation

$$\begin{aligned} T(n) &= n\log n + n(\log n - 1) + n(\log n - 2) \\ &\quad + \cdots + n(\log n - k + 1) \\ &= (n\log n)\log n - n(1 + 2 + \cdots + k - 1) \\ &= n\log^2 n - nk(k-1)/2 \qquad \text{//substitute with } k = \log n \\ &= \Theta(n\log^2 n) \end{aligned}$$

## Summary

Classical sequences and series

Complexity analysis: solving recurrence relation

- Direct Iteration
- Simplification-then-Iteration
- Recursion Tree

Master theorem and its proof

Application of Master Theorem