Recurrent Algorithms: Divide-and-Conquer Principle

Lecturer: Georgy Gimel'farb

COMPSCI 220 Algorithms and Data Structures

1. Divide-and-Conquer principle in analysing algorithms
2. Finding the close-form expression by math induction
3. Finding a close-form recurrence with a telescoping series
4. Examples
5. Algorithm analysis: Capabilities and limitations

Divide-and-Conquer Principle

- **Divide** a large problem into smaller subproblems;
- **Recursively solve** each subproblem, then
- **Combine solutions** of them to solve the original problem.

Running time: by a **recurrence relation** accounting for:

1. The size and the number of the subproblems and
2. The cost of splitting the problem into these subproblems.

The recursive relation $F(n) = \psi\left(F\left(n'_1\right), \ldots, F\left(n'_k\right)\right)$; $k \geq 1$, defines a function, $F(n)$, "in terms of itself", i.e., by involving the same function.

- The non-circular definition: $n > n'_1 > n'_2 > \ldots > n'_k$.
- The recursion terminates at some base case $F(n_0)$, below which the function is undefined.

Divide-and-Conquer Principle

- **Divide** a large problem into smaller subproblems;
- **Recursively solve** each subproblem, then
- **Combine solutions** of them to solve the original problem.

Running time: by a **recurrence relation** accounting for:

1. The size and the number of the subproblems and
2. The cost of splitting the problem into these subproblems.

The recursive relation $F(n) = \psi(F(n'_1), \ldots, F(n'_k))$; $k \geq 1$, defines a function, $F(n)$, "in terms of itself", i.e., by involving the same function.

- The non-circular definition: $n > n'_1 > n'_2 > \ldots > n'_k$.
- The recursion terminates at some base case $F(n_0)$, below which the function is undefined.

Divide-and-Conquer Principle

- **Divide** a large problem into smaller subproblems;
- **Recursively solve** each subproblem, then
- **Combine solutions** of them to solve the original problem.

Running time: by a **recurrence relation** accounting for:

1. The size and the number of the subproblems and
2. The cost of splitting the problem into these subproblems.

The recursive relation $F(n) = \psi\left(F\left(n'_1\right), \ldots, F\left(n'_k\right)\right)$; $k \geq 1$, defines a function, $F(n)$, "in terms of itself", i.e., by involving the same function.

- The non-circular definition: $n > n'_1 > n'_2 > \ldots > n'_k$.
- The recursion terminates at some base case $F(n_0)$, below which the function is undefined.

Recurrence Relation: A Simple Example $F(n) = 2^n$

The **implicit formula**: $F(n) = \underbrace{F(n-1)}_{2^{n-1}} + \underbrace{F(n-1)}_{2^{n-1}}$; $F(0) = 1$,

or

$$F(n) = 2F(n-1); \quad F(0) = 1; \quad n = 1, 2, \ldots$$

n	0	1	2	3	4	5	6	7	8	9	…
$F(n)$	1	2	4	8	16	32	64	128	256	512	…
	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	…

The **explicit**, or **closed-form formula** with $F(0) = 1$: $F(n) = 2^n$

Guess and Prove an Explicit, or Closed-Form $F(n)$

Look at a sequence of results for the implicit recurrent formula:

n	0	1	2	3	4	5	6	7	8	9	...
$F(n)$	1	2	4	8	16	32	64	128	256	512	...

Guess the closed-form formula $F(n) = 2^n$ and prove it with

Mathematical Induction

- **Basis**: $F(0) = 2^0 = 1$
- **Induction hypothesis**: $F(n) = 2^n$ holds some $n \geq 1$.
- **Inductive step** from n to $n+1$:
  $F(n+1) = F(n) + F(n) = 2F(n) = 2 \cdot 2^n = 2^{n+1}$.

This proves the close-form formula having been guessed.

Mathematical Induction (recall Lecture 1)

The induction examines conditions for a closed-form expression, $T(n)$, guessed, rather than derives it and proves directly.

1. **Basis**: $T(n_{\text{base}})$, e.g. $T(0)$ or $T(1)$, holds.
2. **Induction hypothesis**:
   Let $T(n)$ hold for some $n \geq n_{\text{base}}$
   or

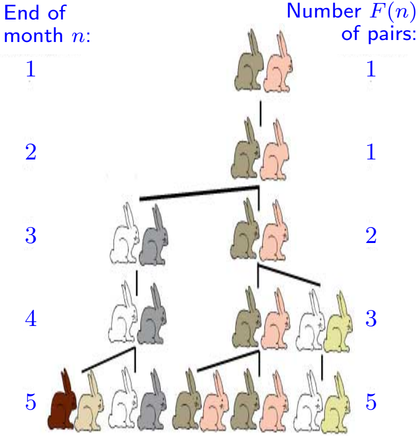
2′. **Strong induction hypothesis**:
   Let $T(k)$ hold for every $k = n_{\text{base}}, n_{\text{base}} + 1, \ldots, n$; $n \geq n_{\text{base}}$.

3 **Induction step**: Then $T(n+1)$ holds for $n+1$.

Both the simple and strong induction are actively used to solve recurrences, which are often met in the algorithm analysis.

Example 1.25: Fibonacci Numbers

https://timwolversonphotos.wordpress.com/category/composition/



Italian mathematician, **Leonardo Fibonacci** [1170–1250]: *"Liber Abaci"* – a problem of breeding rabbits:

- A pair of rabbits takes a month to become mature and start to have pairs of baby rabbits, which also take a month to reach maturity.
- How many rabbits, $F(n)$ would there be after n months?
- The Fibonacci Sequence: $F(n) = F(n-1) + F(n-2)$; $n \geq 3$; $F(1) = F(2) = 1$.

Example 1.25: Fibonacci Numbers

$$1, \quad 1, \quad 2, \quad 3, \quad 5, \quad 8, \quad 13, \quad 21, \quad 34, \quad 55, \quad 89, \quad 144, \quad \ldots$$

$$\begin{aligned}
1 + 1 &= 2 \\
1 + 2 &= 3 \\
2 + 3 &= 5 \\
3 + 5 &= 8 \\
&\ldots\ldots\ldots \\
55 + 89 &= 144 \\
&\ldots\ldots\ldots
\end{aligned}$$

The **implicit formula**: $F(n) = F(n-1) + F(n-2)$

The **recurrence analysis**: Derive a **closed-form** formula for $F(n)$

Characteristic Equation for $F(n) = F(n-1) + F(n-2)$

Because $F(n) > F(n-1) > F(n-2)$ for all $n \geq 2$, it holds that:

$$2F(n-1) > F(n) > 2F(n-2), \text{ that is, } 2^n > F(n) > 2^{n-1}$$

- One may suggest that $F(n) = c\varphi^n$; $1 < \varphi < 2$.
- The implicit equation $c\varphi^n = c\varphi^{n-1} + c\varphi^{n-2}$ leads to the quadratic characteristic equation for φ: $\varphi^2 = \varphi + 1$ – with two solutions: $\varphi_{1,2} = \frac{1}{2}\left(1 \pm \sqrt{5}\right)$.

General solution: the linear combination $F(n) = c_1\varphi_1^n + c_2\varphi_2^n$

- The coefficients c_1 and c_2 follow from the conditions $F(1) = F(2) = 1$, so that finally:

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

Characteristic Equation for $F(n) = F(n-1) + F(n-2)$

Because $F(n) > F(n-1) > F(n-2)$ for all $n \geq 2$, it holds that:

$$2F(n-1) > F(n) > 2F(n-2), \text{ that is, } 2^n > F(n) > 2^{n-1}$$

- One may suggest that $F(n) = c\varphi^n$; $1 < \varphi < 2$.
- The implicit equation $c\varphi^n = c\varphi^{n-1} + c\varphi^{n-2}$ leads to the quadratic characteristic equation for φ: $\varphi^2 = \varphi + 1$ – with two solutions: $\varphi_{1,2} = \frac{1}{2}\left(1 \pm \sqrt{5}\right)$.

General solution: the linear combination $F(n) = c_1\varphi_1^n + c_2\varphi_2^n$

- The coefficients c_1 and c_2 follow from the conditions $F(1) = F(2) = 1$, so that finally:

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

Characteristic Equation for $F(n) = F(n-1) + F(n-2)$

Because $F(n) > F(n-1) > F(n-2)$ for all $n \geq 2$, it holds that:

$$2F(n-1) > F(n) > 2F(n-2), \text{ that is, } 2^n > F(n) > 2^{n-1}$$

- One may suggest that $F(n) = c\varphi^n$; $1 < \varphi < 2$.
- The implicit equation $c\varphi^n = c\varphi^{n-1} + c\varphi^{n-2}$ leads to the quadratic characteristic equation for φ: $\varphi^2 = \varphi + 1$ – with two solutions: $\varphi_{1,2} = \frac{1}{2}\left(1 \pm \sqrt{5}\right)$.

General solution: the linear combination $F(n) = c_1\varphi_1^n + c_2\varphi_2^n$

- The coefficients c_1 and c_2 follow from the conditions $F(1) = F(2) = 1$, so that finally:

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

"Telescoping" a Recurrence

Given: An **implicit recurrence relation** and its **base condition** (i.e., the difference equation and its initial condition), for example:

$$T(n) = 2T(n-1) + 1; \quad T(0) = 0$$

Find: The closed-form (explicit) formula for $T(n)$ by recursive substitution of the same implicit formula:

$$\begin{array}{rcrcr} T(n) & = & 2T(n-1) & + & 1 \\ T(n-1) & = & 2T(n-2) & + & 1 \\ & & \ldots & & \\ T(2) & = & 2T(1) & + & 1 \\ T(1) & = & 2T(0) & + & 1 = 1 \end{array}$$

"Telescoping" $\equiv$ Substitution

$$
\begin{array}{rclcll}
T(n) & = & 2T(n-1) & + & 1 & \text{Step 0: Initial recurrence} \\
2T(n-1) & = & 2^2T(n-2) & + & 2 & \text{Step 1: Substitute } T(n-1) \\
2^2T(n-2) & = & 2^3T(n-3) & + & 2^2 & \text{Step 2: Substitute } T(n-2) \\
 & & \ldots & & & \\
2^{n-1}T(1) & = & 2^nT(0) & + & 2^{n-1} & \text{Step } n-1\text{: Substitute } T(1)
\end{array}
$$

$$T(n) = \underbrace{2^nT(0)}_{2^n\cdot 0=0} + 1 + 2 + 2^2 + \ldots + 2^{n-1}$$

$$1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$$

"Telescoping" $\equiv$ Substitution

$$T(n) = 2T(n-1) + 1 \quad \text{Step 0: Initial recurrence}$$

$$2T(n-1) = 2^2T(n-2) + 2 \quad \text{Step 1: Substitute } T(n{-}1)$$

$$2^2T(n-2) = 2^3T(n-3) + 2^2 \quad \text{Step 2: Substitute } T(n{-}2)$$

$$\ldots$$

$$2^{n-1}T(1) = 2^nT(0) + 2^{n-1} \quad \text{Step } n{-}1\text{: Substitute } T(1)$$

$$T(n) = \underbrace{2^nT(0)}_{2^n\cdot 0=0} + 1 + 2 + 2^2 + \ldots + 2^{n-1}$$

$$1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$$

"Telescoping" $\equiv$ Substitution

$$
\begin{array}{rcllll}
T(n) & = & 2T(n-1) & + & 1 & \text{Step } 0\text{: Initial recurrence} \\
2T(n-1) & = & 2^2T(n-2) & + & 2 & \text{Step } 1\text{: Substitute } T(n-1) \\
2^2T(n-2) & = & 2^3T(n-3) & + & 2^2 & \text{Step } 2\text{: Substitute } T(n-2) \\
 & & \ldots & & & \\
2^{n-1}T(1) & = & 2^nT(0) & + & 2^{n-1} & \text{Step } n-1\text{: Substitute } T(1)
\end{array}
$$

$$T(n) = \underbrace{2^nT(0)}_{2^n\cdot 0=0} + 1 + 2 + 2^2 + \ldots + 2^{n-1}$$

$$1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$$

"Telescoping" $\equiv$ Substitution

$T(n)$	$=$	$2T(n-1)$	$+$	1		Step 0: Initial recurrence
$2T(n-1)$	$=$	$2^2T(n-2)$	$+$	2		Step 1: Substitute $T(n-1)$
$2^2T(n-2)$	$=$	$2^3T(n-3)$	$+$	2^2		Step 2: Substitute $T(n-2)$
		$\dots$				
$2^{n-1}T(1)$	$=$	$2^nT(0)$	$+$	2^{n-1}		Step $n-1$: Substitute $T(1)$

$$T(n) = \underbrace{2^nT(0)}_{2^n\cdot 0=0} + 1 + 2 + 2^2 + \dots + 2^{n-1}$$

$$1 + 2 + 2^2 + \dots + 2^{n-1} = 2^n - 1$$

"Telescoping" $\equiv$ Substitution

$T(n)$	$=$	$2T(n-1)$	$+$	1	Step 0: Initial recurrence
$2T(n-1)$	$=$	$2^2T(n-2)$	$+$	2	Step 1: Substitute $T(n-1)$
$2^2T(n-2)$	$=$	$2^3T(n-3)$	$+$	2^2	Step 2: Substitute $T(n-2)$
		$\ldots$			
$2^{n-1}T(1)$	$=$	$2^nT(0)$	$+$	2^{n-1}	Step $n-1$: Substitute $T(1)$

$$T(n) = \underbrace{2^nT(0)}_{2^n\cdot 0=0} + 1 + 2 + 2^2 + \ldots + 2^{n-1}$$

$$1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$$

"Telescoping" $\equiv$ Substitution

$T(n)$	$=$	$2T(n-1)$	$+$	1	Step 0: Initial recurrence
$2T(n-1)$	$=$	$2^2T(n-2)$	$+$	2	Step 1: Substitute $T(n-1)$
$2^2T(n-2)$	$=$	$2^3T(n-3)$	$+$	2^2	Step 2: Substitute $T(n-2)$
		$\ldots$			
$2^{n-1}T(1)$	$=$	$2^nT(0)$	$+$	2^{n-1}	Step $n-1$: Substitute $T(1)$

$$T(n) = \underbrace{2^nT(0)}_{2^n \cdot 0 = 0} + 1 + 2 + 2^2 + \ldots + 2^{n-1}$$

$$1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$$

Example 1.29: Textbook, p.23

Show that the recurrence $T(n) = T(n-1) + n$; $T(0) = 0$,

results in the closed-form (explicit) formula $T(n) = \frac{n(n+1)}{2}$.

"Telescoping" the recurrence:

$$\begin{array}{rcrcr} T(n) & = & T(n-1) & + & n \\ T(n-1) & = & T(n-2) & + & n-1 \\ & & & & \cdots \\ T(2) & = & T(1) & + & 2 \\ T(1) & = & T(0) & + & 1 = 1 \end{array}$$

Example 1.29: Textbook, p.23

Show that the recurrence $T(n) = T(n-1) + n$; $T(0) = 0$,

results in the closed-form (explicit) formula $T(n) = \frac{n(n+1)}{2}$.

"Telescoping" the recurrence:

$$
\begin{array}{rcrcr}
T(n) & = & T(n-1) & + & n \\
T(n-1) & = & T(n-2) & + & n-1 \\
 & & & & \dots \\
T(2) & = & T(1) & + & 2 \\
T(1) & = & T(0) & + & 1 = 1
\end{array}
$$

1.29: $T(n)$ by Telescoping in More Detail

Successive substitution:

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= \overline{T(n-2) + (n-1)} + n \\
&= \overline{T(n-3) + (n-2)} + (n-1) + n \\
&\qquad \ldots \\
&= \overline{T(2) + 3} + \ldots + (n-2) + (n-1) + n \\
&= \overline{T(1) + 2} + 3 + \ldots + (n-2) + (n-1) + n \\
&= \overline{1} + 2 + 3 + \ldots + (n-2) + (n-1) + n = \frac{n(n+1)}{2}
\end{aligned}
$$

1.29: $T(n)$ by Guessing and Proving by Math Induction

Numerical sequence: $T(1) = 0 + 1 = 1$; $T(2) = 1 + 2 = 3$; $T(3) = 3 + 3 = 6$; $T(4) = 6 + 4 = 10$; $T(5) = 10 + 5 = 15$; ...

Guessing: $T(n) = \frac{n(n+1)}{2}$?

Base condition holds: $T(1) = \frac{1 \cdot 2}{2} = 1$.

Induction hypothesis: If the guessed formula $T(n)$ holds for $n-1$, then it holds also for n.

The proof: $T(n) = T(n-1) + n = \frac{(n-1)n}{2} + n$, i.e.

$$T(n) = \frac{1}{2}\left(n^2 - n + 2n\right) = \frac{1}{2}\left(n^2 + n\right) = \frac{n(n+1)}{2}$$

Thus, the guessed formula for $T(n)$ holds for all $n \geq 1$.

Example 1.30, p.23

Repeated halving principle: halve the input in one step

- Recurrence (implicit formula): $T(n) = T\left(\frac{n}{2}\right) + 1$; $T(1) = 0$.
- Closed-form (explicit) formula: $T(n) \approx \log_2 n$

"Telescoping" (for $n = 2^m$):

$$
\begin{aligned}
T(2^m) &= T(2^{m-1}) + 1 \\
T(2^{m-1}) &= T(2^{m-2}) + 1 \\
&\cdots \\
T(2^2) &= T(2^1) + 1 \\
T(2^1) &= T(2^0) + 1 = 1
\end{aligned}
$$

Example 1.30, p.23

Repeated halving principle: halve the input in one step

- Recurrence (implicit formula): $T(n) = T\left(\frac{n}{2}\right) + 1$; $T(1) = 0$.
- Closed-form (explicit) formula: $T(n) \approx \log_2 n$

"Telescoping" (for $n = 2^m$):

$$\begin{array}{rcrcr}
T(2^m) & = & T(2^{m-1}) & + & 1 \\
T(2^{m-1}) & = & T(2^{m-2}) & + & 1 \\
 & & & & \cdots \\
T(2^2) & = & T(2^1) & + & 1 \\
T(2^1) & = & T(2^0) & + & 1 = 1
\end{array}$$

1.30: $T(n)$ by Telescoping in More Detail

$$
\begin{aligned}
T(2^m) &= T(2^{m-1}) + 1 \\
&= \overline{T(2^{m-2}) + 1} + 1 \\
&= \overline{T(2^{m-3}) + 1} + 1 + 1 \\
&\qquad\qquad \ldots \\
&= \overline{T(2^1) + 1} + \ldots + 1 + 1 + 1 \\
&= \overline{T(2^0) + 1} + 1 + \ldots + 1 + 1 + 1 \\
&= \overline{1} + 1 + \ldots + 1 + 1 + 1 = m, \text{ or } T(2^m) = m
\end{aligned}
$$

- For $n = 2^m$, $T(n) = \lg n$, which is $\Theta(\log n)$.
- For general n, the total number of halving steps cannot be greater than $m = \lceil \lg n \rceil$, so $T(n) \leq \lceil \lg n \rceil$ for all n.

Example 1.31, p.23

Scan and halve the input:

- Recurrence (implicit formula): $T(n) = T\left(\frac{n}{2}\right) + n$; $T(1) = 1$.
- Closed-form (explicit) formula: $T(n) \approx 2n$

"Telescoping" (for $n = 2^m$):

$$
\begin{aligned}
T(2^m) &= T(2^{m-1}) + 2^m \\
T(2^{m-1}) &= T(2^{m-2}) + 2^{m-1} \\
&\quad \cdots \\
T(2^2) &= T(2^1) + 2^2 \\
T(2^1) &= T(2^0) + 2^1 \\
T(2^0) &= 2^0 = 1
\end{aligned}
$$

Example 1.31, p.23

Scan and halve the input:

- Recurrence (implicit formula): $T(n) = T\left(\frac{n}{2}\right) + n$; $T(1) = 1$.
- Closed-form (explicit) formula: $T(n) \approx 2n$

"Telescoping" (for $n = 2^m$):

$$\begin{array}{rcrcr} T(2^m) & = & T(2^{m-1}) & + & 2^m \\ T(2^{m-1}) & = & T(2^{m-2}) & + & 2^{m-1} \\ & & & & \cdots \\ T(2^2) & = & T(2^1) & + & 2^2 \\ T(2^1) & = & T(2^0) & + & 2^1 \\ T(2^0) & = & 2^0 = 1 & & \end{array}$$

1.31: $T(n)$ by Telescoping in More Detail

$$
\begin{aligned}
T(2^m) &= T(2^{m-1}) + 2^m \\
&= \overline{T(2^{m-2}) + 2^{m-1}} + 2^m \\
&= \overline{T(2^{m-3}) + 2^{m-2}} + 2^{m-1} + 2^m \\
&\qquad \cdots \\
&= \overline{T(2^1) + 2^2} + \ldots + 2^{m-2} + 2^{m-1} + 2^m \\
&= \overline{T(2^0) + 2^1} + 2^2 + \ldots + 2^{m-2} + 2^{m-1} + 2^m \\
&= \overline{1} + 2 + \ldots + 2^{m-2} + 2^{m-1} + 2^m = 2^{m+1} - 1
\end{aligned}
$$

Therefore, $T(2^m) \approx 2 \cdot 2^m$, or $T(n) \approx 2n$.

Example 1.32, p.23

"Divide-and-conquer" prototype; $n \geq 2$:

- Recurrence (implicit formula): $T(n) = 2T\left(\frac{n}{2}\right) + n$; $T(1) = 0$.
- Closed-form (explicit) formula: $T(n) \approx n \log_2 n$

Equivalent representation for "telescoping":

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad \Rightarrow \quad \frac{1}{n}T(n) = \frac{2}{n}T\left(\frac{n}{2}\right) + 1$$

$$\Rightarrow \quad \boxed{\frac{T(n)}{n} = \frac{T\left(\frac{n}{2}\right)}{\frac{n}{2}} + 1}$$

For $n = 2^m$, $\frac{T(2^m)}{2^m} = \frac{T(2^{m-1})}{2^{m-1}} + 1$

1.32: $T(n)$ by Telescoping in More Detail

$$
\begin{aligned}
\frac{T(2^m)}{2^m} &= \frac{T(2^{m-1})}{2^{m-1}} + 1 \\
&= \overline{\frac{T(2^{m-2})}{2^{m-2}} + 1} + 1 \\
&= \overline{\frac{T(2^{m-3})}{2^{m-3}} + 1} + 1 + 1 \\
& \qquad \cdots \\
&= \overline{\frac{T(2^1)}{2^1}} + 1 + \ldots + 1 + 1 + 1 \\
&= \overline{\frac{T(2^0)}{2^0}} + 1 + 1 + \ldots + 1 + 1 + 1 \\
&= \overline{0} + 1 + \ldots + 1 + 1 + 1 = m
\end{aligned}
$$

Therefore, $T(2^m) = m \cdot 2^m$, or $T(n) \approx n \lg n$.

Capabilities and Limitations

Rough time complexity analysis cannot result immediately in an efficient program.

- But it helps to predict empirical running time of the program.

Limitations of the "Big-Oh / Theta / Omega" analysis:

- It hides the constants (e.g. c and n_0) crucial for a practical task.
- It is unsuitable for small input.
- It is unsuitable if costs of access to input data items vary.
- It is unsuitable if there is lack of sufficient memory.

However,time complexity analysis provides ideas how to develop new and efficient algorithms.