


Recurrent Algorithms

Divide-and-conquer principle:

- Divide a large problem into smaller subproblems and recursively solve each subproblem, then
- Combine solutions of the subproblems to solve the original problem

Running time: by a **recurrence relation** combining the size and number of the subproblems and the cost of dividing the problem into the subproblems


Lecture 5 COMPSCI 220 - AP G. Gimel'farb 1



Simple Example: $F(n) = 2^n$

- **Implicit formula:**
 $F(n) = F(n - 1) + F(n - 1); F(0) = 1,$
 or $F(n) = 2F(n - 1); F(0) = 1; n = 1, 2, \dots$
- **1, 2, 4, 8, 16, 32, 64, 128, 256, 512, ...** → $2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, \dots$
- **Explicit formula** with $F(0) = 1$: $F(n) = 2^n$

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 2



Guess and Prove an Explicit $F(n)$


Guess $F(n) = 2^n$ and prove with math induction:

- **Base inductive condition:** $F(0) = 2^0 = 1$
- **Induction hypothesis:**
 for any $n \geq 1$, if $F(n-1) = 2^{n-1}$ holds for $n-1$, then $F(n) = 2^n$ holds for n

– **Proof:**

$$F(n) = 2F(n-1) = 2 \cdot 2^{n-1} = 2^n$$

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 3




Mathematical Induction

Instead of directly proving a relationship $T(n)$, math induction proves the following conditions:

1. **Base condition** : $T(n_{\text{base}})$, e.g., $T(0)$ or $T(1)$
2. **Induction hypothesis**: for every $n > n_{\text{base}}$ if $T(n-1)$ holds for $n-1$, then $T(n)$ holds for n ; or
2. **Strong induction**: if $T(k)$ holds for every $k = n_{\text{base}}, \dots, n-1$, then $T(n)$ holds for n


Lecture 5 COMPSCI 220 - AP G. Gimel'farb 4



Example 1.29: Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...
- $1 + 1 = 2$
- $1 + 2 = 3$
- $2 + 3 = 5$
- $3 + 5 = 8 \dots$
- **Implicit formula:** $F(n) = F(n - 1) + F(n - 2)$
- **Analysis:** derive an **explicit formula** for $F(n)$

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 5



Characteristic Equation

- $F(n) > F(n - 1)$ for all $n \geq 1 \Rightarrow 2^{n-1} < F(n) < 2^n \Rightarrow F(n) = c\phi^n; 1 < \phi < 2 \Rightarrow c\phi^n = c\phi^{n-1} + c\phi^{n-2} \Rightarrow \phi^2 = \phi + 1 \Rightarrow \phi_{1,2} = (1 \pm \sqrt{5})/2$
- **General solution: linear combination**
 $F(n) = c_1\phi_1^n + c_2\phi_2^n$
- $F(1)=F(2)=1 \Rightarrow F(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 6

“Telescoping” of a Recurrence

- Implicit recurrence relation and its base condition (i.e., the difference equation and the initial condition):

$$T(n) = 2T(n-1) + 1; T(0) = 0$$
- Closed (explicit) form for $T(n)$ by recursive substitution of the same implicit formula:

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

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 7

“Telescoping” ≡ Substitution

$$\begin{aligned} T(n) &= 2T(n-1) + 1 && \text{step 0 : initial recurrence} \\ 2T(n-1) &= 2^2T(n-2) + 2 && \text{step 1 : substitute } T(n-1) \\ 2^2T(n-2) &= 2^3T(n-3) + 2^2 && \text{step 2 : substitute } T(n-2) \\ &\dots \\ 2^{n-1}T(1) &= 2^nT(0) + 2^{n-1} && \text{step } n-1 : \text{substitute } T(1) \end{aligned}$$

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

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 8

Example 1.30, p.27

$$T(n) = T(n-1) + n \Leftrightarrow T(n) = \frac{n(n+1)}{2}$$

Closed (explicit) formula by “telescoping”:

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

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 9

Ex.1.30: $T(n)$ by Telescoping

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

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 10

Ex.1.30: $T(n)$ by Guessing and Proving by Math Induction

- $T(1) = 1; T(2) = 1 + 2 = 3; T(3) = 3 + 3 = 6;$
 $T(4) = 6 + 4 = 10 \Rightarrow$ **Guessing:** $T(n) = \frac{n(n+1)}{2}$
- Base condition** holds: $T(1) = 1 \cdot 2 / 2 = 1$
- Induction hypothesis:** if the guessed formula $T(n)$ holds for $n - 1$, then it holds also for n
- Proof:** $T(n) = T(n-1) + n = \frac{(n-1)n}{2} + n = \frac{n(n+1)}{2}$
- Thus, the guessed formula for $T(n)$ holds for all $n > 1$

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 11


Example 1.31, p.27

- Repeated halving principle:** halve the input in one step:

$$T(n) = T(n/2) + 1 \Leftrightarrow T(n) \cong \log_2 n$$
- “Telescoping” (for $n = 2^m$):

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


Lecture 5 COMPSCI 220 - AP G. Gimel'farb 12



Ex.1.31: Explicit Formula for $T(n)$

$$\begin{aligned}
 T(2^m) &= T(2^{m-1}) + 1 \\
 &= \overline{T(2^{m-2}) + 1} + 1 \\
 &\dots = \overline{T(2^1) + 1 + 1 + \dots + 1} \\
 &= \overline{T(2^0) + 1 + 1 + \dots + 1 + 1} \\
 T(2^m) = m &\Rightarrow T(n) = \log_2 n
 \end{aligned}$$

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 13




Example 1.32, p.27

- Scan and halve the input: $T(n) = T(n/2) + n \Leftrightarrow T(n) \cong 2n$
- “Telescoping” (for $n = 2^m$):

$T(2^m) = T(2^{m-1}) + 2^m$	$T(2^2) = T(2^1) + 2^2$
$T(2^{m-1}) = T(2^{m-2}) + 2^{m-1}$	$T(2^1) = T(2^0) + 2^1$
\dots	$T(2^0) = 1$


Lecture 5 COMPSCI 220 - AP G. Gimel'farb 14



Ex.1.32: Explicit Formula for $T(n)$

$$\begin{aligned}
 T(2^m) &= T(2^{m-1}) + 2^m \\
 &= \overline{T(2^{m-2}) + 2^{m-1}} + 2^m \\
 &\dots = \overline{T(2^1) + 2^2 + \dots + 2^{m-1}} + 2^m \\
 &= \overline{T(2^0) + 2^1 + 2^2 + \dots + 2^{m-1}} + 2^m \\
 T(2^m) = 2^{m+1} - 1 &\Rightarrow T(n) \cong 2n
 \end{aligned}$$


Lecture 5 COMPSCI 220 - AP G. Gimel'farb 15



Example 1.33, p.28

- “Divide-and-conquer” prototype; $n \geq 2$: $T(n) = 2T(n/2) + n \Leftrightarrow T(n) \cong n \log_2 n$
- “Telescoping”: $\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1; T(1) = 0$
- For $n = 2^m \rightarrow \frac{T(2^m)}{2^m} = \frac{T(2^{m-1})}{2^{m-1}} + 1$


Lecture 5 COMPSCI 220 - AP G. Gimel'farb 16



Ex.1.33: Explicit Formula for $T(n)$

$$\begin{aligned}
 T(2^m)/2^m &= T(2^{m-1})/2^{m-1} + 1 \\
 &= \overline{T(2^{m-2})/2^{m-2} + 1} + 1 \\
 &\dots = \overline{T(2^1)/2^1 + 1 + \dots + 1} + 1 \\
 &= \overline{T(2^0)/2^0 + 1 + 1 + \dots + 1} + 1 \\
 &= 0 + 1 + \dots + 1 = m \\
 T(2^m) = m \cdot 2^m &\Rightarrow T(n) = n \log_2 n
 \end{aligned}$$

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 17



Capabilities and Limitations

- Rough complexity analysis cannot result immediately in an efficient program but it helps to predict empirical running time of the program
- Limitations of “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
- But complexity analysis provides ideas how to develop new efficient methods

Lecture 5 COMPSCI 220 - AP G. Gimel'farb 18