Recurrent Algorithms

Divide-and-conquer principle:
to divide a large problem into smaller ones and recursively solve each subproblem, then
to 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

“Telescoping” a Recurrence

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

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

Closed (explicit) form for $T(n)$ by “telescoping”:

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

“Telescoping” ≡ Substitution

$$
\begin{aligned}
\mathrm{T}(n) &= \cancel{2\,\mathrm{T}(n-1)} + 1 \\
\cancel{2\,\mathrm{T}(n-1)} &= \cancel{2^2\,\mathrm{T}(n-2)} + 2 \\
\cancel{2^2\,\mathrm{T}(n-2)} &= \cancel{2^3\,\mathrm{T}(n-3)} + 2^2 \\
&\ldots \\
\cancel{2^{n-1}\,\mathrm{T}(2)} &= 2^n\,\mathrm{T}(1) + 2^{n-1}
\end{aligned}
$$

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

Basic Recurrence: 1

Closed (explicit) form by "telescoping":

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

$$\begin{aligned}
\mathrm{T}(n) &= \mathrm{T}(n-1) + n \\
\mathrm{T}(n-1) &= \mathrm{T}(n-2) + n - 1 \\
&\cdots \\
\mathrm{T}(2) &= \mathrm{T}(1) + 2 \\
\mathrm{T}(1) &= 1
\end{aligned} \Bigg\downarrow$$

1: Explicit Expression for $\mathrm{T}(n)$

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

Guessing to Solve a Recurrence

Infer (guess) a hypothetic solution $T(n)$; $n \geq 0$ from a sequence of numbers $T(0), T(1), T(2), \ldots,$ obtained from the recurrence relation

Prove $T(n)$ by math induction:

Base condition: T holds for $n = n_{\text{base}}$,e.g. $T(0)$ or $T(1)$

Induction hypothesis to verify: for every $n > n_{\text{base}}$, if T holds for $n - 1$, then T holds for n

Strong induction: if T holds for $n_{\text{base}}, \ldots, n - 1$, then…

Explicit Expression for $\mathrm{T}(n)$

$\mathrm{T}(1) = 1$; $\mathrm{T}(2) = 1 + 2 = 3$; $\mathrm{T}(3) = 3 + 3 = 6$;

$\mathrm{T}(4) = 6 + 4 = 10 \Rightarrow$ Hypothesis:

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

Base condition holds: $\mathrm{T}(1) = 1\cdot 2 / 2 = 1$

If the hypothetic closed-form relationship $\mathrm{T}(n)$ holds for $n - 1$ then it holds also for n:

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

Thus, the expression for $\mathrm{T}(n)$ holds for all $n > 1$

Basic Recurrence: 2

Repeated halving principle: halve the input in one step:

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

$$T(n) = T(n/2) + 1 \iff T(n) \cong \log_2 n$$

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

2: Explicit Expression for $\mathrm{T}(n)$

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

Basic Recurrence: 3

Scan and halve the input:

"Telescoping" (for $n = 2^m$): $T(n) = T(n/2) + n \quad \Leftrightarrow \quad T(n) \cong 2n$

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

3: Explicit Expression for $\mathrm{T}(n)$

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

$$\mathrm{T}(2^m) = 2^{m+1} - 1 \;\Rightarrow\; \mathrm{T}(n) \cong 2n$$

Basic Recurrence: 4

“Divide-and-conquer” prototype:

$$T(n) = 2\,T(n/2) + n \quad \Leftrightarrow \quad T(n) \cong n \log_2 n$$

“Telescoping”:

$$\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1; \quad T(1) = 0$$

For

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

4: Explicit Expression for $\mathrm{T}(n)$

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

$$\mathrm{T}(2^m) = m \cdot 2^m \quad \Rightarrow \quad \mathrm{T}(n) = n \log_2 n$$

General “Divide-and-Conquer”

Theorem: The recurrence $T(n) = a\,T(n/b) + cn^k;\ T(1) = c$

with integer constants $a \geq 1$ and $b \geq 2$ and positive constants c and k has the solution:

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{if} \quad b^k < a \\ O(n^k \log n) & \text{if} \quad b^k = a \\ O(n^k) & \text{if} \quad b^k > a \end{cases}$$

Proof by telescoping: $n = b^m \Rightarrow T(b^m) = a\,T(b^{m-1}) + cb^{mk}$

General “Divide-and-Conquer”

Telescoping:

$$
\begin{aligned}
T(b^m) &= a\cancel{T(b^{m-1})} + cb^{mk} \\
\cancel{aT(b^{m-1})} &= a^2\cancel{T(b^{m-2})} + acb^{(m-1)k} \\
\cancel{a^2T(b^{m-2})} &= a^3\cancel{T(b^{m-3})} + acb^{(m-2)k} \\
\ldots &= \ldots \\
a^{m-1}\cancel{T(b)} &= a^mT(1) + a^{m-1}cb^k \\
\hline
\mathrm{T}(b^m) &= a^m c + a^{m-1}cb^k + \ldots + acb^{(m-1)k} + cb^{mk}
\end{aligned}
$$

$$
= c\sum_{t=0}^{m} a^{m-t}b^{tk} = ca^m\sum_{t=0}^{m}\left({}^{b^k}\!/_{a}\right)^t
$$

Recursion Example

```
long factorial( int n )
{
   if( n <= 1 )
     return 1;
   else
     return n*factorial(n- 1);
}
```

In terms of big-Oh:

$t(1) = 1$

$t(n) = 1 + t(n\text{-}1) = 1 + 1 + t(n\text{-}2)$

$\quad = \ldots k + t(n\text{-}k)$

Choose $k = n\text{-}1$

$t(n) = n\text{-}1 + t(1) = n\text{-}1 + 1 = O(n)$

Consider the following time complexity:

$t(0) = 1$

$t(n) = 1 + 2t(n\text{-}1) = 1 + 2(1 + 2t(n\text{-}2)) = 1 + 2 + 4t(n\text{-}2)$

$\quad = 1 + 2 + 4(1 + 2t(n\text{-}3)) = 1 + 2 + 4 + 8t(n\text{-}3)$

$\quad = 1 + 2 + \ldots + 2^{k-1} + 2^k t(n\text{-}k)$

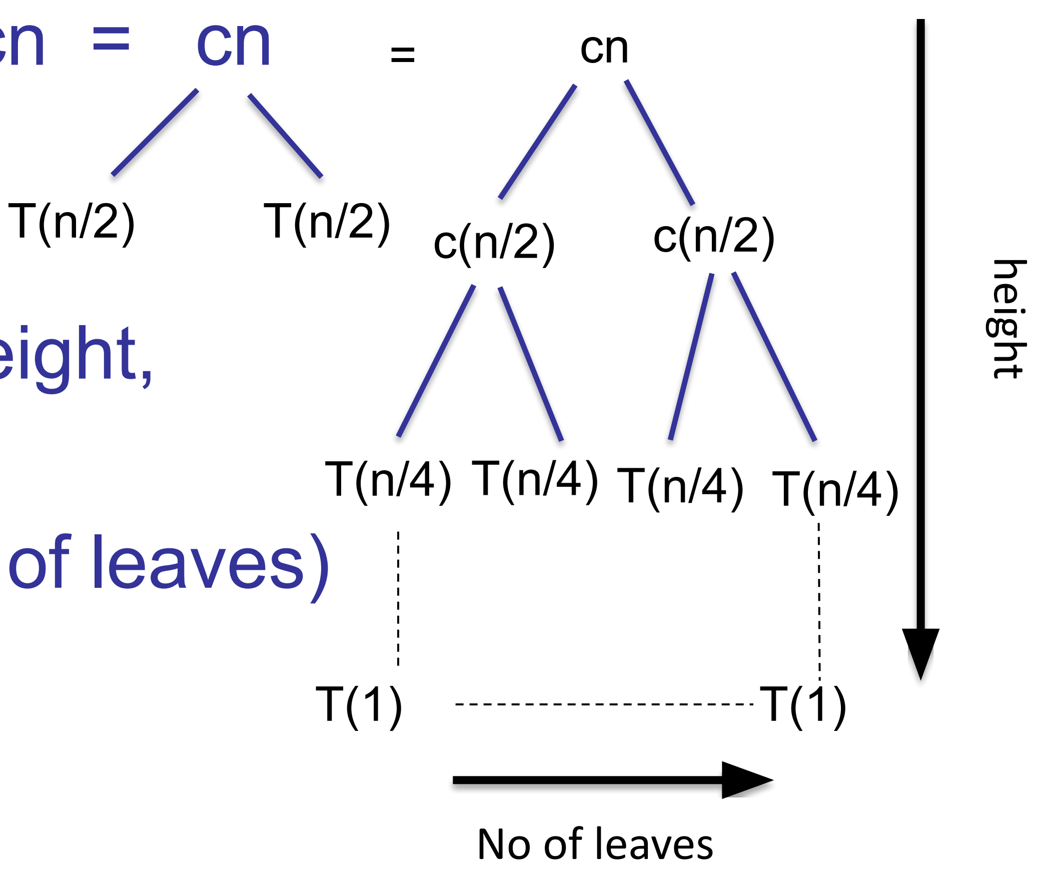
Choose $k = n$

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

Another Approach (Recursive Tree Method)

- $T(n) = 2T(n/2) + O(n)$, $T(1) = O(1)$

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



$T(n/2^k) = T(1)$ where k= height,

$K= \lg n$

$O(n) = O(\text{height} * \text{number of leaves})$

$= O(k*n)=O(n\lg n)$

Another Example

- $T(n) = T(2n/3) + T(n/3) + O(n)$; $T(1) = 1$
- Two Sub-problems
- Different Sub-problem size $(2n/3)$ and $(n/3)$
- Two different heights k_1 ,k_2 for $(2n/3)$ and $(n/3)$.
- $k_1 = \log_{3/2}n$, $k_2 = \log_3 n$
- Number of leaves is n
- Complexity ??
  - $O(\# \text{ of leaves} * \max(k_1,k_2)) \approx O(n*\max(\log_{3/2}n, \log_3 n)) \approx O(n\log_{3/2}n) \approx O(n\log_2 n)$

Capabilities and Limitations

Rough complexity analysis cannot result immediately in an efficient practical program but it helps in predicting of empirical running time of the program

“Big-Oh” analysis is unsuitable for small input and hides the constants c and n_0 crucial for a practical task

“Big-Oh” analysis is unsuitable if costs of access to input data items vary and if there is lack of sufficient memory

But complexity analysis provides ideas how to develop new efficient methods