Worst-Case Performance

- **Upper bounds**: simple to obtain
- **Lower bounds**: ***a difficult matter...***
- Worst case data may be unlikely to be met in practice
- Unknown "Big-Oh" constants c and n_0 may not be small
- Inputs in practice lead to much lower running times
- ***Example:*** the most popular fast sorting algorithm, **QuickSort**, has $O(n^2)$ running time in the worst case but in practice the time is $O(n \log n)$

Average-Case Performance

- Estimate average time for each operation
- Estimate frequencies of operations
- May be a difficult challenge... (take **COMPSCI.320** for details)
- May be no natural "average" input at all
- May be hard to estimate (average time for each operation depends on data)

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":

$$T(n) = 2\,T(n-1) + 1$$
$$T(n-1) = 2\,T(n-2) + 1$$
$$\ldots$$
$$T(2) = 2\,T(1) + 1$$

"Telescoping" ≡ Substitution

$$T(n) = 2\,T(n-1) + 1$$
$$2\,T(n-1) = 2^2\,T(n-2) + 2$$
$$2^2\,T(n-2) = 2^3\,T(n-3) + 2^2$$
$$\ldots$$
$$2^{n-1}\,T(2) = 2^n\,T(1) + 2^{n-1}$$

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

Basic Recurrence: 1

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

- Closed (explicit) form by "telescoping":

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

1: Explicit Expression for $T(n)$

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

Lecture 5 COMPSCI.220.FS.T - 2004 7

Guessing to Solve a Recurrence

- Infer (guess) a hypothetic solution $T(n)$; $n \geq 0$ from a sequence of numbers $T(0)$, $T(1)$, $T(2)$, …, obtained from the recurrence relation
- Prove $T(n)$ by math induction:

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

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

Strong induction: if T holds for n_{base}, …, n - 1, then...

Lecture 5 COMPSCI.220.FS.T - 2004 8

Explicit Expression for $T(n)$

- $T(1) = 1$; $T(2) = 1 + 2 = 3$; $T(3) = 3 + 3 = 6$; $T(4) = 6 + 4 = 10 \Rightarrow$ Hypothesis: $T(n) = \frac{n(n+1)}{2}$
- Base condition holds: $T(1) = 1 \cdot 2 / 2 = 1$
- If the hypothetic closed-form relationship $T(n)$ holds for $n - 1$ then it holds also for n:

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

- Thus, the expression for $T(n)$ holds for all $n > 1$

Lecture 5 COMPSCI.220.FS.T - 2004 9

Basic Recurrence: 2

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

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

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

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

Lecture 5 COMPSCI.220.FS.T - 2004 10

2: Explicit Expression for $T(n)$

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

Lecture 5 COMPSCI.220.FS.T - 2004 11

Basic Recurrence: 3

- Scan and halve the input:

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

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

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

Lecture 5 COMPSCI.220.FS.T - 2004 12

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

$$\mathrm{T}(2^m) = \mathrm{T}(2^{m-1}) + 2^m$$
$$= \overline{\mathrm{T}(2^{m-2}) + 2^{m-1}} + 2^m$$
$$\ldots = \overline{\mathrm{T}(2^1) + 2^2} + \ldots + 2^{m-1} + 2^m$$
$$= \overline{\mathrm{T}(2^0) + 2^1} + 2^2 + \ldots + 2^{m-1} + 2^m$$
$$\mathrm{T}(2^m) = 2^{m+1} - 1 \;\Rightarrow\; \mathrm{T}(n) \cong 2n$$

Basic Recurrence: 4

- **"Divide-and-conquer" prototype:**

$$\mathrm{T}(n) = 2\,\mathrm{T}(n/2) + n \;\Leftrightarrow\; \mathrm{T}(n) \cong n \log_2 n$$

- "Telescoping": $\dfrac{\mathrm{T}(n)}{n} = \dfrac{\mathrm{T}(n/2)}{n/2} + 1; \quad \mathrm{T}(1) = 0$

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

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

$$\mathrm{T}(2^m)/2^m = \mathrm{T}(2^{m-1})/2^{m-1} + 1$$
$$= \overline{\mathrm{T}(2^{m-2})/2^{m-2} + 1} + 1$$
$$\ldots = \overline{\mathrm{T}(2^1)/2^1 + 1} + \ldots + 1 + 1$$
$$= \overline{\mathrm{T}(2^0)/2^0 + 1} + 1 + \ldots + 1 + 1$$
$$= 0 + 1 + \ldots + 1 = m$$
$$\mathrm{T}(2^m) = m \cdot 2^m \;\Rightarrow\; \mathrm{T}(n) = n \log_2 n$$

General "Divide-and-Conquer"

Theorem: The recurrence $\mathrm{T}(n) = a\,\mathrm{T}(n/b) + cn^k;\ \mathrm{T}(1) = c$ with integer constants $a \geq 1$ and $b \geq 2$ and positive constants c and k has the solution:

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

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

General "Divide-and-Conquer"

- Telescoping: $T(b^m) = aT(b^{m-1}) + cb^{mk}$

$$aT(b^{m-1}) = a^2T(b^{m-2}) + acb^{(m-1)k}$$
$$a^2T(b^{m-2}) = a^3T(b^{m-3}) + acb^{(m-2)k}$$
$$\ldots \quad = \quad \ldots$$
$$a^{m-1}T(b) = a^mT(1) + a^{m-1}cb^k$$
$$\mathrm{T}(b^m) = a^mc + a^{m-1}cb^k + \ldots + acb^{(m-1)k} + cb^{mk}$$
$$= c\sum_{t=0}^{m} a^{m-t}b^{tk} = ca^m \sum_{t=0}^{m} \left(b^k/a\right)^t$$

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