CMPS 3120

Algorithm Analysis

Dr. Chengwei Lei
CEECS
California State University, Bakersfield
Example 1: Maximum element

**Algorithm**  \textit{MaxElement}(A[0..n - 1])

//Determines the value of the largest element in a given array
//Input: An array A[0..n - 1] of real numbers
//Output: The value of the largest element in A

maxval \leftarrow A[0]

\textbf{for} i \leftarrow 1 \textbf{to} n - 1 \textbf{do}

\quad \textbf{if} A[i] > maxval

\qquad maxval \leftarrow A[i]

\textbf{return} maxval
Example 2: Element uniqueness problem

**ALGORITHM**  
*UniqueElements*(\(A[0..n − 1]\))

// Determines whether all the elements in a given array are distinct
// Input: An array \(A[0..n − 1]\)
// Output: Returns “true” if all the elements in \(A\) are distinct
// and “false” otherwise

**for** \(i \leftarrow 0\) **to** \(n − 2\) **do**

**for** \(j \leftarrow i + 1\) **to** \(n − 1\) **do**

\[\mbox{if } A[i] = A[j]\]

**return** true
Example 3: Matrix multiplication

\begin{algorithm}
\textbf{MatrixMultiplication}(A[0..n−1, 0..n−1], B[0..n−1, 0..n−1])
\begin{algorithmic}
\STATE //Multiplies two n-by-n matrices by the definition-based algorithm
\STATE //Input: Two n-by-n matrices A and B
\STATE //Output: Matrix C = AB
\FOR {i ← 0 \text{ to } n−1}
\FOR {j ← 0 \text{ to } n−1}
\STATE \text{C}[i, j] ← 0.0
\FOR {k ← 0 \text{ to } n−1}
\STATE \text{C}[i, j] ← \text{C}[i, j] + \text{A}[i, k] \ast \text{B}[k, j]
\ENDFOR
\ENDFOR
\STATE \text{return C}
\end{algorithmic}
\end{algorithm}
Example 4: Gaussian elimination

Algorithm GaussianElimination(A[0..n-1,0..n])

//Implements Gaussian elimination of an n-by-(n+1) matrix A

for i ← 0 to n - 2 do
    for j ← i + 1 to n - 1 do
        for k ← i to n do

Find the efficiency class and a constant factor improvement.
Example 5: Counting binary digits

**ALGORITHM**  \( \text{Binary}(n) \)

//Input: A positive decimal integer \( n \)
//Output: The number of binary digits in \( n \)'s binary representation
\[
\text{count} \leftarrow 1
\]
\[
\textbf{while } n > 1 \textbf{ do}
\]
\[
\text{count} \leftarrow \text{count} + 1
\]
\[
n \leftarrow \lfloor n/2 \rfloor
\]
\[
\textbf{return } \text{count}
\]

It cannot be investigated the way the previous examples are.
Plan for Analysis of Recursive Algorithms

- Decide on a parameter indicating an input’s size.
- Identify the algorithm’s basic operation.
- Check whether the number of times the basic op. is executed may vary on different inputs of the same size. (If it may, the worst, average, and best cases must be investigated separately.)
- Set up a recurrence relation with an appropriate initial condition expressing the number of times the basic op. is executed.
- Solve the recurrence (or, at the very least, establish its solution’s order of growth) by backward substitutions or another method.
Arithmetic series

- An arithmetic series is a sequence of numbers such that the difference of any two successive members of the sequence is a constant.

  e.g.: 1, 2, 3, 4, 5
  or 10, 12, 14, 16, 18, 20

- In general:

  \[ a_i = a_{i-1} + d \]

  Or: \[ a_i = a_1 + (i - 1)d \]
Sum of arithmetic series

If \( a_1, a_2, \ldots, a_n \) is an arithmetic series, then

\[
\sum_{i=1}^{n} a_i = \frac{n(a_1 + a_n)}{2}
\]

e.g. \( 1 + 3 + 5 + 7 + \ldots + 99 = ? \)

(series definition: \( a_i = 2i-1 \))
This is \( \sum_{i = 1 \text{ to } 50} (a_i) \)
A geometric series is a sequence of numbers such that the ratio between any two successive members of the sequence is a constant.

e.g.: 1, 2, 4, 8, 16, 32
or 10, 20, 40, 80, 160
or 1, 1/2, 1/4, 1/8, 1/16

In general:

\[ a_i = r a_{i-1} \]  \hspace{1cm} \text{Recursive definition}

Or:

\[ a_i = r^i a_0 \]  \hspace{1cm} \text{Closed form, or explicit formula}
Sum of geometric series

\[
\sum_{i=0}^{n} r^i = \begin{cases} 
(1 - r^{n+1})/(1 - r) & \text{if } r < 1 \\
(r^{n+1} - 1)/(r - 1) & \text{if } r > 1 \\
n + 1 & \text{if } r = 1
\end{cases}
\]

\[
\sum_{i=0}^{n} 2^i = ?
\]

\[
\lim_{n \to \infty} \sum_{i=0}^{n} \frac{1}{2^i} = ?
\]

\[
\lim_{n \to \infty} \sum_{i=1}^{n} \frac{1}{2^i} = ?
\]
Sum of geometric series

\[ \sum_{i=0}^{n} r^i = \begin{cases} 
  (1 - r^{n+1})/(1 - r) & \text{if } r < 1 \\
  (r^{n+1} - 1)/(r - 1) & \text{if } r > 1 \\
  n + 1 & \text{if } r = 1 
\end{cases} \]

\[ \sum_{i=0}^{n} 2^i = \frac{2^{n+1} - 1}{2 - 1} = 2^{n+1} - 1 \approx 2^{n+1} \]

\[ \lim_{n \to \infty} \sum_{i=0}^{n} \frac{1}{2^i} = \lim_{n \to \infty} \sum_{i=0}^{n} (\frac{1}{2})^i = \frac{1}{1 - \frac{1}{2}} = 2 \]

\[ \lim_{n \to \infty} \sum_{i=1}^{n} \frac{1}{2^i} = \lim_{n \to \infty} \sum_{i=0}^{n} (\frac{1}{2})^0 - (\frac{1}{2})^0 = 2 - 1 = 1 \]
Important formulas

\[ \sum_{i=1}^{n} 1 = n \in \Theta(n) \]

\[ \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \in \Theta(n^2) \]

\[ \sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1} \begin{cases} \Theta(1) & (r < 1) \\ \Theta(r^n) & (r > 1) \end{cases} \]

\[ \sum_{i=1}^{n} i^2 \approx \frac{n^3}{3} \in \Theta(n^3) \]

\[ \sum_{i=1}^{n} i^k \approx \frac{n^{k+1}}{k + 1} \in \Theta(n^{k+1}) \]

\[ \sum_{i=1}^{n} i 2^i = (n - 1)2^{n+1} + 2 \in \Theta(2^n) \]

\[ \sum_{i=1}^{n} \frac{1}{i} \in \Theta(\lg n) \]

\[ \sum_{i=1}^{n} \lg i \in \Theta(n \lg n) \]
Sum manipulation rules

\[ \sum_i (a_i + b_i) = \sum_i a_i + \sum_i b_i \]

\[ \sum_i c a_i = c \sum_i a_i \]

\[ \sum_{i=m}^{n} a_i = \sum_{i=m}^{x} a_i + \sum_{i=x+1}^{n} a_i \]

Example:

\[ \sum_{i=1}^{n} (4i + 2^i) = ? \]

\[ \sum_{i=1}^{n} \frac{n}{2^i} = ? \]
Sum manipulation rules

\[ \sum_i (a_i + b_i) = \sum_i a_i + \sum_i b_i \]
\[ \sum_i c a_i = c \sum_i a_i \]
\[ \sum_{i=m}^{n} a_i = \sum_{i=m}^{x} a_i + \sum_{i=x+1}^{n} a_i \]

Example:

\[ \sum_{i=1}^{n} (4i + 2^i) = 4 \sum_{i=1}^{n} i + \sum_{i=1}^{n} 2^i = 2n(n+1) + 2^{n+1} - 2 \]
\[ \sum_{i=1}^{n} \frac{n}{2^i} = n \sum_{i=1}^{n} \frac{1}{2^i} \approx n \]
Examples

- \( \sum_{i=1}^{n} \frac{n}{2^i} = n \cdot \sum_{i=1}^{n} \left(\frac{1}{2}\right)^i = ? \)

- using the formula for geometric series:
  \( \sum_{i=0}^{n} \left(\frac{1}{2}\right)^i = 1 + \frac{1}{2} + \frac{1}{4} + \ldots \left(\frac{1}{2}\right)^n = 2 \)

- Application: algorithm for allocating dynamic memories
Examples

- $\sum_{i=1}^{n} \log(i) = \log 1 + \log 2 + \ldots + \log n$
  
  $= \log 1 \times 2 \times 3 \times \ldots \times n$

  $= \log n!$

  $= \Theta(n \log n)$

- Application: algorithm for selection sort using priority queue
Recursive definition of sum of series

- $T(n) = \sum_{i=0}^{n} i$ is equivalent to:
  - $T(n) = T(n-1) + n$
  - $T(0) = 0$

- $T(n) = \sum_{i=0}^{n} a^i$ is equivalent to:
  - $T(n) = T(n-1) + a^n$
  - $T(0) = 1$

Recursive definition is often intuitive and easy to obtain. It is very useful in analyzing recursive algorithms, and some non-recursive algorithms too.
Example 1: Recursive evaluation of $n!$

**Definition:**

$n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$ for $n \geq 1$ and $0! = 1$

Recursive definition of $n!$: $F(n) = F(n-1) \cdot n$ for $n \geq 1$ and $F(0) = 1$

**Algorithm**

```
F(n)

// Computes n! recursively
// Input: A nonnegative integer n
// Output: The value of n!
if n = 0 return 1
else return F(n - 1) * n
```

**Size:**

**Basic operation:**

**Recurrence relation:**
Solving the recurrence for $M(n)$

\[ M(n) = M(n-1) + 1, \quad M(0) = 0 \]