Analysis of Algorithms

Hwansoo Han
Running Time

- Algorithm of choice
  - Easy to understand, code, and debug
  - Efficient use of computer resources => runs fast

- Running time depends on input size
  - \( T(n) \), where \( n \) is the size of input

- Worst case vs. average running times
Big-Oh and Big-Omega

- Growth rate
  - Trend of running time, T(n)

- Upper bound, $O(f(n))$
  - $T(n) \leq cf(n), \ n \geq n_0$ (c and $n_0$ are constants)

- Lower bound, $\Omega(g(n))$
  - $T(n) \geq cg(n)$ for an infinite number of values of $n$

- Examples
  - $T(n) = (n+1)^2 : T(n) \leq 4n^2, \ n \geq 1 \Rightarrow T(n) \ is \ O(n^2)$
  - $T(n) = n$ (odd $n \geq 1$) : $T(n) \geq n^2/100, \ n = 0, 2, 4, \ldots$
    - $n^2/100$ (even $n \geq 0$) \Rightarrow $T(n) \ is \ \Omega(n^2)$
Tyranny of Growth Rate

- Which one is better?
  - $O(n)$ vs. $O(n^2)$ vs. $O(n^3)$
Tyranny of Growth Rate

Which one is better?
- O(n) vs. O(n^2) vs. O(n^3)

Answer:
- Depends on INPUT SIZE

For small sizes: O(n^2) or O(n^3) is better
For large sizes: O(n) is better

As computers become more powerful, we desire to solve larger and more complex problems
- Low growth rate algorithms need to be discovered!
Practical Algorithms

- Sometimes, the upper bound of growth rate is not important
  - Used only a few times
  - Run on “small” inputs only
    - Constant factors in running time, $T(n)$, are more important!
    - e.g.) $n^2$ vs. $1000000n$, $n^2$ vs. $n + 10000000000000000$
  - Maintaining issue
  - Too much space
    - Many I/Os in and out of disk slow down the implementation
  - Accuracy and stability in numerical algorithms
Calculating the Running Time

- Rule for sums: execute $P_1$ and $P_2$ sequentially
  - $T_1(n)$, $T_2(n)$: running times of $P_1$ and $P_2$
  - $T_1(n)$ is $O(f(n))$, $T_2(n)$ is $O(g(n))$
  - $T_{1+2}(n)$ is $O(\max(f(n), g(n)))$

- Rule for products: execute $P_2$ within $P_1$
  - e.g.) $P2$ is the body of $P1$ loop
  - $T_1(n)$, $T_2(n)$: running times of $P_1$ and $P_2$
  - $T_1(n)$ is $O(f(n))$, $T_2(n)$ is $O(g(n))$
  - $T_{12}(n)$ is $O(f(n)g(n))$
Example: Bubble Sort

```plaintext
procedure bubble ( var A: array [1..n] of integer );
    { bubble sorts array A into increasing order }

var
    i, j, temp: integer;

begin

(1)    for i := 1 to n-1 do
(2)        for j := n downto i+1 do
                { swap A[j - 1] and A[j] }
(4)                temp := A[j-1];
            end
end; { bubble }
```

Example: Bubble Sort

1. (4),(5),(6)
2. (3)-(6)
3. (2)-(6)
4. (1)-(6)

\[ i, j, \text{temp}: \text{integer}; \]
\begin{align*}
\text{begin} & \\
(1) & \text{for } i := 1 \text{ to } n-1 \text{ do} \\
(2) & \quad \text{for } j := n \text{ downto } i+1 \text{ do} \\
(3) & \quad \text{if } A[j-1] > A[j] \text{ then begin} \\
(4) & \qquad \{ \text{swap } A[j-1] \text{ and } A[j] \} \\
(5) & \qquad \text{temp } := A[j-1]; \\
(6) & \qquad A[j-1] := A[j]; \\
(7) & \qquad A[j] := \text{temp}; \\
\text{end} & \\
\text{end; } \{ \text{bubble} \} \\
\end{align*}
Running Time for Procedures

- Non-recursive
  - Count the running time of whole body

- Recursive

function fact ( n: integer ): integer;
{ fact(n) computes n! } begin
  (1) if n <= 1 then
      (2) fact := 1
  else
      (3) fact := n * fact(n-1)
end; { fact }
Recursive Algorithms

- $T(n) = T(n/2) + d$

```plaintext
function R2(a: integer,  s, n: integer): integer
var
    mid: integer;
begin
    mid = (s+n)/2;
    if (A[mid] = a)
        return mid;
    else if (A[mid] < a)
        R2(a, mid+1, n);
    else
        R2(a, s, mid-1);
end
```