2-2 The Efficiency of Algorithms

Hengfeng Wei

hfwei@nju.edu.cn

March 05, 2020
The Analysis of Algorithms
Donald E. Knuth (1938 ~)
Donald E. Knuth (1974)

“For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the “art of computer programming” through his well-known books in a continuous series by this title.”
Fibonacci numbers in the analysis of Euclid’s GCD algorithm $H_n$ in the analysis of FIND-MAX @ Stanford Lecture by Knuth

“People who analyze algorithms have double happiness.

First of all they experience the sheer beauty of elegant mathematical patterns that surround elegant computational procedures.

Then they receive a practical payoff when their theories make it possible to get other jobs done more quickly and more economically.”
How Fast is It?

Time (and Space) Complexity of Algorithms

\[ \Theta, \Omega, O, o, \omega \]
Space Complexity of Algorithms

We only care about the extra space caused by the algorithm. The space for inputs is not part of space complexity of algorithms.

\texttt{INSERTION-SORT}(A, n) : \textit{O}(1) \quad (\text{constant})
Is it the Fastest?

Complexity of Problems

This is much harder and is not our focus today.
Whenever you design an algorithm, you provide an upper bound for the complexity of the problem.

Whenever you encounter a “hardcore” of the problem, you obtain a lower bound for all possible algorithms.

Often, there is an “algorithmic gap” between them.

When the gap is gone, you get the optimal algorithm.

\[
\text{sorting}(A, n) : \Theta(n \log n) = O(n \log n) \cap \Omega(n \log n)
\]
Q : How fast is your algorithm?

A : It runs 3.1415926 seconds.
Disadvantages:

- On different machines
- At different time
- On different inputs

No Standards.
We need a uniform model of computation.

The RAM (Random Access Machine) Model of Computation
The RAM (Random Access Machine) Model of Computation

- Each memory access takes constant time.
- Each "primitive" operation takes constant time.
- Compound operations should be decomposed.

Counting up the number of time units.
Disadvantages:

- On different machines
- At different time
- On different inputs

Counting up the number of time units as a function of the input size in typical cases.
\begin{algorithm}
\textbf{Insertion-Sort} \((A)\)
\begin{algorithmic}[1]
\State \textbf{for} \( j = 2 \) \textbf{to} \( \text{A.length} \)
\State \hspace{1em} \textit{key} = \text{A}[j]
\State \hspace{1em} // \text{Insert} \text{A}[j] \text{ into the sorted }
\State \hspace{1em} \text{sequence} \text{A}[1..j-1].
\State \hspace{1em} \textit{i} = j - 1
\State \hspace{1em} \textbf{while} \( i > 0 \) \text{ and} \text{A}[i] > \textit{key}
\State \hspace{1em} \text{A}[i + 1] = \text{A}[i]
\State \hspace{1em} \textit{i} = \textit{i} - 1
\State \text{A}[i + 1] = \textit{key}
\end{algorithmic}
\end{algorithm}

\[ T(n) = c_1 n + c_2 (n - 1) + c_4 (n - 1) \]

\[ + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n - 1) \]

\dots as a function of the input size \dots
**INSERTION-SORT** (*A*)

1. **for** *j* = 2 to *A*.length
2.  
3. // Insert *A*[j] into the sorted sequence *A*[1..*j* − 1].
4.  
5. *i* = *j* − 1
6. **while** *i* > 0 and *A*[i] > *key*
7.  
8. *A*[i + 1] = *A*[i]
9.  
10. *i* = *i* − 1
11. *A*[i + 1] = *key*

\[
T(n) = c_1 n + c_2 (n - 1) + c_4 (n - 1)
\]
\[
+ c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n - 1)
\]

\[
T(n): \text{Depends on which input of size } n
\]
Problem $P$  Algorithm $A$

Inputs: $\mathcal{X}_n$ of size $n$

\[
W(n) = \max_{x \in \mathcal{X}_n} T(x)
\]

\[
B(n) = \min_{x \in \mathcal{X}_n} T(x)
\]

\[
A(n) = \sum_{x \in \mathcal{X}_n} T(x) \cdot P(x) = \mathbb{E}[T] = \sum_{t \in T(\mathcal{X}_n)} t \cdot P(T = t)
\]

\[= \sum_{t \in T(\mathcal{X}_n)} t \cdot P(T = t)\]
**Insertion-Sort**($A$)

1. **for** $j = 2$ to $A.length$
2. \hspace{0.5em} $key = A[j]$
4. \hspace{0.5em} $i = j - 1$
5. **while** $i > 0$ and $A[i] > key$
6. \hspace{1em} $A[i + 1] = A[i]$
7. \hspace{1em} $i = i - 1$
8. \hspace{1em} $A[i + 1] = key$

\begin{align*}
B(n) &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \\
W(n) &= \frac{c_5 + c_6 + c_7}{2}n^2 + (c_1+c_2+c_4+c_8 - \frac{c_5 + c_6 + c_7}{2})n - (c_2+c_4+c_5+c_8) \\
A(n) &= 2.25n^2 + 7.75n - 3H_n - 6 \quad (H_n = \sum_{k=1}^{n} \frac{1}{k} \approx \ln n)
\end{align*}
Q: How fast is your algorithm?

listen carefully.

\[ W(n) = \frac{c_5 + c_6 + c_7}{2} n^2 + (c_1 + c_2 + c_4 + c_8 - \frac{c_5 + c_6 + c_7}{2}) n - (c_2 + c_4 + c_5 + c_8) \]
BIGOMICRON AND BIG OMEGA AND BIG THETA

Donald E. Knuth
Computer Science Department
Stanford University
Stanford, California 94305

Reference:
“Big Omicron and Big Omega and Big Theta”, Donald E. Knuth, 1976.

Asymptotics
Q: How fast is your algorithm?

\[
W(n) = \frac{c_5 + c_6 + c_7}{2} n^2 + (c_1 + c_2 + c_4 + c_8 - \frac{c_5 + c_6 + c_7}{2}) n - (c_2 + c_4 + c_5 + c_8)
\]

\[
W(n) = O(n^2)
\]

“Order at most \( n^2 \)”

“\( W(n) \) is a function whose order of magnitude is upper-bounded by a constant times \( n^2 \), for all large \( n \).”
\[ f(n) = O(g(n)) \]

"\( f(n) \) is a function whose order of magnitude is upper-bounded by a constant times \( g(n) \), for all large \( n \)."

\[ O(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \right\} \]
\( f(n) = O(g(n)) \)

\[
O(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \right\}
\]

It is a tradition to write \( f(n) = O(g(n)) \) instead of \( f(n) \in O(g(n)) \).
\[42n^2 + 2020n = O(n^2) = O(n^3)\]

\[42n^2 + 2020n \in O(n^2) \subseteq O(n^3)\]
\[ O(f(n)) + O(g(n)) \triangleq \left\{ h + l \mid h \in O(f(n)), l \in O(g(n)) \right\} \]

\[ O(f(n))O(g(n)) \triangleq \left\{ hl \mid h \in O(f(n)), l \in O(g(n)) \right\} \]

\[ O(f(n)) - O(g(n)) \triangleq \]
\[ O(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) \leq cg(n) \right\} \]

\[ 42n = O(0.50n^2) \quad 42n^2 = O(0.50n^2) \]

\[ Q : \text{What does } O(1) \text{ mean?} \]

\[ A : \text{It means constants.} \]
$$\Omega(g(n)) = \left\{ f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq cg(n) \leq f(n) \right\}$$

$$0.50n^2 = \Omega(42n) \quad 0.50n^2 = \Omega(42n^2)$$

$$\Theta(g(n)) = \left\{ f(n) \mid \exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \right\}$$

$$0.50n^2 = \Theta(42n^2)$$
\[ o(g(n)) = \left\{ f(n) \mid \forall c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq f(n) < cg(n) \right\} \]

\[ 42n = o(0.50n^2) \]

\[ \omega(g(n)) = \left\{ f(n) \mid \forall c > 0, \exists n_0 > 0, \forall n \geq n_0 : 0 \leq cg(n) < f(n) \right\} \]

\[ 0.50n^2 = \omega(42n) \]
\[ O \quad \Omega \quad \Theta \]

\[ o \quad \omega \quad \theta \]

\[ f(n) \sim g(n) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 1 \]

\[ 42n^2 + 2020n \sim 42n^2 + 2019n \]
\[ f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \land f(n) = \Omega(g(n)) \]

\[ f(n) = O(g(n)) \iff g(n) = \Omega(f(n)) \]

\[ f(n) = o(g(n)) \iff g(n) = \omega(f(n)) \]
\[ O(f(n)) + O(g(n)) = O(f(n) + g(n)) \]

\[ O(f(n))O(g(n)) = O(f(n)g(n)) \]
Q : How to compare functions in terms of $O/\Omega/\Theta$?

$$O(1) = O(\log \log n) = O(\log n) = O((\log n)^c) = O(n^\epsilon) = O(n^c) = O(n^c \log n) = O(n^{\log n}) = O(c^n) = O(n^n)$$

$$0 < \epsilon < 1 < c$$
Stirling Formula (by *James Stirling*):

\[ n! \sim \sqrt{2\pi n} \left( \frac{n}{e} \right)^n \]

\[ \log(n!) = \Theta(n \log n) \]

\[ H_n = \sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n) \]
Learning by Doing
\[ A[0, \ldots n - 1] \quad 1 \leq l \leq n \]

\textbf{ROTATE}(A, n, l) : Rotate A left by \( l \) places

\begin{align*}
0 & 1 & 2 & 3 & 4 \\
3 & 4 & 0 & 1 & 2
\end{align*}

\textbf{Critical Operation: copy}
1: **procedure** ROTATE($A$, $n$, $l$)
2: for $i=1 \ldots l$ do
3: ROTATE-BY-ONE($A$, $n$)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotate-one-by-one</td>
<td>$nl = O(n^2)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
1: **procedure** ROTATE($A, n, l$)
2:     copy $A[0\ldots l - 1]$ into $v$
3:     move $A[l\ldots n - 1]$ left $l$ places
4:     copy $v$ to $A[l\ldots n - 1]$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotate-copy</td>
<td>$O(n)$</td>
<td>$l = O(n)$</td>
</tr>
</tbody>
</table>
\( n = 5, \quad l = 3 \)

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
3 & 4 & 0 & 1 & 2 \\
\end{array}
\]

\((0, 2, 4, 1, 3)\)

\( n = 9, \quad l = 6 \)

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
6 & 7 & 8 & 0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\((0, 3, 6) \quad (1, 7, 4) \quad (2, 8, 5)\)
Correctness Proof?

Permutations as **Product of Disjoint Cycles**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotate-cyclic</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
\[ B \cdot A = (A^R \cdot B^R)^R \]

\[
\begin{array}{ccccc}
0 & 1 & 2 & 3 & 4 \\
2 & 1 & 0 & 4 & 3 \\
3 & 4 & 0 & 1 & 2 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotate-reverse</td>
<td>(O(n))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Time</td>
<td>Space</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>rotate-one-by-one</td>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>rotate-copy</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>rotate-cyclic</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>rotate-reverse</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Chapter 9: Asymptotics

\( O \quad \Omega \quad \Theta \quad o \quad \omega \)
Thank You!