Time complexity analysis

Let’s start with selection sort

C. Seshadhri

University of California, Santa Cruz
sesh@ucsc.edu

March 30, 2016
Sorting

Input

- An array e.g., [1, 5, 6, 7, 3, 2, 1]

Output

- Sorted array [1, 1, 2, 3, 5, 6, 7]
Formally

**Input**
- An array e.g., \([a_1, a_2, a_3, \ldots, a_n]\)

**Output**
- A permuted version of the array \([a'_1, a'_2, a'_3, \ldots, a'_n]\) such that \(a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_n\)
Sorting

Comes up everywhere

- Give me webpages on “Data Structures”, in decreasing order of relevance
- Give me a list of coffee shops, in decreasing order of ratings
- Give me a list of grocery stores, in increasing order of distance
Sorting with keys

**Input**
- An array of key-value pairs e.g.,
  
  $$[(k_1, v_1), (k_2, v_2), (k_3, v_3), \ldots, (k_n, v_n)]$$

**Output**
- A permuted version of the array
  
  $$[(k'_1, v'_1), (k'_2, v'_2), (k'_3, v'_3), \ldots, (k'_n, v'_n)]$$
  such that
  
  $$v'_1 \leq v'_2 \leq v'_3 \leq \ldots \leq v'_n$$
What is an Algorithm?

- A sequence of well defined computational steps that
  - takes some input
  - transforms it into an output (with a certain property)

Alternatively

- An algorithm describes a specific computational procedure for achieving a specified input/output relationship
Selection Sort

(Let length of array $A$ be $n$.)

**SELECTION-SORT(A)**

1. For $i = 1$ to $n$:
   1. Find minimum element $b$ in $A[i\ldots n]$.
   2. Set $b$ to be $i$th element in sorted order
Selection Sort

(Let length of array $A$ be $n$.)

**Selection-Sort**($A$)

1. **for** $i = 1$ **to** $n$
2. \hspace{1em} $min = A[i]$ \hspace{1em} // Initialize min
3. \hspace{1em} $minInd = i$ \hspace{1em} // Initialize min index
4. **for** $j = i$ **to** $n$
5. \hspace{3em} **if** $A[j] < min$ \hspace{1em} // Aha! Min needs to be changed
6. \hspace{5em} \hspace{1em} $min = A[j]$ \hspace{1em} // So change min and min index
7. \hspace{5em} \hspace{1em} $minInd = j$
8. \hspace{1em} // Swap $A[i]$ with $A[minInd]$
9. \hspace{1em} $A[minInd] = A[i]$
10. \hspace{1em} $A[i] = min$
Selection-Sort on an array of $n$ items makes at most how many comparisons? (Give the best answer.)

- (R) $n$
- (B) $n - 1$
- (G) $(n - 1)(n - 2)/2$
- (O) $n^2 - 2n$
How many comparisons?

Selection-Sort on an array of $n$ items makes at most how many comparisons? (Give the best answer.)

- (R) $n$
- (B) $n - 1$
- (G) $(n - 1)(n - 2)/2$
- (O) $n^2 - 2n$
How “long” does Selection-Sort take?

- It really depends on your machine!
- How can we mathematically talk about the “running time” of Selection-Sort?
Analyzing Selection-Sort

Definition

The *running time* of Selection-Sort (or your favorite algorithm) on input $A$ is the number of “elementary” operations that Selection-Sort takes on $A$.

- Beats wall clock time.
- Challenges:
  - What is *elementary*?
Analyzing Selection-Sort

**Definition**

The *running time* of Selection-Sort (or your favorite algorithm) on input $A$ is the number of “elementary” operations that Selection-Sort takes on $A$.

- Beats wall clock time.
- Challenges:
  - What is *elementary*?
  - Ok, for sorting, any assignment or comparison counts as one operation.
  - Running time is not one number; it depends heavily on input (especially size).
Worst-case Analysis

Definition

The worst-case running time or worst-case time complexity of Selection-Sort (or your favorite algorithm) is the function $T(n)$ where:

$T(n)$ is the maximum running time over all inputs of size $n$. 
(Let length of array $A$ be $n$.)

**Selection-Sort($A$)**

1. for $i = 1$ to $n$
2.     $min = A[i]$  // Initialize min
3.     $minInd = i$  // Initialize min index
4.     for $j = i$ to $n$
5.         if $A[j] < min$  // Aha! Min needs to be changed
6.             $min = A[j]$  // So change min and min index
7.             $minInd = j$
8.     // Swap $A[i]$ with $A[minInd]$
10.    $A[i] = min$
Exactly finding $T(n)$

- It’s a real pain.
- Once you get the basic idea, it’s boring to figure out exactly.
The solution?
Selection Sort

The solution?
Focus on the asymptotic growth of $T(n)$

- Asymptotic: Fancy way of saying that we will focus on how the function grows as $n \to \infty$. 
Oh Notation

- Given time complexity $T(n)$ and “convenient/simple function” $f(n)$ we say that $T(n) \in O(f(n))$ if
  - there exists constants $c$ and $n_0$
  - such that

  $$0 \leq T(n) \leq c \cdot f(n) \text{ for all } n \geq n_0.$$  

- Notational convention: $T(n) = O(f(n))$, $T(n)$ is $O(f(n))$, $T(n)$ is in $O(f(n))$.
- We’re providing a convenient upper bound for $T(n)$. 
In Pictures

The diagram shows a graph with axes labeled $n$ and $n_0$. The graph includes two curves:

- $T(n)$, which starts at a certain point and increases as $n$ increases.
- $c \cdot g(n)$, which is a straight line that intersects $T(n)$ at $n_0$.

The graph suggests a comparison between the time complexity $T(n)$ and the function $c \cdot g(n)$, indicating the growth rates of the algorithms or processes being compared.
Is $2n + 30 = O(n)$?

- (R) Yes
- (B) No
Example - 1

- $2n + 30 = O(n)$
- Need to show that there exists $c$ and $n_0$ such that
  \[
  0 \leq 2n + 30 \leq c \cdot n \text{ for all } n \geq n_0
  \]
- Let $c = 3$ and $n_0 = 30$. 
Is $2n + 30 = O(n^2)$?

- (R) Yes
- (B) No
Example - 2

- $2n + 30 = O(n^2)$
- Need to show that there exists $c$ and $n_0$ such that
  \[ 0 \leq 2n + 30 \leq c \cdot n^2 \text{ for all } n \geq n_0 \]
- Let $c = 2$ and $n_0 = 6$. 
Is $5n^3 + 10n = O(n^2)$?

- (R) Yes
- (B) No
Example - 3
Example - 3

- Reductio ad absurdum
- $5n^3 + 10n \not\in O(n^2)$
Example - 3

- $5n^3 + 10n \notin O(n^2)$
- Proof by contradiction. Suppose $c$ and $n_0$ existed.

\[
0 \leq 5n^3 + 10n \leq c \cdot n^2 \text{ for all } n \geq n_0
\]

\[
0 \leq 5n + \frac{10}{n} \leq c \text{ for all } n \geq n_0
\]

But as $n \to \infty$ we have $5n + \frac{10}{n} \to \infty$. 

C. Seshadhri (UCSC)
Selection Sort

(Let length of array $A$ be $n$.)

\text{SELECTION-SORT}(A)

\begin{enumerate}
\item \textbf{for} $i = 1$ \textbf{to} $n$
\item $min = A[i]$ \quad // Initialize min
\item $minInd = i$ \quad // Initialize min index
\item \textbf{for} $j = i$ \textbf{to} $n$
\item \hspace{1em} \textbf{if} $A[j] < min$ \quad // Aha! Min needs to be changed
\item \hspace{2em} $min = A[j]$ \quad // So change min and min index
\item \hspace{2em} $minInd = j$
\item \hspace{1em} // Swap $A[i]$ with $A[minInd]$
\item $A[minInd] = A[i]$
\item $A[i] = min$
\end{enumerate}

Selection-Sort time complexity is $O(n^2)$. 
Omega Notation

- Given functions $T(n)$ and $f(n)$ we say that $T(n) \in \Omega(f(n))$ if
  - there exists constants $c$ and $n_0$
  - such that

  $$0 \leq cf(n) \leq T(n) \text{ for all } n \geq n_0.$$

- Notational convention: $T(n) = \Omega(f(n))$
Theta Notation

- Given functions $T(n)$ and $f(n)$ we say that $T(n) \in \Theta(f(n))$ if
  - there exists constants $c_1, c_2$ and $n_0$
  - such that

\[ 0 \leq c_1 f(n) \leq T(n) \leq c_2 \cdot f(n) \text{ for all } n \geq n_0. \]

- Notational convention: $T(n) = \Theta(f(n))$
An Alternate View Point

\[ T(n) = O(f(n)) \implies \lim_{n \to \infty} \frac{T(n)}{f(n)} \leq \text{const} \]

\[ T(n) = \Omega(f(n)) \implies \lim_{n \to \infty} \frac{T(n)}{f(n)} \geq \text{const} \]

\[ T(n) = \Theta(f(n)) \implies \text{const}_1 \leq \lim_{n \to \infty} \frac{T(n)}{f(n)} \leq \text{const}_2 \]
In Pictures

(a) $f(n) = \Theta(g(n))$

(b) $f(n) = O(g(n))$

(c) $f(n) = \Omega(g(n))$
Example - 4

- $5n^2 + 10n = \Theta(n^2)$
- Need to show that there exists $c_1$, $c_2$ and $n_0$ such that
  
  $0 \leq c_1 n^2 \leq 5n^2 + 10n \leq c_2 \cdot n^2$ for all $n \geq n_0$

- Rewrite as
  
  $0 \leq c_1 \leq 5 + \frac{10}{n} \leq c_2$ for all $n \geq n_0$

But as $n \to \infty$ we have $5 + \frac{10}{n} \to 5$. So select $c_1 = 4$ and $c_2 = 6$, with $n_0 = 10$. 
Prove

For any two functions \( f(n) \) and \( g(n) \), we have

\[
f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).
\]
The time complexity of Selection-Sort is:

- (R) $\Theta(n^2)$
- (B) $O(n^2)$ but not $\Theta(n^2)$.
- (G) $\Omega(n^2)$ but not $\Theta(n^2)$.
- (O) I have no idea what’s going on.
Selection Sort

(Let length of array \( A \) be \( n \).)

**Selection-Sort** \((A)\)

1. \( \text{for } i = 1 \text{ to } n \)
2. \( \text{min} = A[i] \)  // Initialize min
3. \( \text{minInd} = i \)  // Initialize min index
4. \( \text{for } j = i \text{ to } n \)
5. \( \text{if } A[j] < \text{min} \)  // Aha! Min needs to be changed
6. \( \text{min} = A[j] \)  // So change min and min index
7. \( \text{minInd} = j \)
8. \( \text{// Swap } A[i] \text{ with } A[\text{minInd}] \)
9. \( A[\text{minInd}] = A[i] \)
10. \( A[i] = \text{min} \)

Selection-Sort time complexity is \( \Theta(n^2) \).
Little $o$ and $\omega$ Notation

**$o$ Notation**

- $O$ notation is not asymptotically tight. For instance $n = O(n)$ and also $n = O(n^2)$.
- Given functions $g(n)$ and $f(n)$ we say that $f(n) \in o(g(n))$ if
  - there exists a constant $n_0$
  - such that

  $0 \leq f(n) \leq cg(n)$ for all $c > 0$ and $n \geq n_0$.

- Carefully note the order of the qualifiers.
- Intuitively, $f(n)$ becomes insignificant compared to $g(n)$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$
Example - 5

- $5n^2 + 10n \notin o(n^2)$
- $\lim_{n \to \infty} 5 + \frac{10}{n} = 5 \neq 0$
- $5n^2 + 10n = o(n^3)$
- $\lim_{n \to \infty} \frac{5}{n} + \frac{10}{n^2} = 0$
Given functions $g(n)$ and $f(n)$ we say that $f(n) \in \omega(g(n))$ if

- there exists a constant $n_0$
- such that

$$0 \leq cg(n) \leq f(n)$$

for all $c > 0$ and $n \geq n_0$.

Carefully note the order of the qualifiers.

Intuitively, $f(n)$ grows much faster as compared to $g(n)$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty.$$
Questions?