Complexity

Notes from Dr. Charlie Obimbo and Dr. Andrew Hamilton-Wright
Revised by Yan Yan.

Contents

Some Definitions

- An ALGORITHM is a well-defined computational procedure that transforms inputs into outputs, achieving a desired input-output relationship.
- A computational PROBLEM is a specification of the desired input-output relationship.
- An INSTANCE of a Problem is an example (or set of inputs and outputs that adhere to the computational specification of the problem.)
- A CORRECT algorithm halts with the correct output for every input instance.

Algorithm

- Example:
  - Sorting - a common operation.
  - Many sorting algorithms available.
  - Best choice depends on application.
- Problem

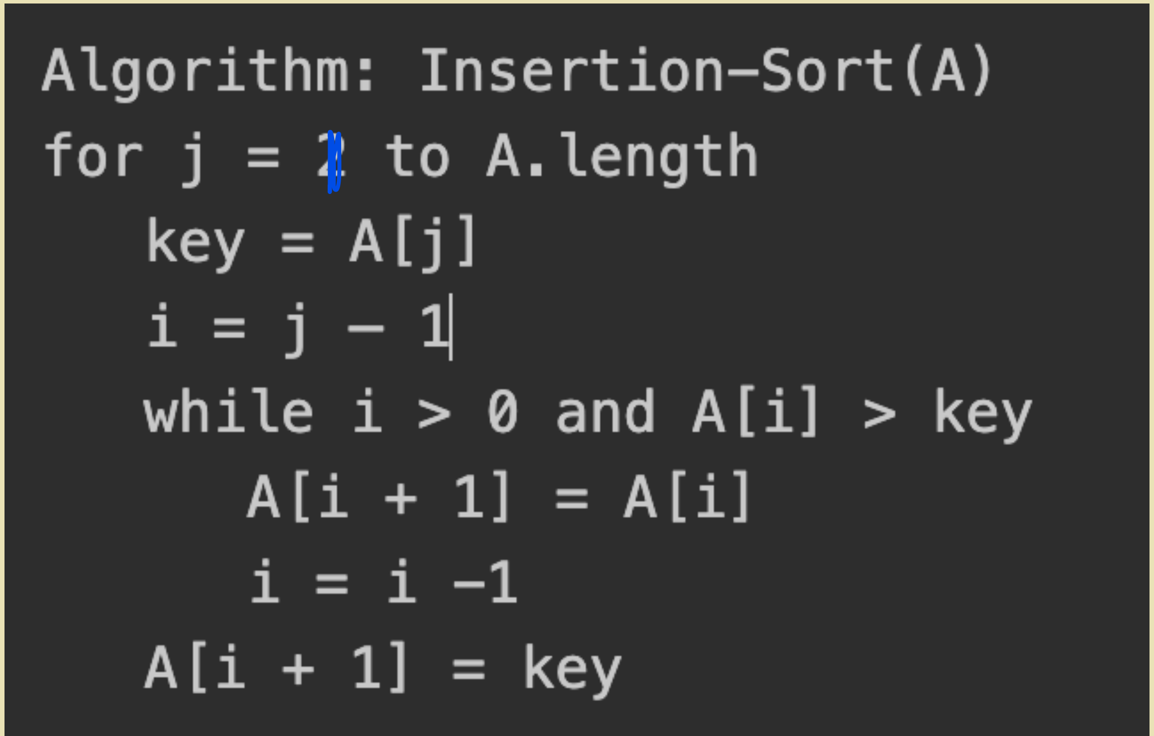
►INPUT: Sequence of n objects $(a_1, a_2, ..., a_n)$.

►OUTPUT: Permutation (reordering) $(a\ , a\ , ..., a\ )$ of the input sequence such that $a \le a \le ... \le a$, by a certain key.

- Instance
  - (7,4,3,1,5,4) → (1,3,4,4,5,7)

Insertion Sort Algorithm

- Uses only a fixed amount of storage that needed for the data:
- Pseudocode:



```
Algorithm: Insertion-Sort(A)
for j = 2 to A.length
    key = A[j]
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i -1
    A[i + 1] = key
```

Analyzing Algorithm

- Predict resource utilization
  - Memory
  - Running time
- Depend on architecture
  - Running Time could depend on
    - Problem Size
    - Input Size
    - Number of primitive operations used to solve the problem

Analyzing Algorithm

- Input Size:
  - Sorting: number of items
  - Graphs: number of vertices and edges
- Operations
  - Examples: additions, multiplications, comparisons
  - Constant time: C_i per ith line of pseudocode

Analyzing Algorithm -- Operations

- Best Case $B(n)$:
  - constraints on the input, other than size, resulting in the fastest possible running time.
- Worst Case $W(n)$:
  - constraints on the input, other than size, resulting in the slowest possible running time.
- Average Case $A(n)$:
  - average running time over every possible type of input (usually involves the probabilities of different types of input).

Analyzing Algorithm -- Operations

Some examples – what's the total operations?

- $x = x + 1;$

```
for $(i = 1; i <= n; i++)$
    $x = x + 1;$
```

Linear Loop

```
for $(i = 1; i <= n; i++)$
    for $(j = 1; j <= n; j++)$
        $x = x + 1;$
```

Nested Loop (Quadratic)

Analyzing Algorithm -- Operations

Some examples – what's the total operations?

- for (*i* = 1; *i* <= *n*; *i* *=*2*)

```
for (i = 1; i <= n; i *=2)
    x = x + 1;
```

- for (*i* = n; *i* >= *1*; *i* /=*2*)

```
for (i = n; i >= 1; i /=2)
    x = x + 1;
```

$1\ 2\ 2^2\ 2^3 \cdots = n$

Logarithmic Loops

$2^{\underbrace{1\,2\,3\,..\,k}_{k}} = n$

Analyzing Algorithm -- Operations

Exercise– what's the total operations?

```
for ($i = 1; i <= n; i++$)
    for ($j = 1; j <= n; j *= 2$)
        statement block;
```

$n \log_2 n$

Order of Growth

- The ORDER of a running-time function $\theta(n)$ is the fastest growing term, discarding constant factors.
- Insertion Sort
  - Best Case: $an + b \rightarrow \qquad \theta(n)$
  - Worst Case: $an^2 + bn + c \rightarrow \qquad \theta(n^2)$

Order of Growth

- Most programs are modularized, and use functions.

- How does one determine the complexity of a program containing module A - $\theta(n^2)$ followed by module B - $\theta(2^n)$?
  - $n^2 + 2^n = O(?)$

$= O(2^n)$

Order of Growth

- Sub-linear, Linear, Polynomial and Exponential

$\theta(1)$ $\theta(\log n)$	(constant time) (sub-linear)
$\theta(\mathrm{n})$	(linear)
$\theta(n \log n)$ $\theta(n^2)$ $\theta(n^3)$	(linear) (quadratic) (cubic)
$\theta(2^n)$ $\theta(n!)$	(exponential) (factorial)

Order of Growth

- Sub-linear, Linear, Polynomial and Exponential

$$1 < \log n < n < n \log n < n^2 < n^3 < 2n < n!$$

Big-O

- We are more interested in knowing the generic order of the magnitude of the algorithm instead of the exact operations.
  - 10 v.s. 20, not much difference
  - 10 v.s. 1000, a matter of concern
- Number of data n, executions can be defined as $f(n)$
- Dominant factor of $f(n)$ is sufficient to determine the order of the magnitude

  → $O(n)$

Big-O

- **Definition**

If $f(n)$ and $g(n)$ are the functions defined on a positive integer number ***n****, then*

$f(n) = O(g(n))$ (read: f is Big-“O” of g)

or written as $f(n) \in O(g(n))$

if and only if positive constants c and n exist, such that

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

Big-O

- Constant c could depend on
  - the programming language used,
  - the quality of the compiler or interpreter,
  - the CPU speed,
  - the size of the main memory and the access time to it,
  - the knowledge of the programmer,
  - the algorithm itself, which may require simple but also time-consuming machine instructions

Big-O

- How to understand the definition?
  - a strict upper bound for $f(n)$ --> worst case
  - f is (asymptotically) ≤ g
  - Big-O is actually Omicron, but it suffices to write "O"

- Examples
  - $g(n)=O(n^3)$ and $f(n)$ can include: n^3, $n^3 + n$, $5n^3 + 10$.

Big-O

- Another (more mathematical) **Definition**

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that $f(n) \in O(g(n))$

if $\exists c \in R^+$ and $n_0 \in N$ such that for every integer $n \geq n_0$, $f(n) \leq cg(n)$.

Big-O Example

- Show that $2n=O(n^2)$

By definition, we need to find a constant c such that

$$f(n) \le cg(n)$$

$$2n \le c\,n^2$$

$$\frac{2}{n} \le c$$

$c = 2;\ n_0 = 1$

Can we do better on big-O?

Big-O Exercise

- Show that $2n=O(n)$

By definition, we need to find a constant c such that

$$f(n) \le cg(n)$$

$$2n \le cn$$

$c = 2; n_0 = 1$

Omega Notation (Ω)

- A tight lower bound for *f(n)*.
  - The function can never do better than the specified value, but it may do worse
- **Definition**

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that *or* $f(n) \in \Omega(g(n))$

if $\exists c \in R^+$ and $n_0 \in N$ such that for every integer $n \geq n_0$, $f(n) \geq cg(n)$.

Omega Notation (Ω)

- How to understand the definition?
  - a strict lower bound for $f(n)$ --> best case
  - f is (asymptotically) ≥ g
- Examples
  - $g(n)=\Omega(n^2)$ *and* $f(n)$ *can include:* n^2, $n^3 + n^2$.

Omega Notation (Ω) Example

- Show that $2n \neq \Omega(n^2)$.

By definition, we need to find a constant *c* such that

$$f(n) \geq cg(n)$$

Assume that there is such *c*

$$2n \geq cn2$$

$$c \leq \frac{2}{n}$$

Omega Notation (Ω) Example

- Show that $2n \neq \Omega(n^2)$.

$$c \leq \frac{2}{n}$$

c depends on n. With n increases,

$$\lim_{n\to\infty} \frac{2}{n} = 0.$$

But $c \in R^+$

Omega Notation (Ω) Exercise

- Show that $2n = \Omega(n)$

By definition, we need to find a constant c such that

$$f(n) \geq cg(n)$$

$$2n \geq cn$$

$c = 1;\ n_0 = 1$

Theta Notation (Θ)

- A tight bound for $f(n)$.
- **Definition**

Let f and g be two functions $f, g : N \rightarrow R^+$.

We say that $f(n) \in \Theta(g(n))$

if $f \in \Omega(g)$ and $f \in O(g)$

Theta Notation (Θ)

- How to understand the definition?
  - $\exists c_1, c_2 \in R^+$ *and* $n_0 \in N$, *f(n) is between* $c_1 g(n)$ *and* $c_2 g(n)$, $\forall\ n \geq n_0$
  - f is (asymptotically) = g
- Examples
  - $g(n) = \Theta(n^2)$ *and* $f(n)$ *can include:* n^2, $n + n^2$.

Theta Notation (Θ) Example

- Show that $2n = \Theta(n)$

By definition, we need to find a constant c_1 and c_2 such that

$$c_1 g(n) \le f(n) \le c_2 g(n),$$

$$c_1 n \le 2n \le c_2 n,$$

$c_1 = 1;\ c_2 = 2;\ n_0 = 1$

Theta Notation (Θ) Exercise

- Show that $n + n^2 = \Theta\ (n2)$

By definition, we need to find a constant c_1 and c_2 such that

$$c_1 g(n) \le f(n) \le c_2 g(n),$$

$$c_1\, n^2 \le n + n^2 \le c_2\, n^2,$$

$c_1 = 1/2;\ c_2 = 2;\ n_0 = 2$ Are c_1 , c_2 unique?

Other Notions

- Little o Notation
  - a non-asymptotically tight upper bond
- Little Omega Notation (ϖ)
  - a non-asymptotically tight lower bond

Little-o

- **Definition**

Let f and g be two functions $f, g : N \rightarrow R^{+}$.

We say that $f(n) \in o(g(n))$

if $\exists c \in R^{+}$ and $n_0 \in N$ such that for any $c > 0$, $n_0 > 0$,

$f(n) \leq cg(n)$, for every integer $n \geq n_0$

Little-o

- **Examples**

$5\,n^3 = O(n^3)$

$5\,n^3 \neq o(n^3)$

$5\,n^2 = o(n^3)$

Little Omega Notation (ϖ)

- **Definition**

Let f and g be two functions $f, g : N \to R^{+}$.

We say that *or* $f(n) \in \varpi(g(n))$

if $\exists c \in R^{+}$ and $n_0 \in N$ such that for any $c > 0$, $n_0 > 0$,

$f(n) \geq cg(n)$, for every integer $n \geq n_0$

Little Omega (ϖ)

- **Examples**

$5n^3 = \Omega(n^3)$

$5n^3 \neq \varpi(n^3)$

$5n^3 = \varpi(n^2)$

References and Useful Resources

- Video “Asymptotic Notations 101: Big O, Big Omega, & Theta” https://www.youtube.com/watch?v=0oDAlMwTrLo
- Insertion sort https://www.geeksforgeeks.org/insertion-sort-algorithm/

That's about this lecture!