CHAPTER 3 Algorithms

3.1 Algorithms

1. Introduction

An *algorithm* is a finite set of precise instructions for performing a computation or for solving a problem.

Pseudocode:

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure *max*($a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return *max*{*max* is the largest element}

2. Searching Algorithms

ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{*location* is the subscript of the term that equals x, or is 0 if x is not found}

ALGORITHM 3 The Binary Search Algorithm.

procedure *binary search* (x: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1${i is left endpoint of search interval}
$j := n$ {j is right endpoint of search interval}
while $i < j$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ **then** $i := m + 1$
    else $j := m$
if $x = a_i$ **then** *location* $:= i$
else *location* $:= 0$
return *location*{*location* is the subscript i of the term a_i equal to x, or 0 if x is not found}

3. Some other Algorithms

ALGORITHM 4 The Bubble Sort.

procedure *bubblesort*($a_1, \ldots, a_n$: real numbers with $n \geq 2$)
for $i := 1$ **to** $n - 1$
    for $j := 1$ **to** $n - i$
        if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
{$a_1, \ldots, a_n$ is in increasing order}

ALGORITHM 5 The Insertion Sort.

procedure *insertion sort*($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j := 2$ **to** n
    $i := 1$
    while $a_j > a_i$
        $i := i + 1$
    $m := a_j$
    for $k := 0$ **to** $j - i - 1$
        $a_{j-k} := a_{j-k-1}$
    $a_i := m$
$\{a_1, \ldots, a_n$ is in increasing order$\}$

Algorithms that make what seems to be "best" choice at each step are called greedy algorithms.

3.2 The Growth of Functions

Big-O Notation

【Definition】Let f and g be functions from Z(or R) to R. We say that "$f(x)$ is $O(g(x))$" if there are constants C and k such that $|f(x)| \leq C|\,g(x)|$, whenever $x > k$.

Note:

1. Equivalent expressions:

$$f(x) = O(g(x))$$

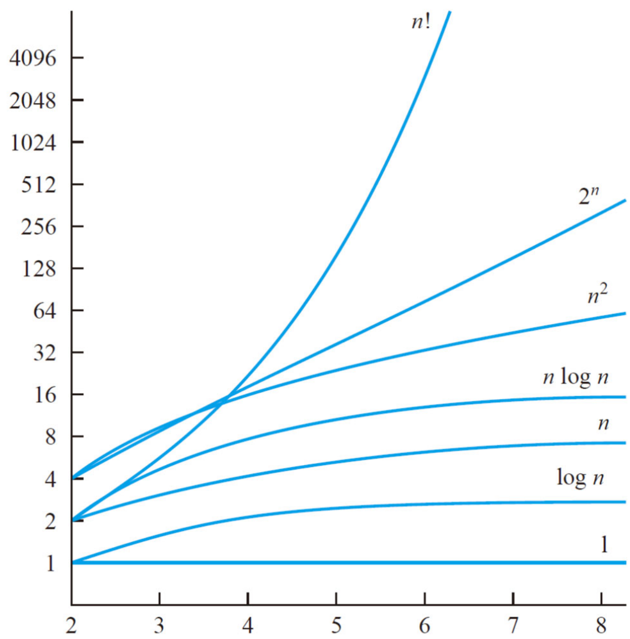
$$f(x) \in O\,(g(x))$$

2. The pair C, k satisfiers the definition is never unique. Moreover, if one such pair exists, there are infinitely many such pairs.

3. When $f(x)$ is $O(g(x))$, and $h(x)$ is a function that has larger absolute values than $g(x)$ does for sufficiently large values of x, it follows that $f(x)$ is $O(h(x))$.

- Two functions $f(x)$ and $g(x)$ that satisfy both of these big-O relationships are *of the same order*.

【Theorem】Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$, where $a_0, a_1, \ldots, a_n$ are real numbers. Then $f(x)$ is $O(x^n)$.



(1) Addition of functions

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1 + f_2)(x)$ is $O(\max(g_1(x),g_2(x)))$.

(2) Multiplication of functions

If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$, then $(f_1f_2)(x)$ is $O(g_1(x)g_2(x))$.

Big-Omega

【Definition】 Let f and g be functions from Z (or R) to R. We say that "$f(x)$ is $\Omega(g(x))$" if there are constants C and k such that $|f(x)| \geq C|g(x)|$, whenever $x > k$.

Big-Theta

【Definition】 Let f and g be functions from Z (or R) to R. We say that "$f(x)$ is $\Theta(g(x))$" if "$f(x)$ is $O(g(x))$" and "$f(x)$ is $\Omega(g(x))$", i.e., there are constants C_1, C_2, and k such that

$0 \leq C_1g(x) \leq f(x) \leq C_2g(x)$, whenever $x > k$.

$f(x)$ is $\Theta(g(x))$ is read as: "$f(x)$ is big-Theta of $g(x)$", "$f(x)$ is of order $g(x)$"

3.3 Complexity of Algorithms

Time Complexity: Time complexity is usually expressed in terms of the number of basic operations (comparisons, arithmetic operations, etc.) rather than the actual computer time used.

Basic operations:

- searching algorithms - key comparisons
- sorting algorithms - list component comparisons
- numerical algorithms - floating point ops. (flops) -

multiplications/divisions and/or additions/subtractions.