计算机科学与技术学院
College of Computer Science and Technology

Chapter 3 The Fundamentals: Algorithms

Discrete Mathematics and Its Applications
Zhejiang University/CS/Course/Xiaogang Jin
E-Mail: xiaogangj@cise.zju.edu.cn

3.1 Algorithms

□ INTRODUCTION

Definition: An <u>algorithm</u>: a finite set of precise instructions for performing a computation or solving problem.

Example: Describe an algorithm for finding the maximum(largest) value in finite sequence of integers.

Solution: We perform the following steps:
1. Set the temporary maximum equal to the first integer in the sequence.
2. Compared the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer.
3. Repeat the previous step if there are more integer in the sequence.
4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

ALGORITHM1 **Finding the Maximum Element**

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

Pseudocode: Instructions given in a generic language similar to a computer language such as C^{++} or Pascal.

Properties of Algorithm:

• Input: the algorithm receives input from a special set.

• Output: the algorithm produces output

• Definiteness: The steps are precisely defined.

• Correctness: Produce the correct output for each set of input values

• Finiteness: Stop after a finite number of steps for any input in the set.

• Effectiveness: Perform each step of an algorithm exactly and in a finite amount of time.

• Generality: Applicable for all problems of desired form, not just for a particular set of input values.

□ SEARCHING ALGORITHMS

Problem: Locate an element x in an ordered list of distinct elements $a_1, a_2, \cdots, a_n$ or determine that it is not in the list.

— Searching problem

ALGORITHM2 **The Linear Search Algorithm**

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

ALGORITHM3 **The Binary Search Algorithm**

```
procedure binary search ($x$ : integer, $a_1, a_2, \cdots, a_n$ : distinct integers)
$i := 1$ { $i$ is left endpoint of search interval}
$j := n$ { $j$ is right endpoint of search interval}
if $i < j$
begin
   $m := \lfloor (i+j)/2 \rfloor$
   if $x > a_m$ then $i := m + 1$
   else $j := m$
end
if $x = a_i$ then $location := i$
else $location := 0$
{$location$ is the subscript of term that equals $x$ or is 0 if $x$ is not found}
```

3.2 Complexity of Algorithms

Definition:

• Complexity: the amount of time and/or space needed to execute the algorithm.

• Space Complexity: be tied with the particular data structures of used to implement the algorithm(not be considered here).

• Time Complexity: can be expressed in terms of the number of operation used by the algorithm when the input has a particular size.

Remark: The operation used to measure the time complexity can be *the comparison, the addition, the multiplication, the division and other basic operation.*

Example:

finding the

Solution:

one to de

```
ALGORITHM1    Finding the Maximum Element

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

reached and another to determine whether to update the temporary maximum. $\cdots\cdots\cdots 2(n-1)$.

when $i = n + 1$ exit the loop. $\cdots\cdots\cdots\cdots 1$

Time complexity of the Algorithm for finding the maximum element in a set is $2(n-1) + 1 = 2n - 1 = \mathcal{O}(n)$.

Example: Describe the time complexity of linear search algorithm.

Solution: At each step of the loop, two comparisons are performed; one more comparison is made outside the loop.

– if $x = a_i$, $2i + 1$ comparison are used.
– The most comparisons, $2n + 1$ are required when the element is not in the list.

Hence, a linear search requires at most $\mathcal{O}(n)$ comparisons.

When x is known to be in the list. The average number of comparisons used equals $\frac{3+5+7+\cdots+(2n+1)}{n} = n+2 = \mathcal{O}(n)$

Remark: Types of Complexity:

- **Best-case time** = minimum time needed to execute the algorithm for inputs of size n

- **Worst-case time** = maximum time needed to execute the algorithm for inputs of size n

- **Average-case time** = average time needed

Example: Describe the time complexity of binary search algorithm.

Solution: Assume there are $n = 2^k$ elements in the list $a_1, a_2, \cdots, a_n$ where k is a nonnegative integer. If n is not the power of 2, the list can be considered part of a larger list with 2^{k+1} elements, where $2^k < n < 2^{k+1}$.

$j := n$ { j is right endpoint of search interval}

At each stage, comparison i and j, and if $i < j$ a comparison is done to determine whether x is greater than the middle term of the restricted list. Finally, when one term is left in the list, one comparison tells us that there are no additional terms left, and one comparison is used to determine if this term is x.

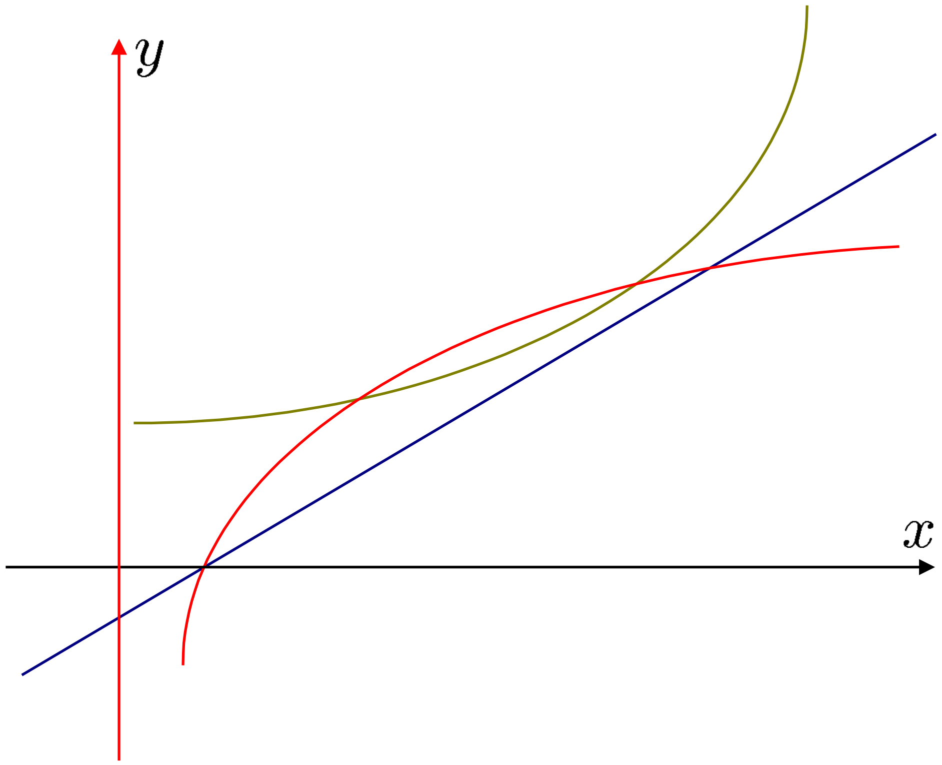
end

Hence, at most $2k + 2 = 2\log n + 2 = \mathcal{O}(\log n)$ comparisons are required to perform a binary search. (if n is not a power of 2, the search requires at most $2\lfloor \log n \rfloor + 2 = \mathcal{O}(\lfloor \log n \rfloor)$). }

Some Terminology to Describe the Time Complexity.

TABLE 1 Commonly Used Terminology for the Complexity of Algorithms

Complexity	Terminology
$\mathcal{O}(1)$	Constant complexity
$\mathcal{O}(\log n)$	Logarithmic complexity
$\mathcal{O}(n)$	Linear complexity
$\mathcal{O}(n \log n)$	$n \log n$ complexity
$\mathcal{O}(n^b)$	Polynomial complexity
$\mathcal{O}(b^n)$ $(b > 1)$	Exponential complexity
$\mathcal{O}(n!)$	Factorial complexity



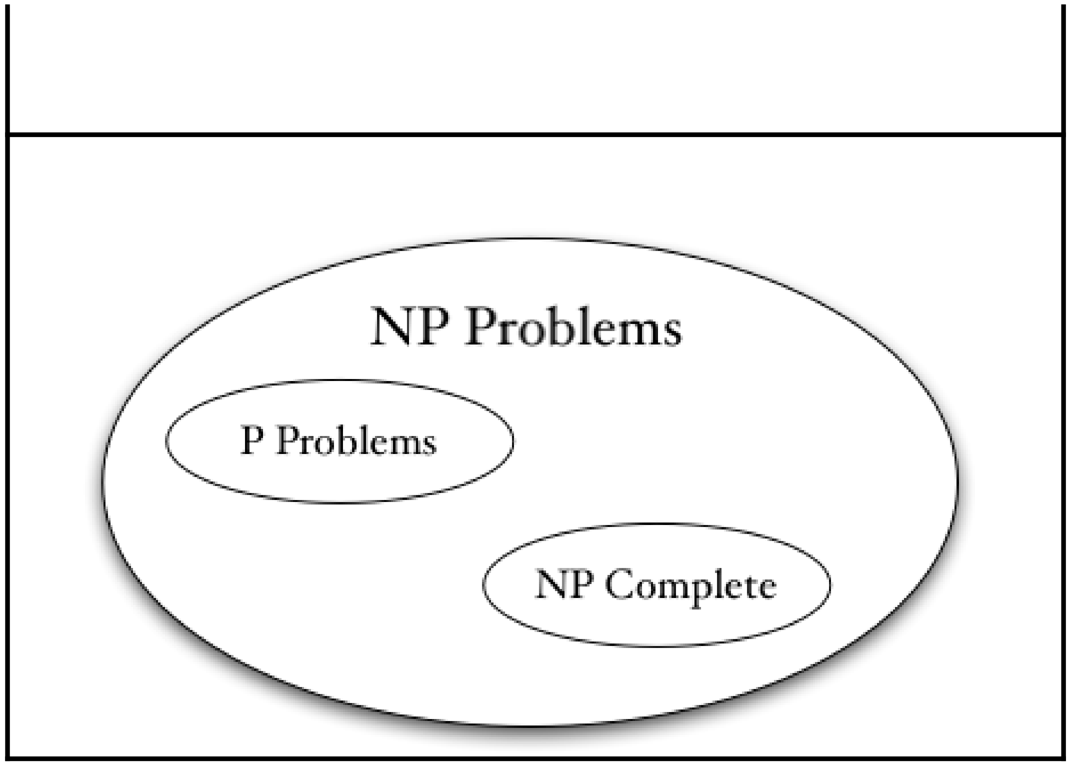
The Growth of Functions

□ $\mathcal{P}$ Class:
Problems can be solved by polynomial time algorithm.

□ $\mathcal{NP}$ Class:
Problems for which a solution can be checked in polynomial time.

□ $\mathcal{NP}$-Complete Problem:
If any of these problems can be solved by polynomial worst-case time algorithm, then all can be solved by polynomial worst-case time algorithms.

- The World As We Don't Know it


NP Problems
P Problems
NP Complete