Algorithm Efficiency Analysis

Lecturer: Heshan Du
Email: Heshan.Du@nottingham.edu.cn

University of Nottingham Ningbo China

Some slides are based on the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Aim and Learning Objectives

- To be able to write algorithms in *pseudocode*.
- To be able to perform *running time analysis* for algorithms experimentally.
- To be able to *classify algorithms* using important functions and *compare algorithms* w.r.t. their running time.
- To be able to explain the meaning of *big-Oh and its relatives*.
- To be able to apply big-Oh and its relatives to perform running time analysis for algorithms.

Reading

M. T. Goodrich, R. Tamassia and M. H. Goldwasser, *Data Structures and Algorithms in Java*, 6th Edition, 2014.

- Chapter 1, Section 1.8.2. Pseudocode
- Chapter 4. Analysis Tools

Algorithms

- An algorithm is a solution to a *computational problem* that specifies (in general terms) a desired *input-output relationship*.
- An algorithm describes a specific *step-by-step procedure* for solving the problem.
- The procedure takes a value (or values) as *input,* and produces a value (or values) as *output*.
- The input is an *instance* of the computational problem, and the output is the corresponding solution.

Example: sorting a sequence of numbers

- The computational problem of "*sorting a sequence of n numbers into monotonically increasing order*" can be defined formally in terms of the input-output relationship:

- ***Input***: a sequence of n *numbers* $a_1, a_2, \ldots, a_n$

- ***Output***: a permutation (reordering) $a_1', a_2', \ldots, a_n'$ of the input sequence such that $a_1' \leq a_2' \leq \cdots \leq a_n'$.

- Given the input sequence $1, 3, 2$, the expected output is $1, 2, 3$.

Correctness

- An algorithm for a computational problem is *correct*, if for every legal input instance, the required output is produced.

- Examples of correctness
  - An algorithm to compute the maximum of two numbers is correct, if it returns the larger of the two numbers.
  - An algorithm to sort a list of numbers is correct, if it returns a permutation of the list in which each element is less than or equal to its successor.
  - An algorithm to return the index of the largest element in an array A of n numbers is correct, if it returns an index i, $0 \leq i \leq n-1$ such that for all j, $0 \leq j \leq n-1$, $A[j] \leq A[i]$.

Pseudocode

- Before implementation, programmers are often asked to describe algorithms in a way that is intended for human eyes only.
- Such descriptions are called *pseudocode*.
- It is a mixture of natural language and high-level programming constructs that describe the *main ideas* behind a generic implementation of a data structure or algorithm.

Pseudocode

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notation for describing algorithms
- Hides program design issues

Pseudocode Constructs

- Method declaration

  Algorithm ***method*** (***arg*** [, ***arg***…])

  Input …

  Output …
- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces
- Method call

  method (***arg*** [, ***arg***…])
- Return value

  return ***expression***
- Expressions:

  ← Assignment

  = Equality testing

  n^2 Superscripts and other mathematical formatting allowed

arrayMax Algorithm

Algorithm: arrayMax(A, n)
Input: an array of n integers A
Output: the maximal element in A

$currentMax \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n - 1$ **do**
  if $A[i] > currentMax$ **then**
    $currentMax \leftarrow A[i]$
  end if
end for
return $currentMax$

Running Time

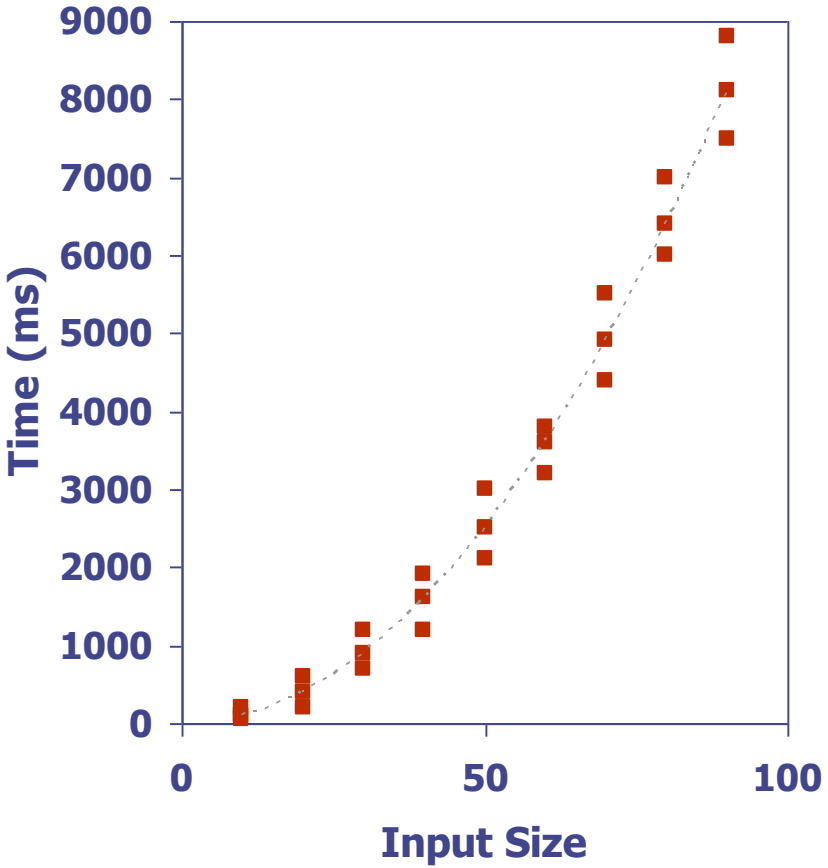
- In general, the running time of an algorithm or data structure operation increases with the input size.
- The running time is affected by
  - hardware environment (e.g., the processor, clock rate, memory, disk)
  - software environment (e.g., the operating system, programming language)

Running Time Analysis

- Focus on *the relationship between the running time of an algorithm and the size of its input*.
- Characterize an algorithm's running time as a *function* of the input size.

Experimental Studies

- Write a program implementing the algorithm
- Run the program (perform *independent* experiments) with inputs of varying size, noting the time needed
- Plot the results



This may be followed by a *statistical analysis* that seeks to fit the best function of the input size to the experimental data.

Record Running Times in Java

```
1  long startTime = System.currentTimeMillis();        // record the starting time
2  /* (run the algorithm) */
3  long endTime = System.currentTimeMillis();          // record the ending time
4  long elapsed = endTime − startTime;                 // compute the elapsed time
```

Limitations of Experiments

- In order to compare two algorithms, the same hardware and software environments must be used.
- Results may not be indicative of the running time on other inputs not included in the experiment.
- It is necessary to implement the algorithm, which may be difficult.

Theoretical Analysis

- Use a high-level description of the algorithm instead of an implementation
- Characterize running time as a function of the input size n
- Take into account all possible inputs
- Allows us to evaluate the speed of an algorithm *independent of* the hardware/software environment

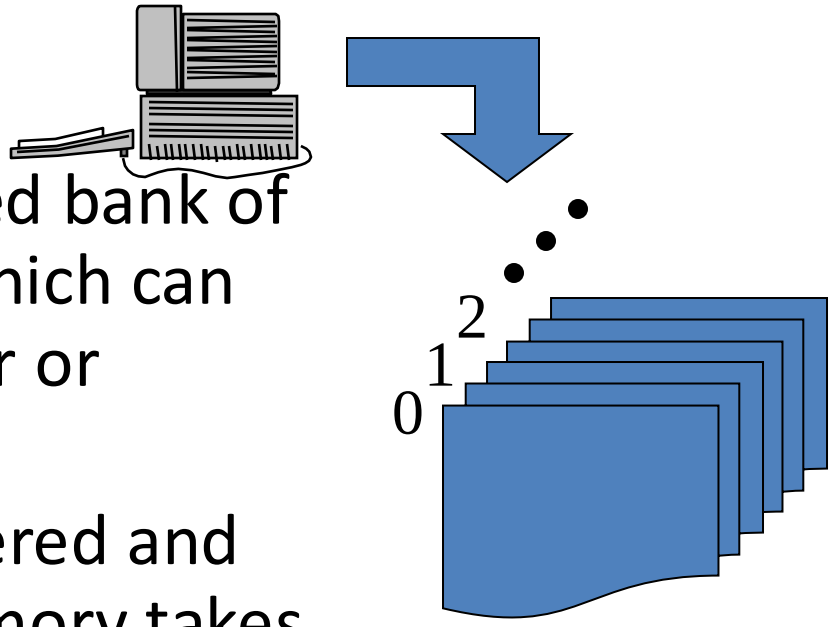
Primitive Operations

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Exact definition not important
- Assumed to take *a constant amount of time* in the RAM model

The Random Access Machine (RAM) Model

A RAM consists of

- A CPU
- An potentially unbounded bank of memory cells, each of which can hold an arbitrary number or character
- Memory cells are numbered and accessing any cell in memory takes unit time

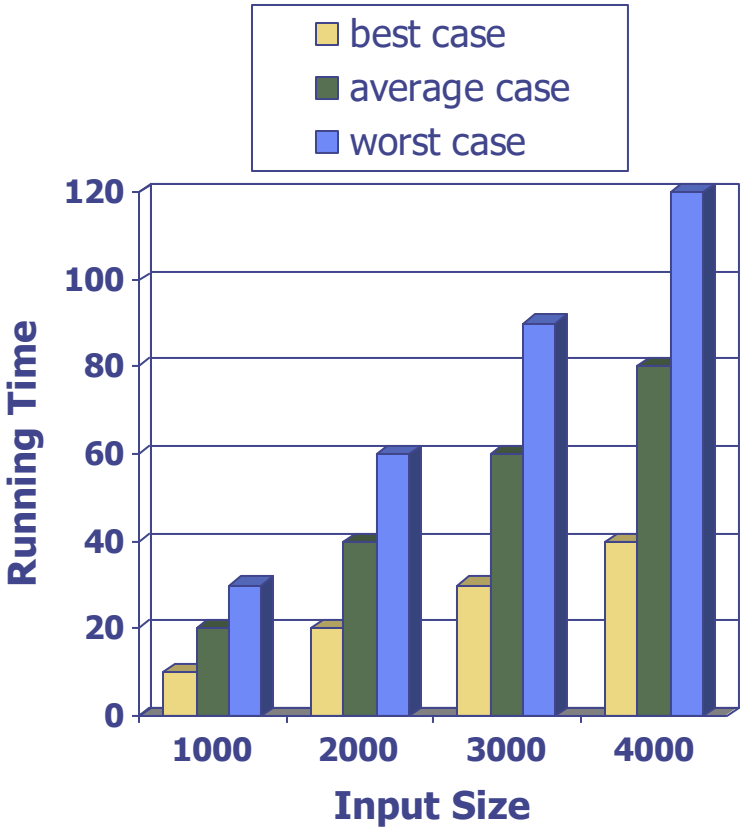


Primitive Operations

- Assigning a value to a variable
- Following an object reference
- Performing an arithmetic operation (+-*/…)
- Comparing two numbers
- Accessing a single element of an array by index
- Calling a method
- Returning from a method

Running Time

- Most algorithms transform input objects into output objects.
- The running time of an algorithm typically grows with the input size.
- Average case time is often difficult to determine.
- We focus on the *worst-case running time.*
  - *Easier to analyze*



Counting Primitive Operations

By inspecting the pseudocode, we count the *maximum* number of primitive operations.

Algorithm: arrayMax(A, n)
Input: an array of n integers A
Output: the maximal element in A

	No. of operations
$currentMax \leftarrow A[0]$	?
for $i \leftarrow 1$ **to** $n-1$ **do**	?
if $A[i] > currentMax$ **then**	?
$currentMax \leftarrow A[i]$	?
end if	
end for	
return $currentMax$	?

Counting Primitive Operations

By inspecting the pseudocode, we count the *maximum* number of primitive operations.

Algorithm: arrayMax(A, n)	No. of operations
Input: an array of n integers A	
Output: the maximal element in A	
$currentMax \leftarrow A[0]$	2
for $i \leftarrow 1$ **to** $n-1$ **do**	1
{test counter: $i \leq (n-1)$}	$2n$
if $A[i] > currentMax$ **then**	$2(n-1)$
$currentMax \leftarrow A[i]$	$2(n-1)$
end if	
{increment counter: $i \leftarrow i+1$}	$2(n-1)$
end for	
return $currentMax$	1
	Total: $8n-2$

Estimating Running Time

- Algorithm arrayMax executes $8n - 2$ primitive operations *in the worst case*.
- Primitive operations are assumed to take a constant amount of time c in the RAM model.
- Let $T(n)$ denote the worst-case time of arrayMax. Then $T(n) = c(8n - 2)$.
- Hence $T(n)$ is a linear function.
- Hence arrayMax runs in *linear time*.

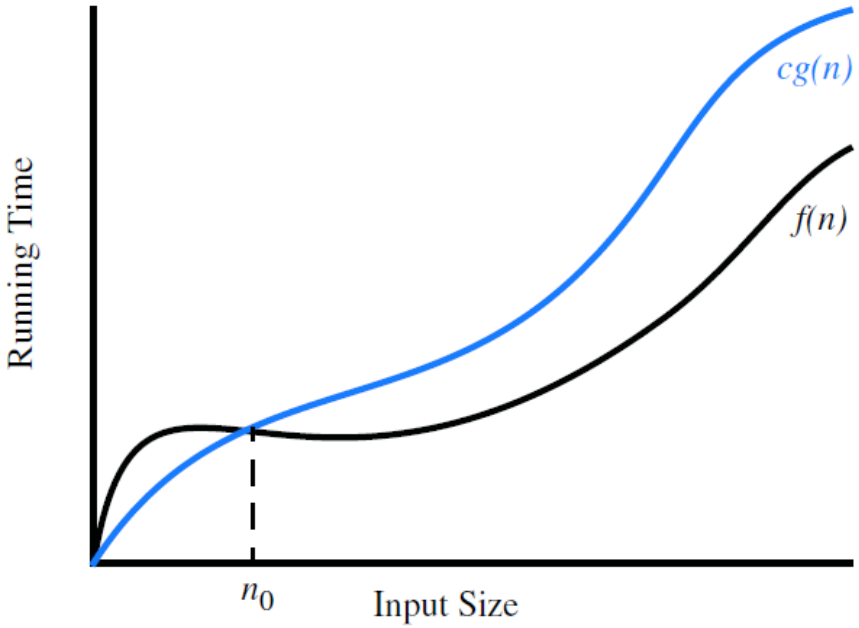
Growth Rate of Running Time

- Changing the hardware/software environment
  - Affects $T(n)$ by a constant factor, but
  - Does not alter the growth rate of $T(n)$
- The *linear growth rate* of $T(n)$ is an *intrinsic property* of the algorithm *arrayMax*
- The growth rate is not affected by constant factors or lower-order terms.
  - $10^2 n + 10^5$ is a linear function
  - $10^5 n^2 + 10^8 n$ is a quadratic function

Big-Oh

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers.

We say that $f(n)$ is $O(g(n))$, if there exist a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for every $n \geq n_0$, $f(n) \leq cg(n)$.



Exercise 1

Show that $2n + 10$ is $O(n)$.

Exercise 2

Is the function n^2 $O(n)$? Why?

Big-Oh and Growth Rate

- The Big-Oh notation gives an upper bound on the growth rate of a function.
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$.
- We can use the Big-Oh notation to rank functions according to their growth rates.

	$f(n)$ is $O(g(n))$	$g(n)$ is $O(f(n))$
$g(n)$ grows more	Yes	No
$f(n)$ grows more	No	Yes
Same growth	Yes	Yes

Examples

Rank the functions based on their asymptotic growth rates

- 100
- $2^n + n^3$
- $10^{100}n + n^{200}$

Big-Oh Rules

- If $f(n)$ is a polynomial of degree d, this is,
  $f(n) = a_0 + a_1 n + \cdots + a_d n^d$,
  and $a_d > 0$, then $f(n)$ is $O(n^d)$, i.e.,
  1. Drop lower-order terms
  2. Drop constant factors
- Use the smallest possible class of functions
  - Say "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- Use the simplest expression of the class
  - Say "$3n + 5$ is $O(n)$" instead of "$3n + 5$ is $O(3n)$"

Asymptotic Algorithm Analysis

- The asymptotic analysis of an algorithm determines the running time in Big-Oh notation.
- To perform the asymptotic analysis
  - We find the worst-case number of primitive operations executed as a function of the input size
  - We express this function with Big-Oh notation
- Example: *arrayMax* runs in $O(n)$ time.
- *Since constant factors and lower-order terms are eventually dropped, we can disregard them when counting primitive operations*.

Relatives of Big-Oh

- Big-Omega
- Big-Theta
- Little-Oh

Big-Omega

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers.

We say that $f(n)$ is $\Omega(g(n))$, if there exist a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that for every $n \geq n_0$,

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

Big-Theta

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers.

We say that $f(n)$ is $\Theta(g(n))$, if there are real constants $c' > 0$ and $c'' > 0$ and an integer constant $n_0 \geq 1$ such that for every $n \geq n_0$,

$$c'g(n) \leq f(n) \leq c''g(n).$$

Little-Oh

Let $f(n)$ and $g(n)$ be functions mapping positive integers to positive real numbers.

We say that $f(n)$ is $\mathrm{o}(g(n))$, if for all positive real constant c, there exists an integer constant $n_0 \geq 1$ such that for every $n \geq n_0$,

$$f(n) < cg(n).$$

Exercise 3

Is the function n^2 o(n)? Why?

Intuition for Asymptotic Notation

- $f(n)$ is $O(g(n))$ if $f(n)$ is asymptotically less than or equal to $g(n)$
- $f(n)$ is $\Omega(g(n))$ if $f(n)$ is asymptotically greater than or equal to $g(n)$
- $f(n)$ is $\Theta(g(n))$ if $f(n)$ is asymptotically equal to $g(n)$
- $f(n)$ is $o(g(n))$ if $f(n)$ is asymptotically strictly less than $g(n)$