CS206 Data Structures

Analysis Tools: Part I

Sung-eui Yoon (윤성의)

Department of Computer Science

KAIST

http://sglab.kaist.ac.kr/~sungeui

KAIST
Korea Advanced Institute of Science and Technology
한국과학기술원

Class Objectives

- ☐ Understand the RAM model for the theoretical analysis of algorithms
- ☐ Get to know the growth rate of running time and big-Oh notation

What is an Algorithm?

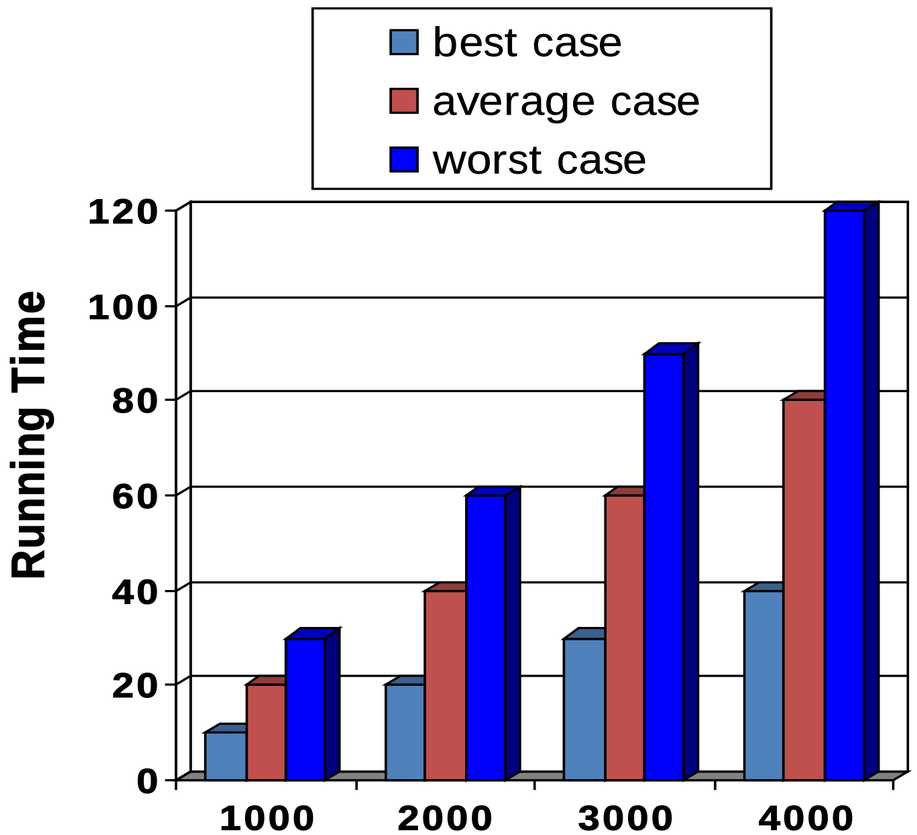
- An algorithm is a step-by-step procedure for solving a well-defined problem in a finite amount of time
  - The problem is defined in terms of input and output

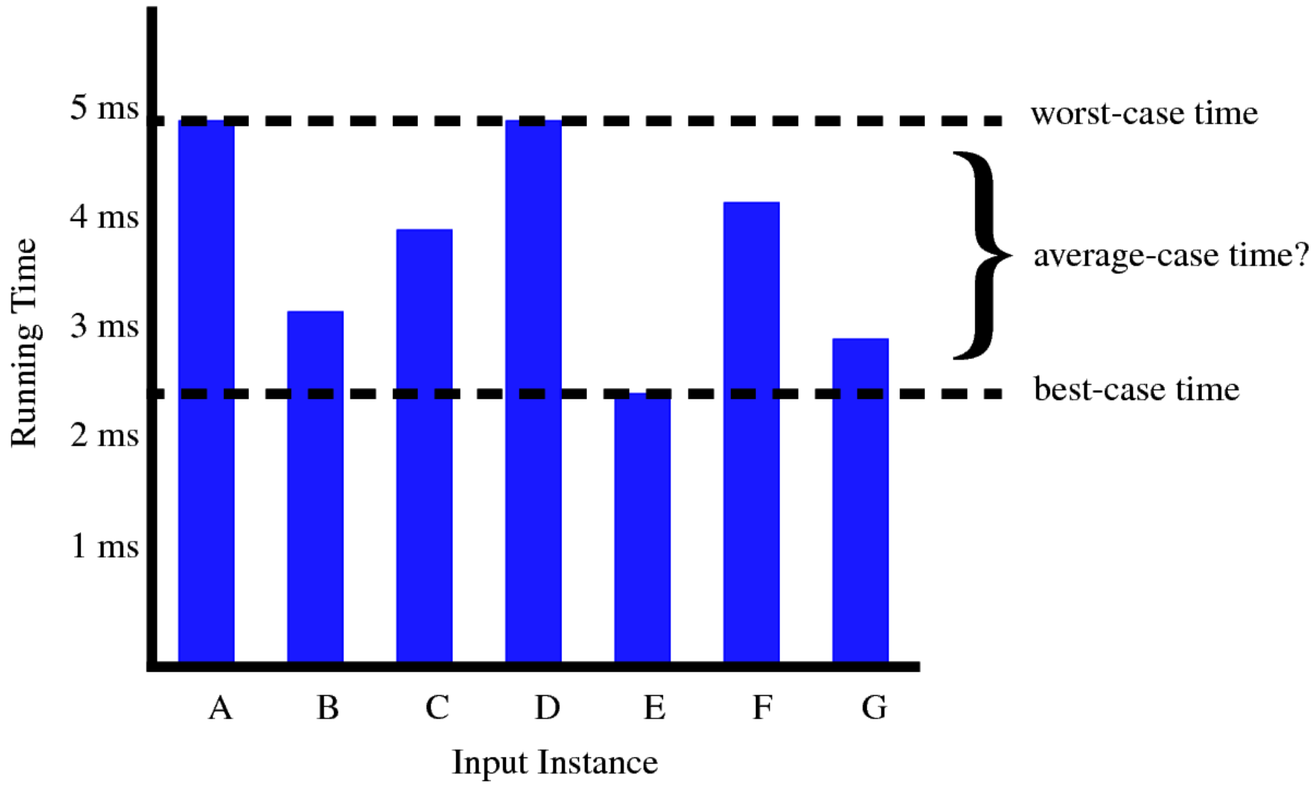
```
Algorithm InsertionSort(A)
    Input: An array A of n comparable elements
    Output: The array A with elements rearranged in non-decreasing order
      for i Ã 1 to n ¡ 1 do
        // Insert A[i] at its proper location in A[0]; A[1]; : : : ; A[i ¡ 1]
        ...
      end for
```

  - There can be many algorithms for the same problem
    - e.g., Quick Sort, Merge Sort, etc.
- Analysis of algorithms involve the following criteria
  - Correctness
  - Amount of time spent
  - Amount of space used
  - Simplicity, clarity
  - Optimality

Running Time

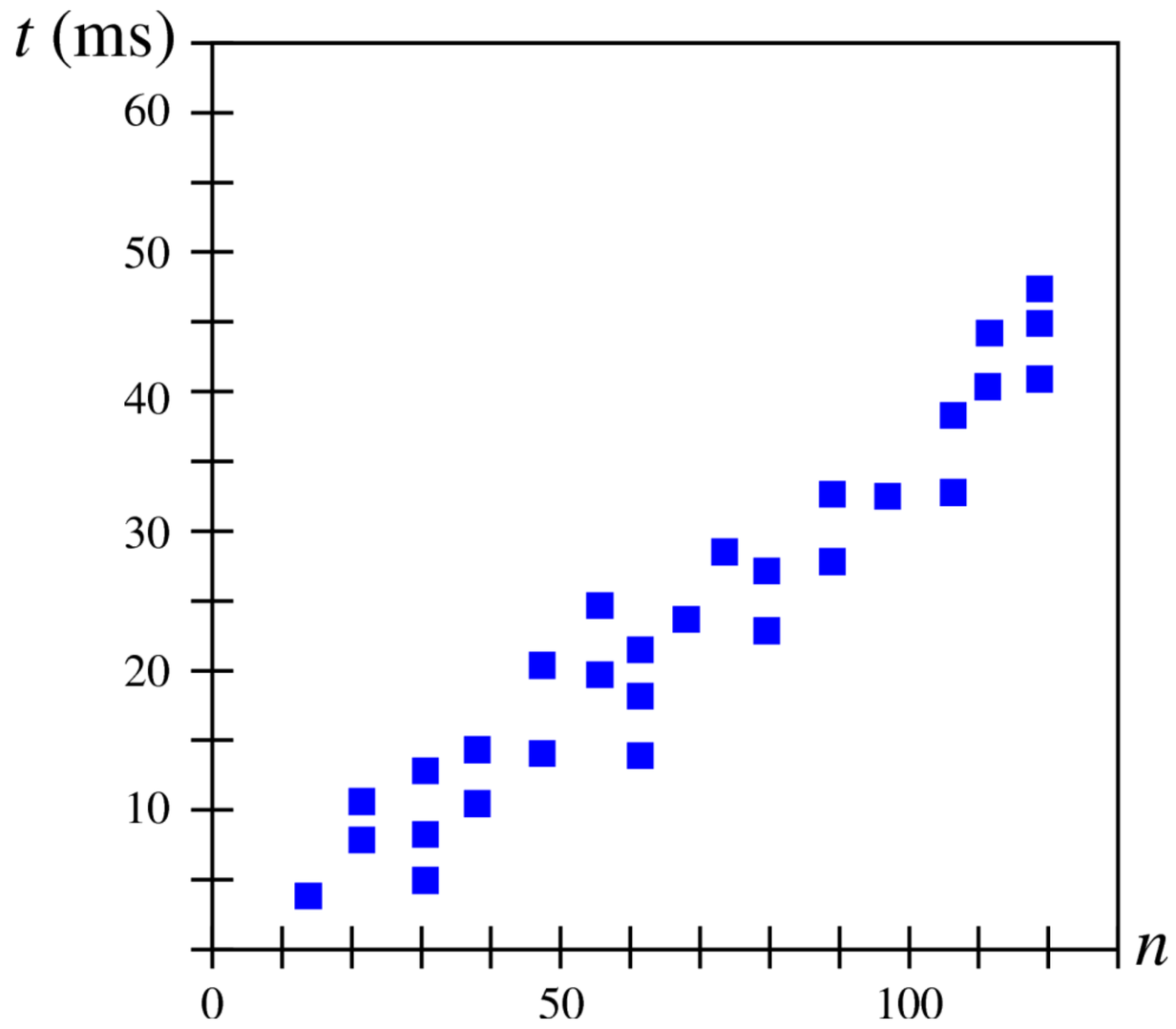
- Most algorithms transform input objects into output objects
- The running time of an algorithm typically grows with the input size
- Average case time is often difficult to determine
- We focus on the worst case running time
  - Easier to analyze
  - Crucial to applications such as games, finance and robotics





Experimental Studies

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Use a method like `System.currentTimeMillis()` to get an accurate measure of the actual running time
- Plot the results or perform statistical analysis



Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult
- 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

- When analyzing a sorting algorithm, can you generate all possible integer sequences of size n?

Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, n.
- Takes into account all possible inputs
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment

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

Example: find max element of an array

```
Algorithm arrayMax(A, n)
    Input: array A of n integers
    Output: maximum element of A
      currentMax Ã A[0]
      for i Ã 1 to n ¡ 1 do
        if A[i] > currentMax then
          currentMax A A[i]
      end if
    end for
    return currentMax
```

Pseudocode Details

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

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

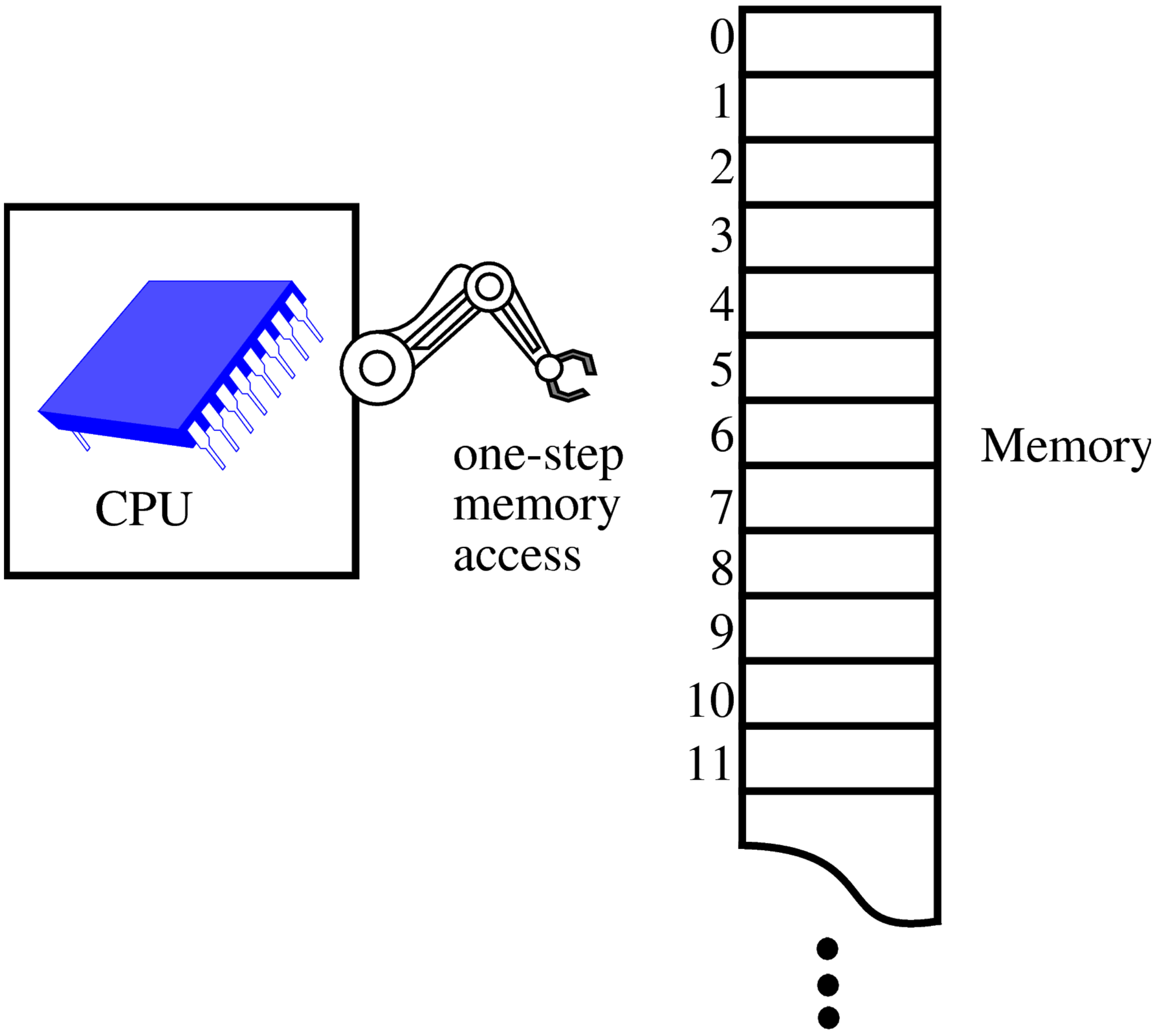
  Input …

  Output …
- Method call

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

  return *expression*
- Expressions
  - ← Assignment (like = in Java)
  - = Equality testing (like == in Java)
  - n^2 Superscripts and other mathematical formatting allowed

Random Access Machine (RAM) Model



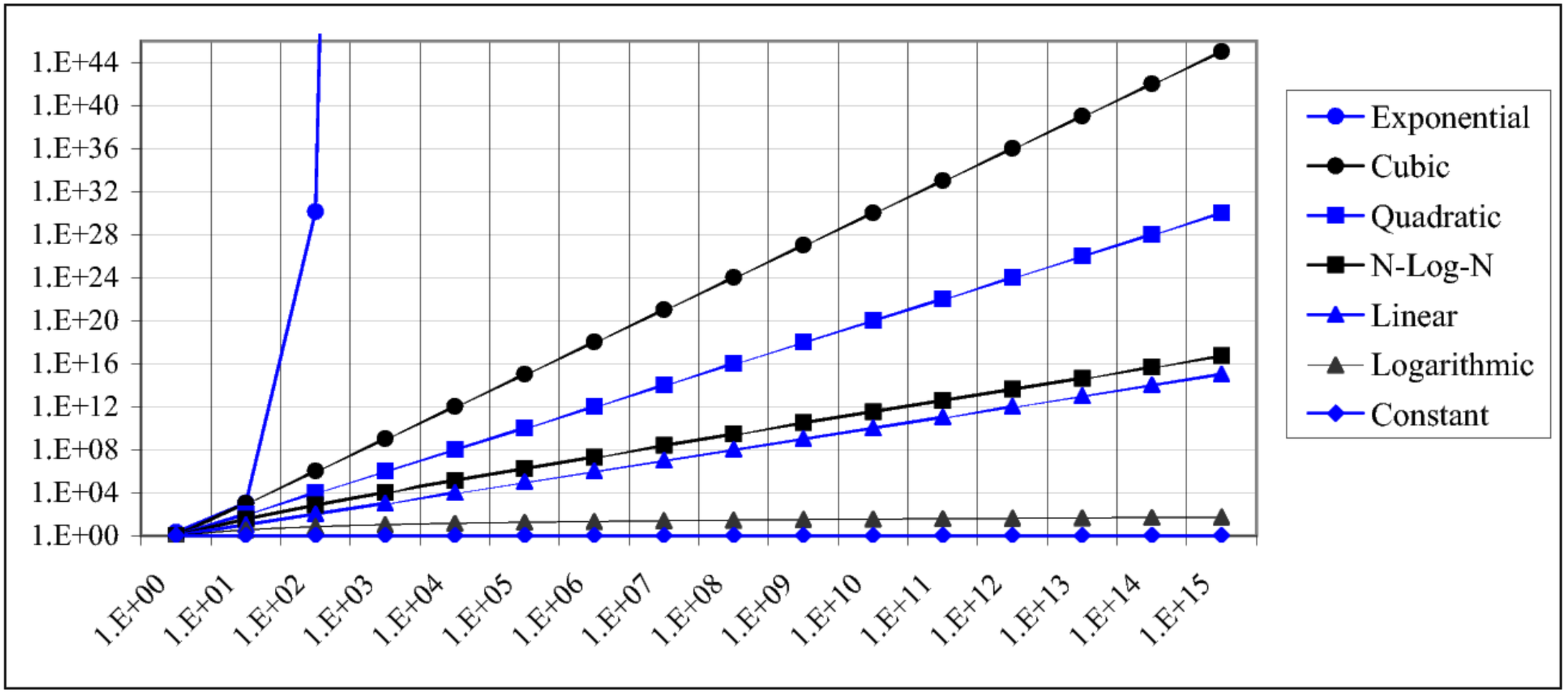
An potentially unbounded bank of memory cells, each of which can hold an arbitrary number or character

Memory cells are numbered and accessing any cell in memory takes unit time

Seven Important Functions

☐ Seven functions that often appear in algorithm analysis:

constant	logarithm	linear	n-log-n	quadratic	cubic	exponent
1	log n	n	n log n	n^2	n^3	2^n



Primitive Operations

- ☐ Basic computations performed by an algorithm
- ☐ Identifiable in pseudocode
- ☐ Largely independent from the programming language
- ☐ Exact definition not important (we will see why later)
- ☐ Assumed to take a constant amount of time in the RAM model

☐ Examples:

- Evaluating an expression
- Assigning a value to a variable
- Indexing into an array
- Calling a method
- Returning from a method

Counting Primitive Operations

- By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size

Algorithm ***arrayMax***(A, n)	# operations
$currentMax \leftarrow A[0]$	2
for $i \leftarrow 1$ **to** $n - 1$ **do**	$2n$
if $A[i] > currentMax$ **then**	$2(n - 1)$
$currentMax \leftarrow A[i]$	$2(n - 1)$
{ increment counter i }	$2(n - 1)$
return *currentMax*	1
	Total $8n - 2$

Estimating Running Time

- Algorithm arrayMax executes 8n – 2 primitive operations in the worst case. Define:
  - a = Time taken by the fastest primitive operation
  - b = Time taken by the slowest primitive operation

- Let T(n) be worst-case time of arrayMax. Then

  $a\,(8n - 2) \leq T(n) \leq b(8n - 2)$

- Hence, the running time T(n) is bounded by two linear functions

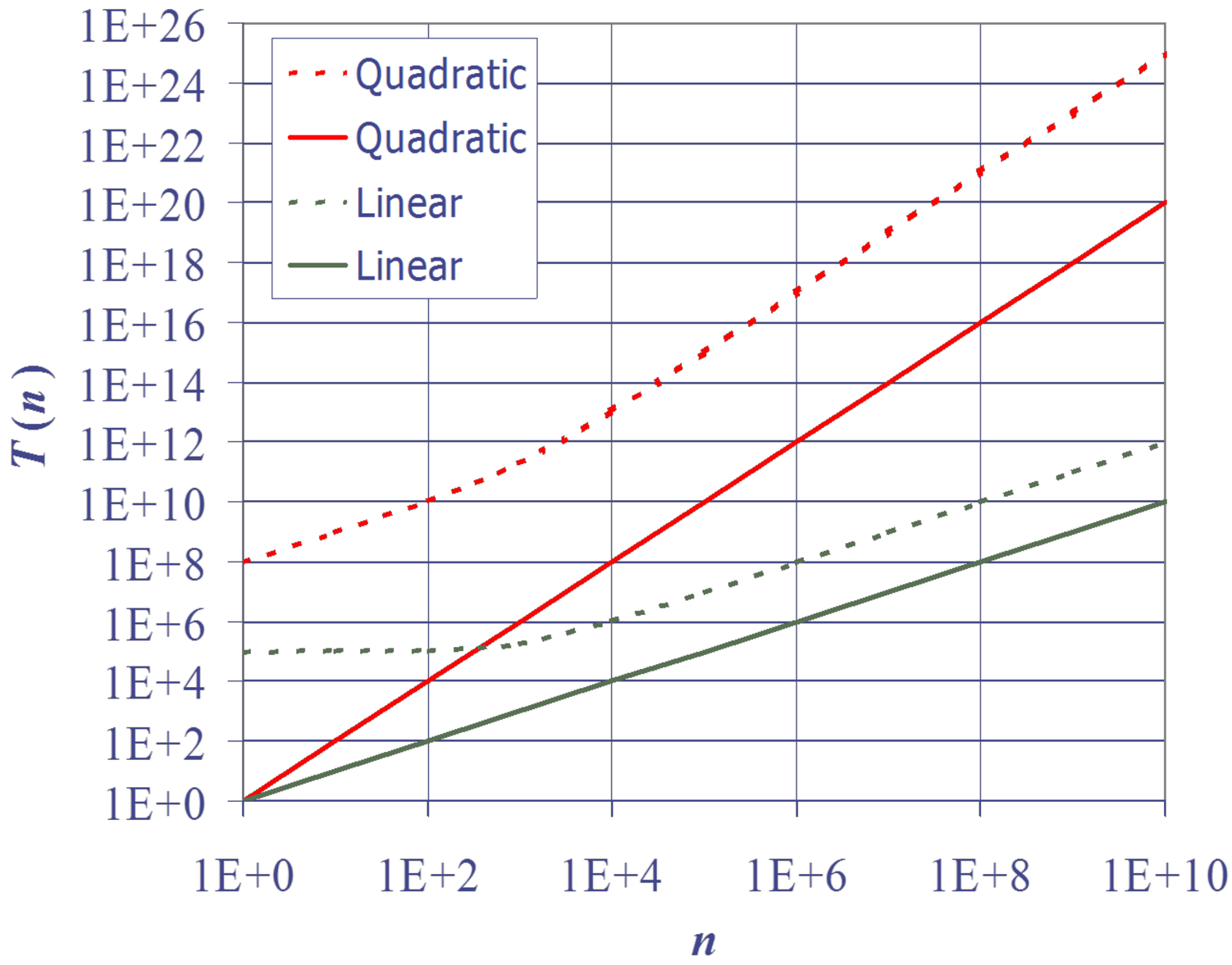
Growth Rate of Running Time

- 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 the running time T(n) is an intrinsic property of algorithm arrayMax

Constant Factors

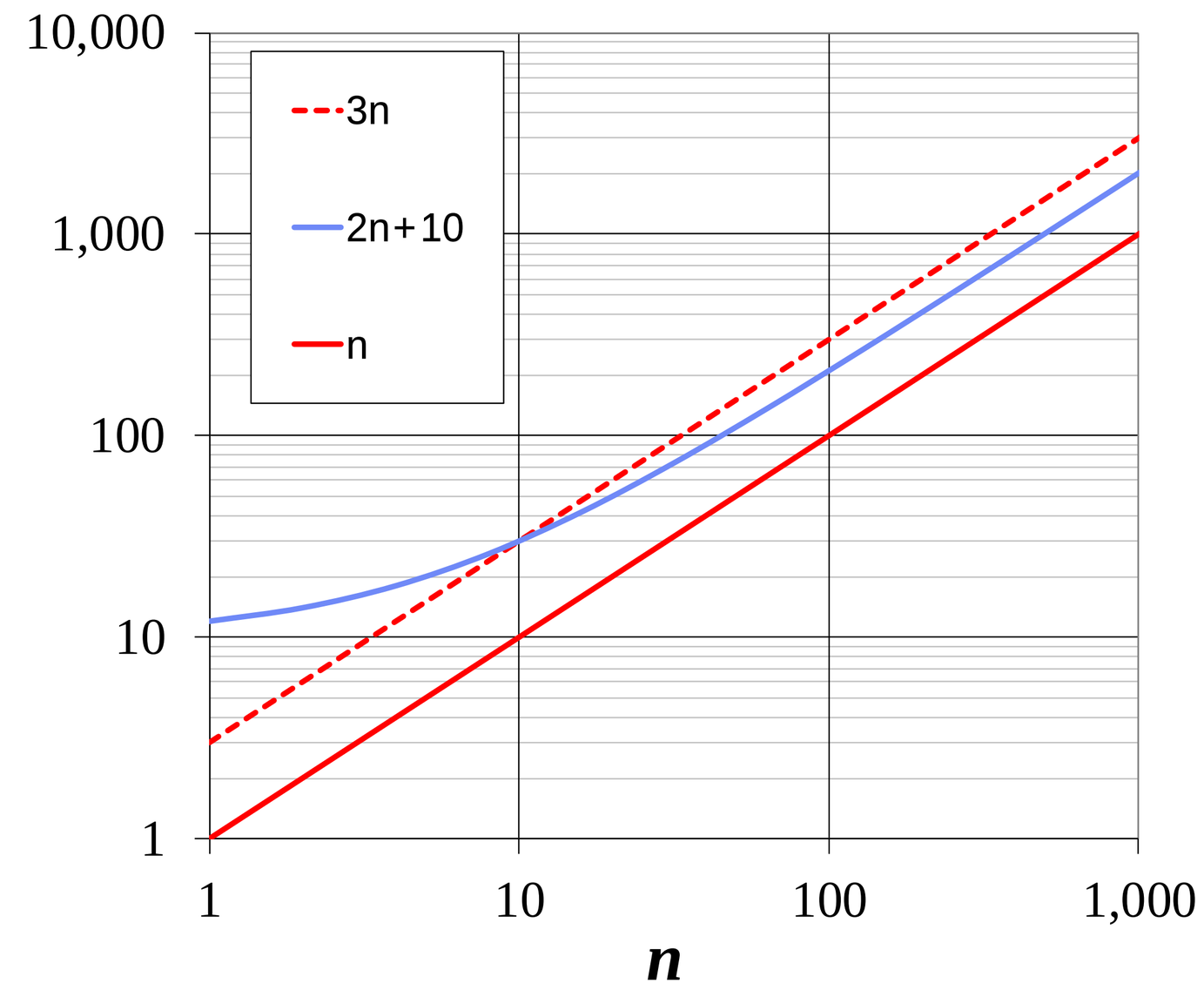
- The growth rate is not affected by
  - constant factors or
  - lower-order terms
- Examples
  - $10^2n + 10^5$ is a linear function
  - $105n^2 + 10^8n$ is a quadratic function



Big-Oh Notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

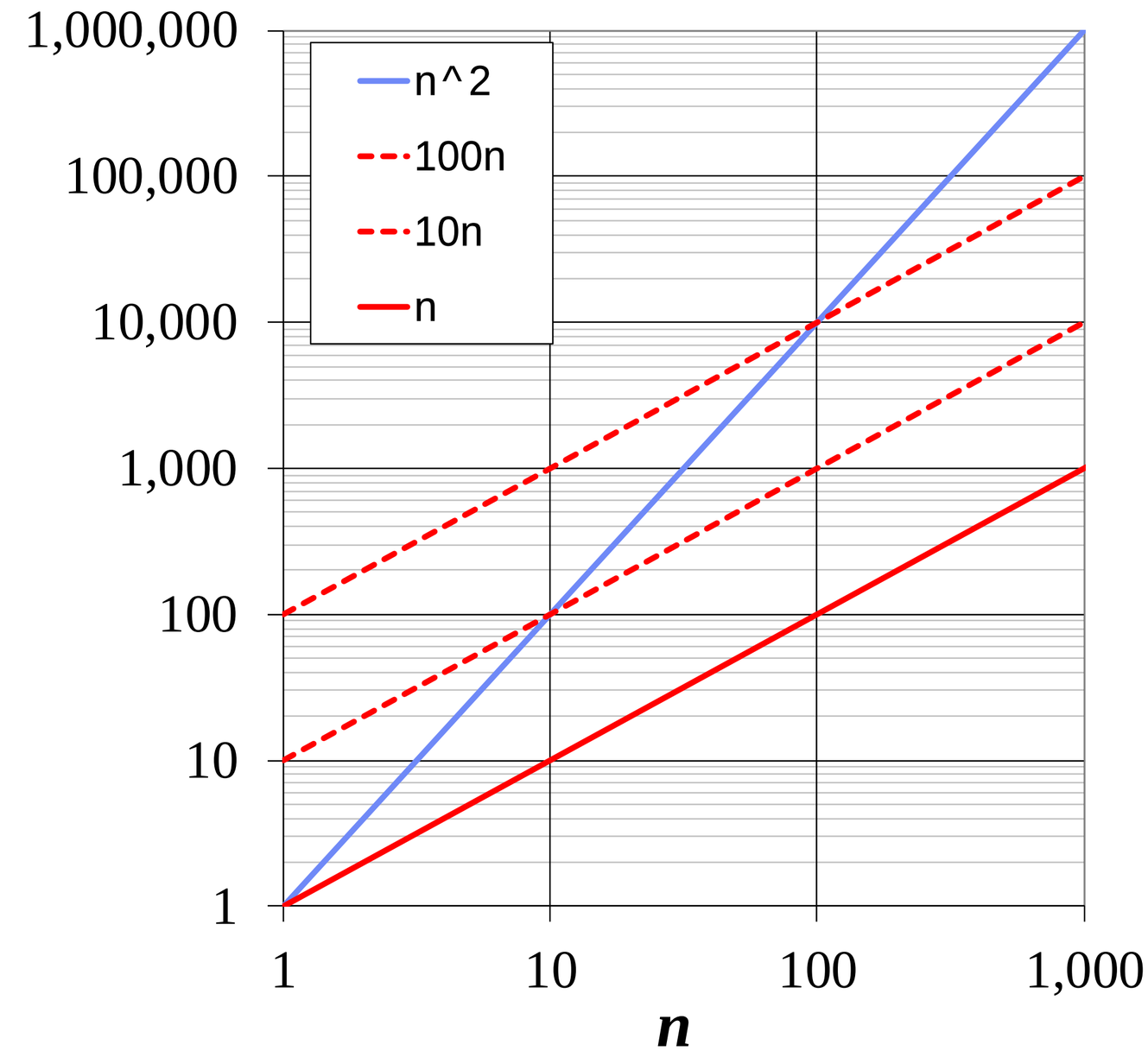
  $f(n) \leq cg(n)$ for $n \geq n_0$



- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$

Big-Oh Example

- Example: the function n^2 is not $O(n)$
  - $n^2 \leq cn$
  - $n \leq c$
  - The above inequality cannot be satisfied since c must be a constant



More Big-Oh Examples

- 7n-2

  7n-2 is O(n)

  need $c > 0$ and $n_0 \geq 1$ s.t. $7n-2 \leq c \bullet n$ for $n \geq n_0$

  this is true for $c = 7$ and $n_0 = 1$

- $3n^3 + 20n^2 + 5$

  $3n^3 + 20n^2 + 5$ is $O(n^3)$

  need $c > 0$ and $n_0 \geq 1$ s.t. $3n^3 + 20n^2 + 5 \leq c \bullet n^3$ for $n \geq n_0$

  this is true for $c = 4$ and $n_0 = 21$

- $3 \log n + 5$

  $3 \log n + 5$ is $O(\log n)$

  need $c > 0$ and $n_0 \geq 1$ s.t. $3 \log n + 5 \leq c \bullet \log n$ for $n \geq n_0$

  this is true for $c = 8$ and $n_0 = 2$

Class Objectives were:

- ☐ Understand the RAM model for the theoretical analysis of algorithms
- ☐ Get to know the growth rate of running time and big-Oh notation

Next Time

☐ Analysis tools: big-Omega and big-Thetha

☐ HW:

- Go over the next lecture slides before the class
- Just 10 min ~ 20 min should be okay

Any Questions?

☐ Come up with one question on what we have discussed in the class and submit at the end of the class

- 1 for typical questions
- 2 for questions with thoughts or that surprised me

☐ Write questions at least 4 times

☐ You can type KLMS