Algorithm Efficiency

Chapter 10

Contents

- What Is a Good Solution?
- Measuring the Efficiency of Algorithms

What Is a Good Solution?

- Criterion
  A solution is good if the total cost it incurs over all phases of its life is minimal.
- Keep in mind, efficiency is only one aspect of a solution's cost
- Note: Relative importance of various components of a solution's cost has changed since early days of computing.

Measuring Efficiency of Algorithms

- Comparison of algorithms should focus on significant differences in efficiency
- Difficulties with comparing programs instead of algorithms
  - How are the algorithms coded?
  - What computer should you use?
  - What data should the programs use?

Execution Time of Algorithm

- Traversal of linked nodes – example:

```
Node<ItemType>* curPtr = headPtr;          ← 1 assignment
while (curPtr != nullptr)                  ← n + 1 comparisons
{
   cout << curPtr->getItem() < endl;       ← n writes
   curPtr = curPtr->getNext();             ← n assignments
}  // end while
```

- Displaying data in linked chain of *n* nodes requires time proportional to *n*

Algorithm Growth Rates

- Measure algorithm's time requirement as a function of problem size
- Compare algorithm efficiencies for large problems
- Look only at significant differences.

Algorithm Growth Rates

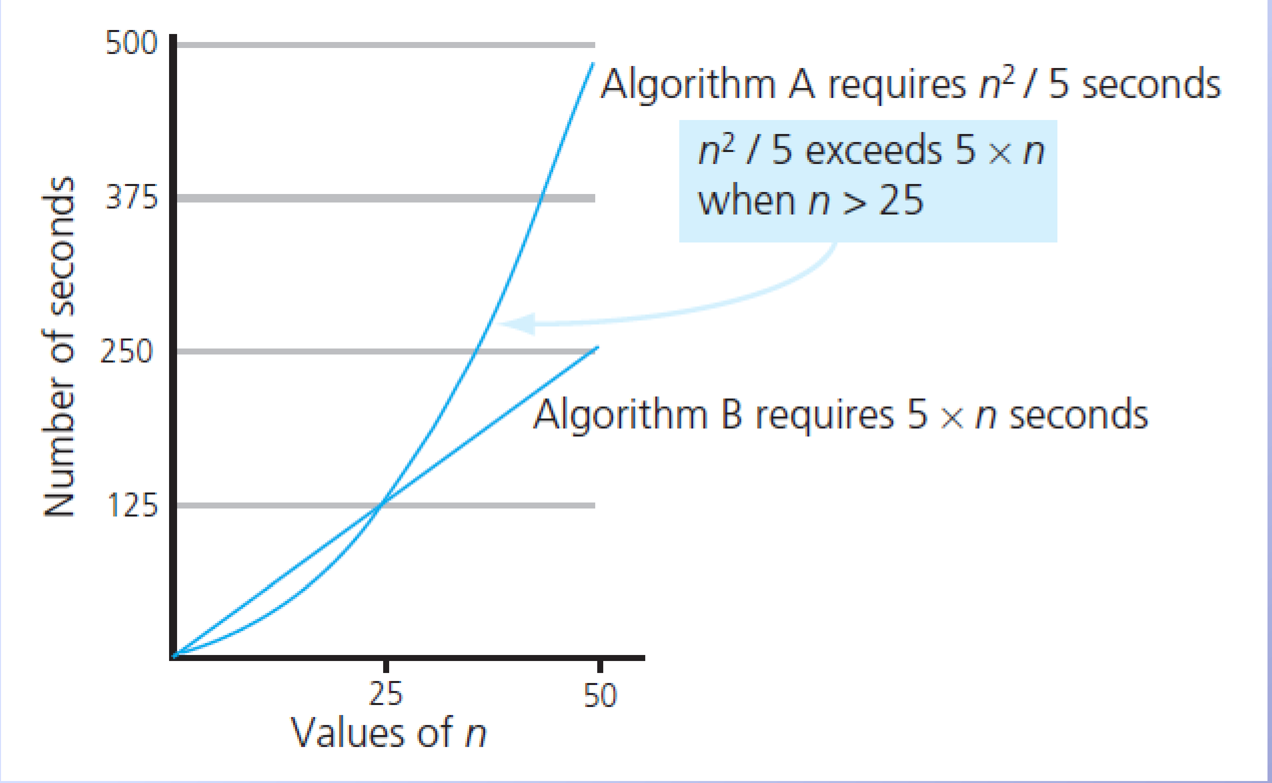

FIGURE 10-1 Time requirements as a function of the problem size n

Analysis and Big O Notation

- Definition:
  - Algorithm *A* is order $f(n)$
    - Denoted $O(f(n))$
  - If constants k and n_0 exist
  - Such that *A* requires no more than $k \times f(n)$ time units to solve a problem of size $n \geq n_0$.

Analysis and Big O Notation

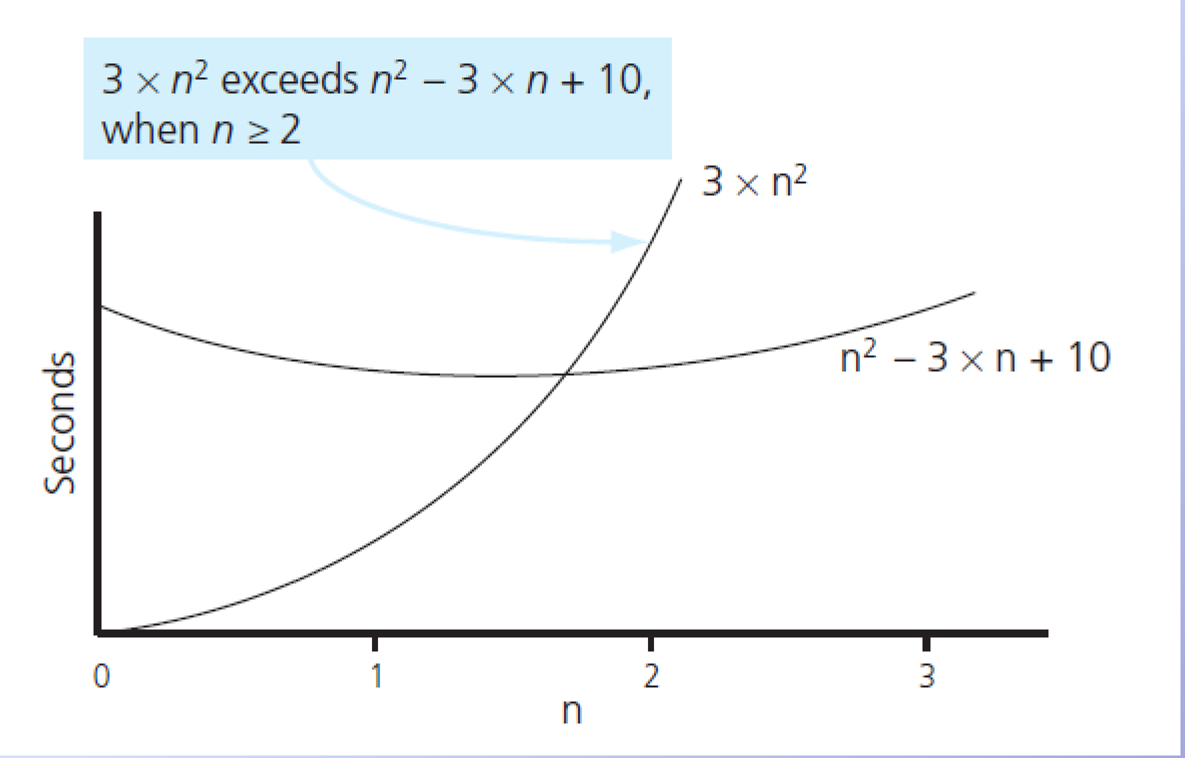

FIGURE 10-2 The graphs of $3 \times n^2$ and $n^2 - 3 \times n + 10$

Analysis and Big O Notation

- Order of growth of some common functions

$$O(1) < O(\log_2 n) < O(n) < O(n \times \log_2 n) <$$
$$O(n^2) < O(n^3) < O(2^n)$$

Analysis and Big O Notation

Function	n: 10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n \times \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

FIGURE 10-3 A comparison of growth-rate functions:
(a) in tabular form

Analysis and Big O Notation

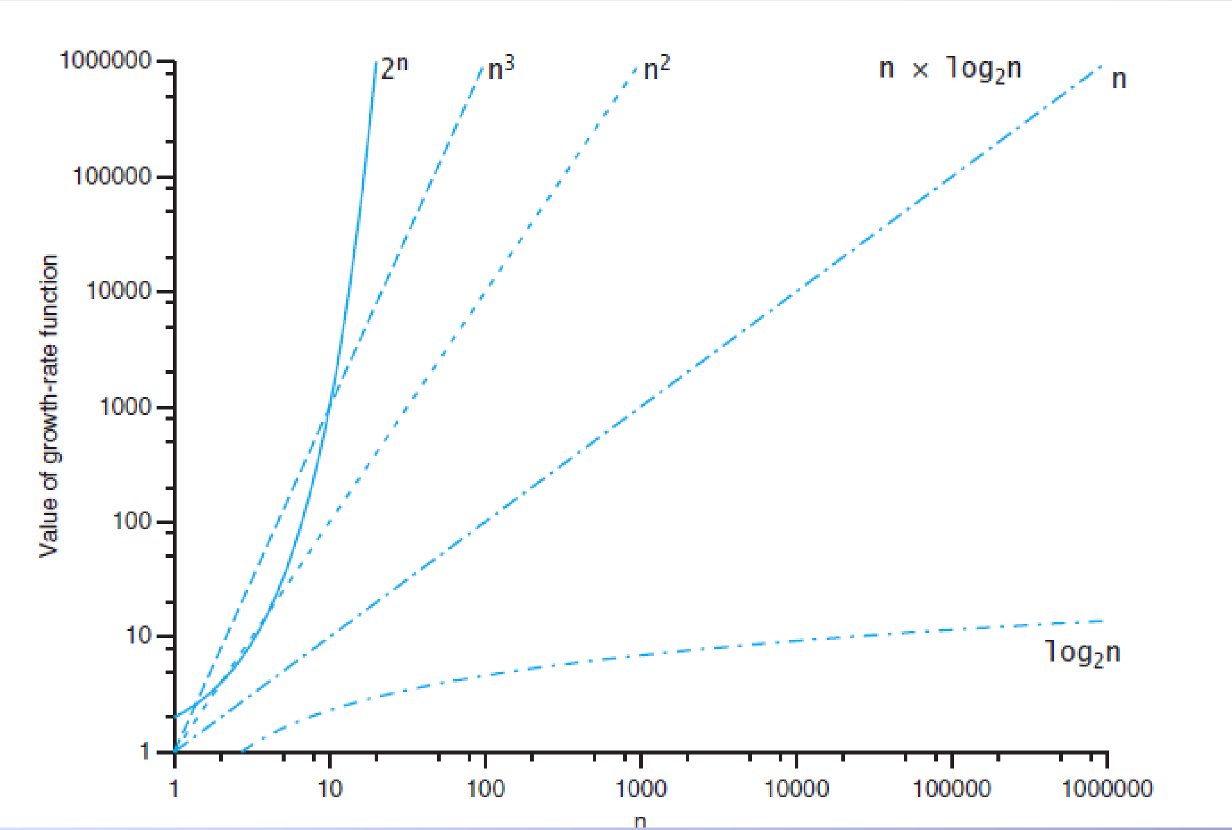

FIGURE 10-3 A comparison of growth-rate functions: (a) in graphical form

Properties of Growth-Rate Functions

- Ignore low-order terms
- Ignore a multiplicative constant in the high-order term
- $O(f(n)) + O(g(n)) = O(f(n) + g(n))$
- Be aware of worst case, average case

Keeping Your Perspective

- Array-based **`getEntry`** is O(1)
- Link-based **`getEntry`** is O(n)
- Consider how frequently particular ADT operations occur in given application
- Some seldom-used but critical operations must be efficient

Keeping Your Perspective

- If problem size always small, ignore an algorithm's efficiency
- Weigh trade-offs between algorithm's time and memory requirements
- Compare algorithms for both style and efficiency

Efficiency of Searching Algorithms

- Sequential search
  - Worst case O(n)
  - Average case O(n)
  - Best case O(1)
- Binary search of sorted array
  - Worst case $O(\log_2 n)$
  - Remember required overhead for keeping array sorted

End

Chapter 10