ECGR 3180
Data Structures and Algorithms

Algorithm Efficiency

Dr. Mihail Cutitaru

Fall 2020

What Is a Good Solution?

- A program incurs a real and tangible cost.
  - Computing time
  - Memory required
  - Difficulties encountered by users
  - Consequences of incorrect actions by program
- A solution is good if …
  - The total cost incurs …
  - Over all phases of its life … is minimal

What Is a Good Solution?

- Important elements of the solution
  - Good structure
  - Good documentation
  - Efficiency
- Be concerned with efficiency when
  - Developing underlying algorithm
  - Choice of objects and design of interaction between those objects

Measuring Efficiency of Algorithms

- Important because
  - Choice of algorithm has significant impact
- Examples
  - Responsive word processors
  - Grocery checkout systems
  - Automatic teller machines
  - Video machines
  - Life support systems

Measuring Efficiency of Algorithms

- Analysis of algorithms
  - The area of computer science that provides tools for contrasting efficiency of different algorithms
  - Comparison of algorithms should focus on significant differences in efficiency
  - We consider comparisons of *algorithms*, not programs

Measuring Efficiency of Algorithms

- Difficulties with comparing programs (instead of algorithms)
  - How are the algorithms coded
  - What computer will be used
  - What data should the program use
- Algorithm analysis should be independent of
  - Specific implementations, computers, and data

The Execution Time of Algorithms

- An algorithm’s execution time is related to number of operations it requires.
  - Algorithm’s execution time is related to number of operations it requires.
- Example: Towers of Hanoi
  - Solution for *n* disks required $2^n – 1$ moves
  - If each move requires time *m*
  - Solution requires $(2^n – 1) \times m$ time units

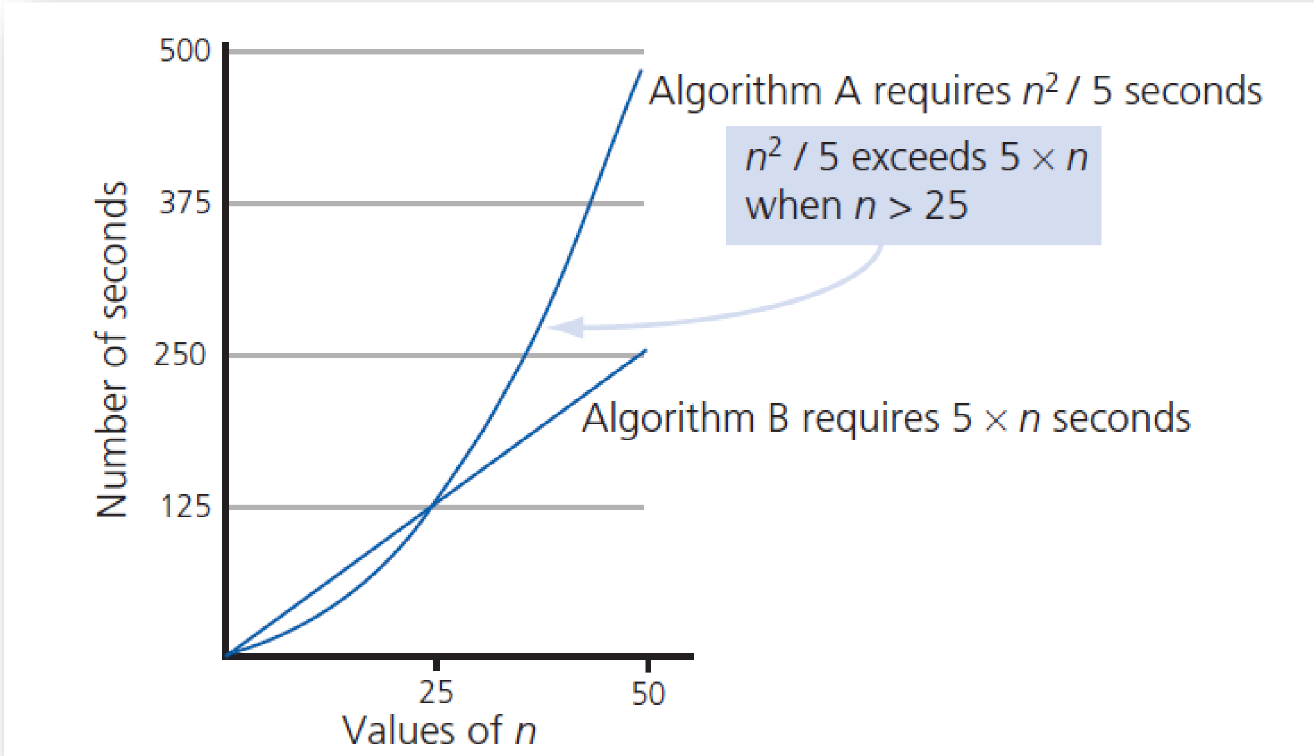
Algorithm Growth Rates

- Measure an algorithm's time requirement as function of problem size
- Most important thing to learn
  - How quickly algorithm's time requirement grows as a function of problem size

Algorithm A requires time proportional to n^2
Algorithm B requires time proportional to n

  - Demonstrates contrast in growth rates

Algorithm Growth Rates

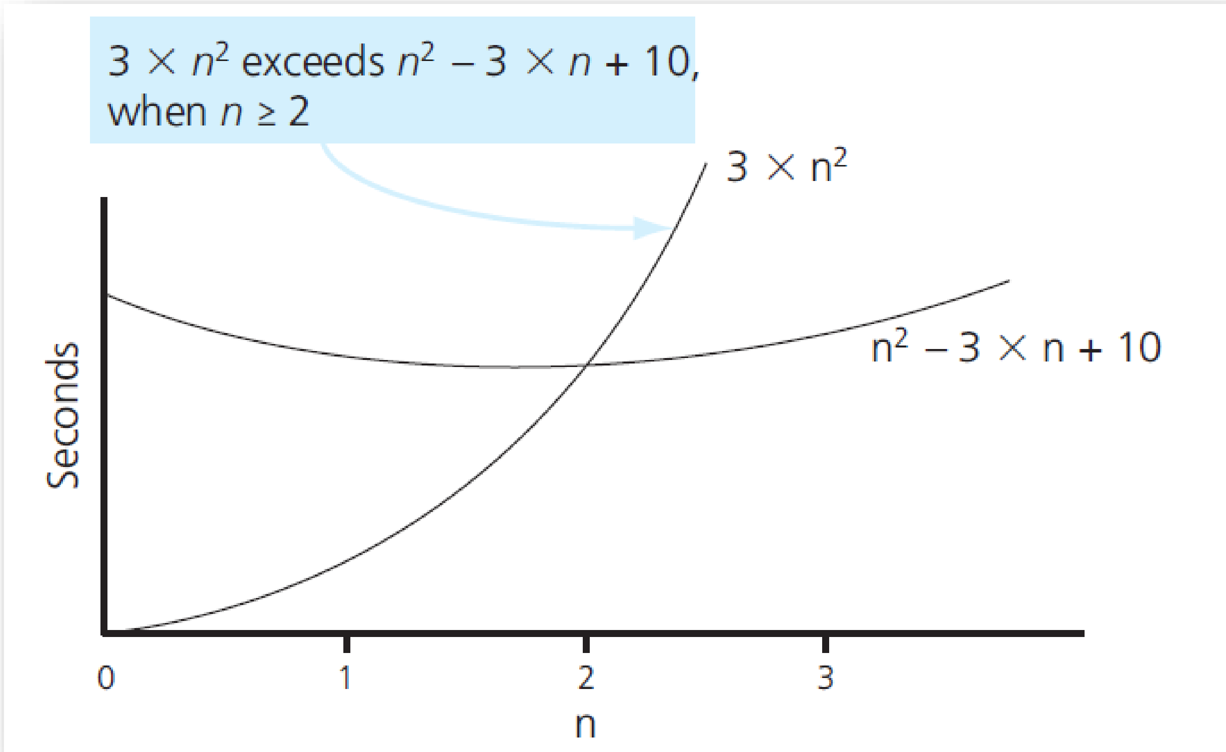


- FIGURE 10-1 Time requirements as a function of problem size n

Analysis and Big-O Notation

- Algorithm *A* is said to be order $f(n)$,
  - Denoted as $O(f(n))$
  - Function f(n) called algorithm's growth rate function
  - Notation with capital O denotes *order*
- Algorithm *A* of order denoted O(f(n))
  - Constants k and n_0 exist such that
  - *A* requires no more than $k \times f(n)$ time units
  - For 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

$$O(1) < O(\log_2 n) < O(n) < O(n \times \log_2 n) <$$

$$n) < O(n^2) < O(n^3) < O(2^n)$$

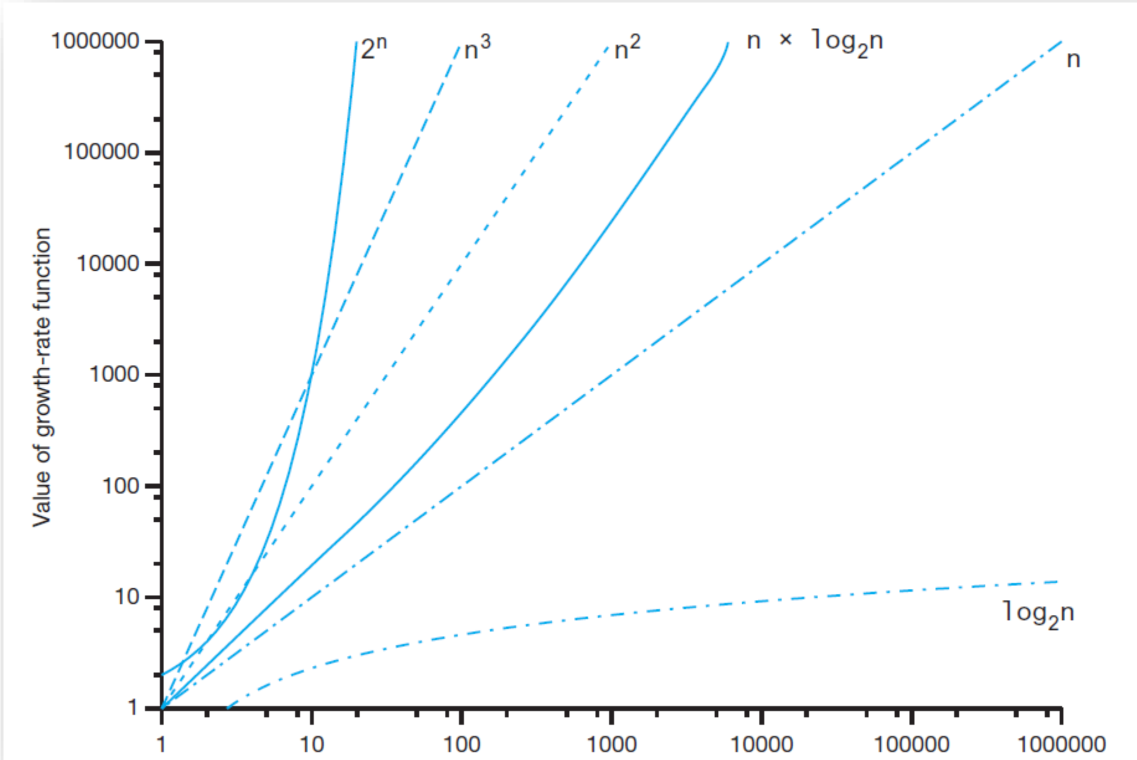
- Order of growth of some common functions

Analysis and Big O Notation

Function	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

Analysis and Big O Notation



- FIGURE 10-4 A comparison of growth-rate functions

Analysis and Big O Notation

- Worst-case analysis
  - Worst case analysis usually considered
  - Easier to calculate, thus more common
- Average-case analysis
  - More difficult to perform
  - Must determine relative probabilities of encountering problems of given size

Keeping Your Perspective

- ADT used makes a difference
  - Array-based `getEntry` is O(1)
  - Link-based `getEntry` is O(n)
- Choosing implementation of ADT
  - Consider how frequently certain operations will occur
  - Seldom used but critical operations must also be efficient

Keeping Your Perspective

- If problem size is always small
  - Possible to ignore algorithm's efficiency
- Weigh trade-offs between
  - Algorithm's time and memory requirements
- Compare algorithms for style and efficiency

Efficiency of Search Algorithms

- Sequential search
  - Worst case: O(n)
  - Average case: O(n)
  - Best case: O(1)
- Binary search
  - O($\log_2 n$) in worst case
  - At same time, maintaining array in sorted order requires overhead cost … can be substantial