Analysis of Algorithms Exercise

- ➪ **Pseudocode**
- ➪ **Problem instance**

Pseudocode = mix of natural language, math, and programming-like expressions.

Quiz
What to change about input to increase run time?

Exercise
Write down 2 **instances** of the search problem that *SequentialSearch* solves.

Algorithm *SequentialSearch*
▸ **Input** Array $A[0..n-1]$, search key K
Output Index of first element of A to match K, otherwise -1

```
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n then return i        ▷ found if i in range
return −1
```

❶ Select suitable input size parameter
❷ Identify a basic operation
❸ Check dependance of basic op
❹ Determine count $C(n)$, somehow
❺ Determine order of growth of $C(n)$

Effciency

➪ Observation

Time efficiency of most algorithms falls into a few categories of (runtime) growth

➪ A classification?

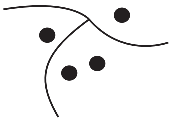
A system for classifying efficiency should avoid dealing individually with efficiency of potentially 1000s of algorithms

cs223fig3.cdr
Thursday, January 27, 2022 12:14:58 PM
Color profile: Disabled

Composite Default screen

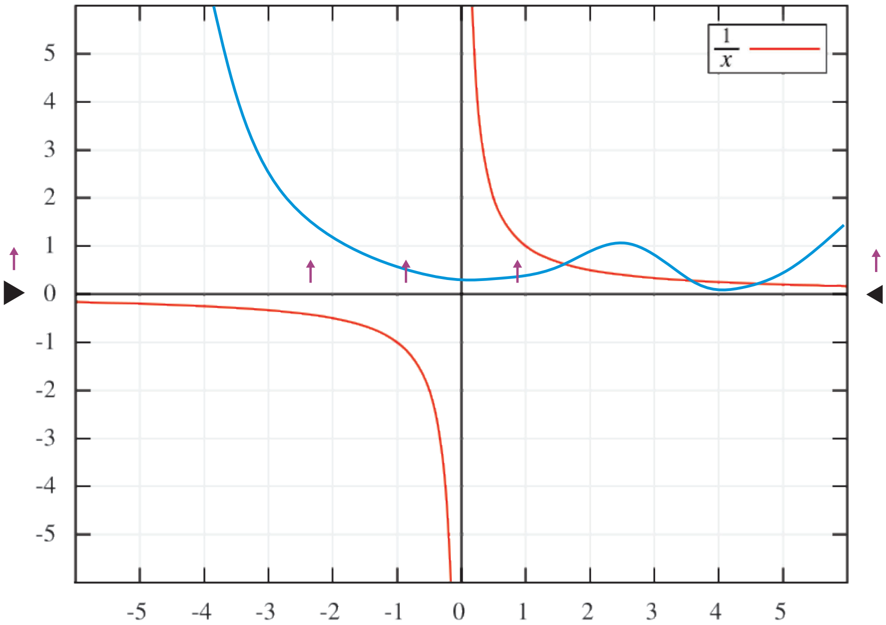
A Useful Tool from Math
Asymptotic Classification

➪ Asymptotes



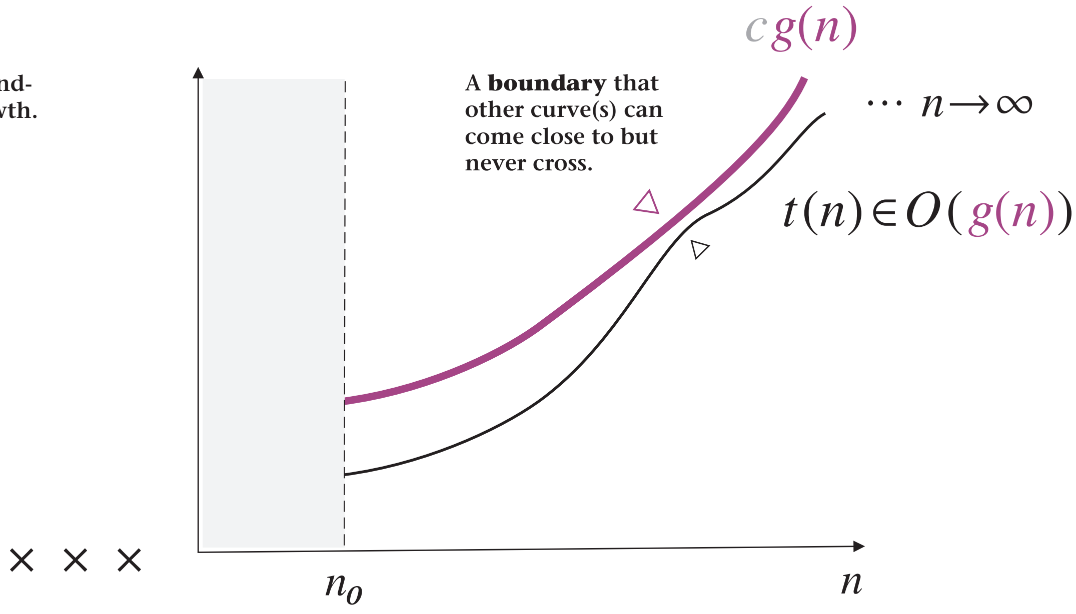
A **boundary** that other curve(s) may come close to but never cross.

Any function can be used to set a boundary, in this case $y = f(x) = ?$

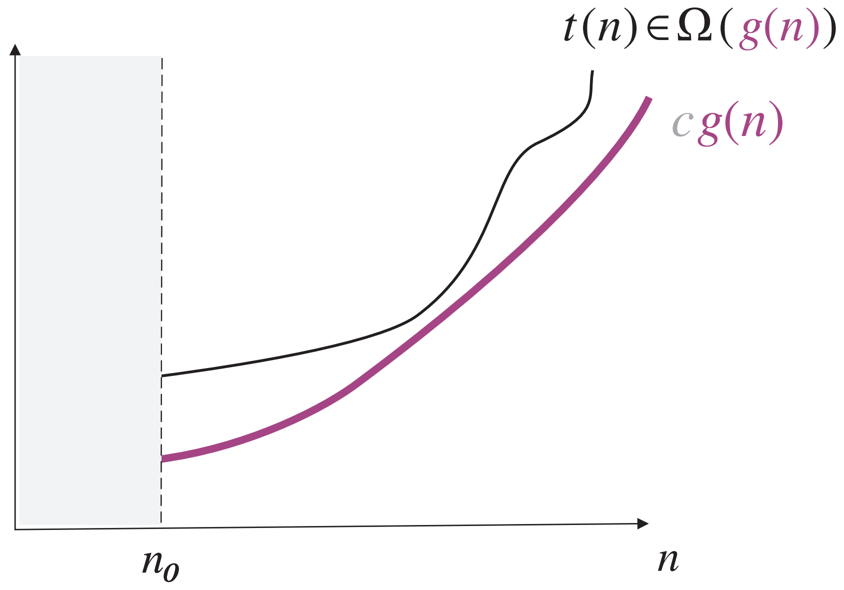


Setting Upper Boundary

Setup an upper boundary on order of growth.



Asymptotic Classification
Setting Lower Boundary



cs223fig3.cdr
Thursday, January 27, 2022 12:14:59 PM
Color profile: Disabled

Composite Default screen

Asymptotic Classification
Θ – Similar Growth

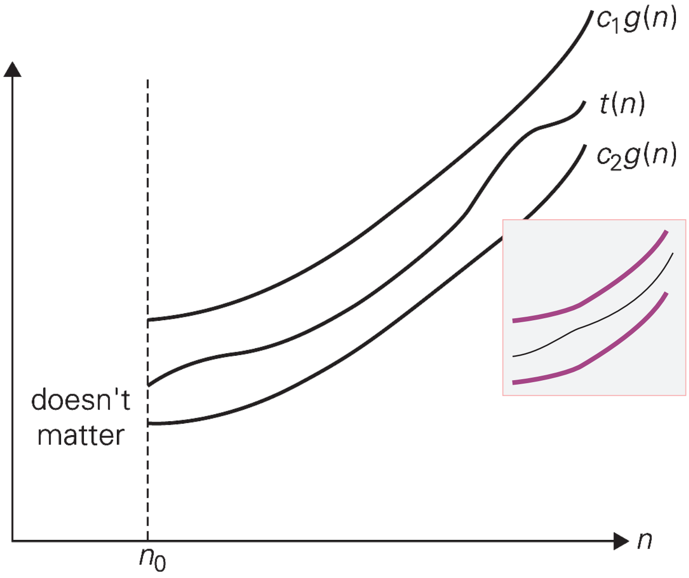
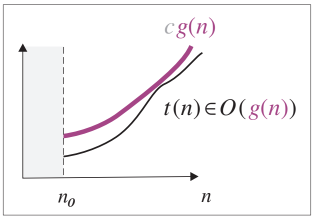


FIGURE 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

cs223fig3.cdr
Thursday, January 27, 2022 12:14:59 PM
Color profile: Disabled

Composite Default screen

Asymptotic Classification Exercise



$$n \in O(n^2)$$

c g(n)

$$100n + 5 \in O(n^2) \qquad \frac{1}{2}n(n-1) \in O(n^2)$$

$$n^3 \notin O(n^2) \qquad 0.00001n^3 \notin O(n^2) \qquad n^4 + n + 1 \notin O(n^2)$$

$$\frac{1}{2}n(n-1) \in \Omega(n^2) \qquad 100n + 5 \notin \Omega(n^2)$$

$$n^3 \in \Omega(n^2)$$

Asymptotic Classification
A Useful Property

8
2
−1
8
12
5
2
−4
8
10

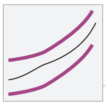
- ➪ **Example: distinct keys in a list**
- ➪ **A solution: overall efficiency?**
- ➪ **Check the theorem**

Another Useful Math Tool
Using Limits

A math limit can investigate relative growth behavior as $n \to \infty$ (which function grows faster).

Which efficiency class of g(n)?

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} \overset{?}{=} \begin{cases} 0 \\ c > 0 \\ \infty \end{cases}$$



n	$t(n)$? $\frac{1}{4}n^2 + 5$	$g(n)$ n^2	$\lim_{n\to\infty} \frac{t(n)}{g(n)}$
1	5.25	1.00	5.25000
2	6.00	4.00	1.50000
3	7.25	9.00	0.80556
4	9.00	16.00	0.56250
5	11.25	25.00	0.45000
6	14.00	36.00	0.38889
7	17.25	49.00	0.35204
8	21.00	64.00	0.32813
…			
97	2357.25	9409.00	0.25053
98	2406.00	9604.00	0.25052
99	2455.25	9801.00	0.25051
100	2505.00	10000.00	0.25050
…			
447	49957.25	199809.00	0.25003
448	50181.00	200704.00	0.25002
449	50405.25	201601.00	0.25002
1000	250005.00	1000000.00	0.25001
…			

Exercise
Use Excel to mathematically compare growth of functions from the Exercise slide. For example: the truth of $n^3 \in \Omega(n^2)$.

Algorithm Efficiency
Basic Classes

Asymptotic efficiency

Using long-term (limiting) runtime behavior to classify efficiency as inputs become increasingly larger.

TABLE 2.2 Basic asymptotic efficiency classes

Class	Name	Comments
1	*constant*	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	*logarithmic*	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 5.5). Note that a logarithmic algorithm cannot take into account all its input (or even a fixed fraction of it): any algorithm that does so will have at least linear running time.
n	*linear*	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.
$n \log n$	"*n-log-n*"	Many divide-and-conquer algorithms (see Chapter 4), including mergesort and quicksort in the average case, fall into this category.
n^2	*quadratic*	Typically, characterizes efficier two embedded loops (see the tary sorting algorithms and ce n-by-n matrices are standard e
n^3	*cubic*	Typically, characterizes efficier three embedded loops (see the nontrivial algorithms from line class.

2^n	*exponential*	Typical for algorithn n-element set. Ofter in a broader sense to growth as well.
$n!$	*factorial*	Typical for algorithn of an n-element set.