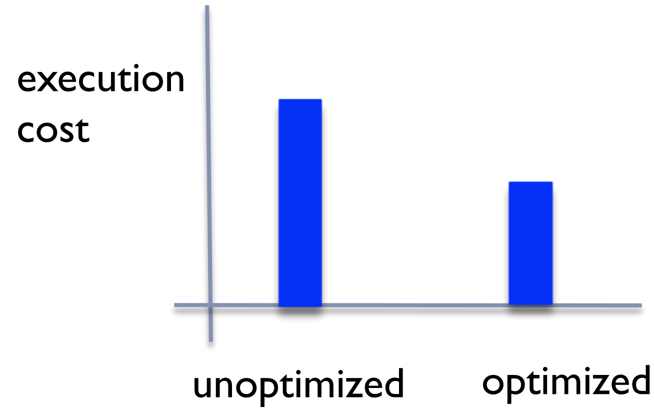
CS Teaching Academy

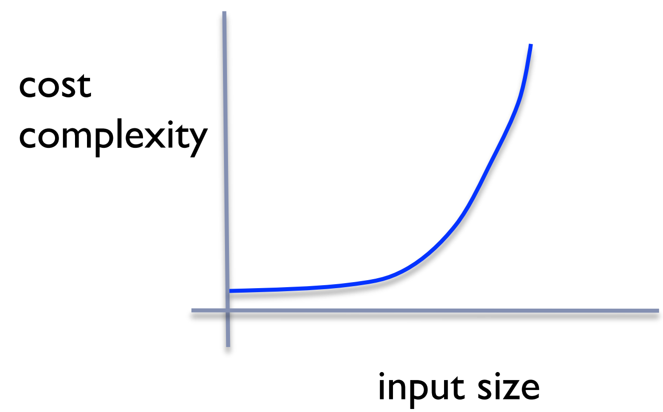
Chapter 5, Module 2: Efficiency & Correctness

Program Efficiency Perspectives

- Program level:
  - reduce execution cost by optimizing algorithms and data
  - compilers do this automatically



- Algorithmic level:
  - choose algorithms based on their cost profiles, also known as *cost complexity*



Cost Complexity of an Algorithm

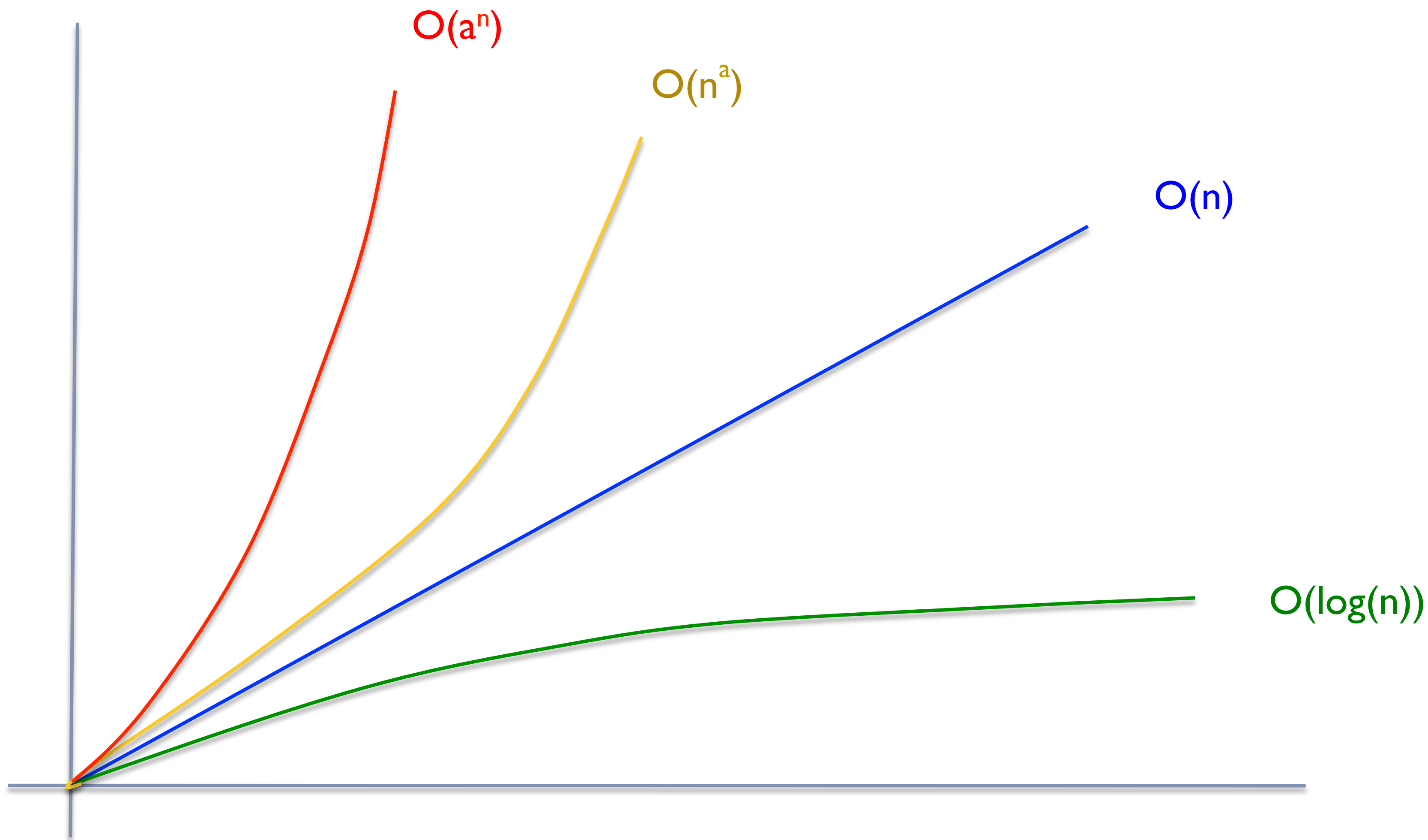
- Growth in cost with respect to problem size

Algorithm	Size=10	Size=100	Size=1000	Cost
A	20	200	2000	2n
B	100	10000	1000000	n^2
C	4	7	10	log2(n)

Operations Required

- Cost = number of operations required, average or worst case
- Independent of computer hardware
- Determined based on algorithm properties only

Common Cost Complexity Classes



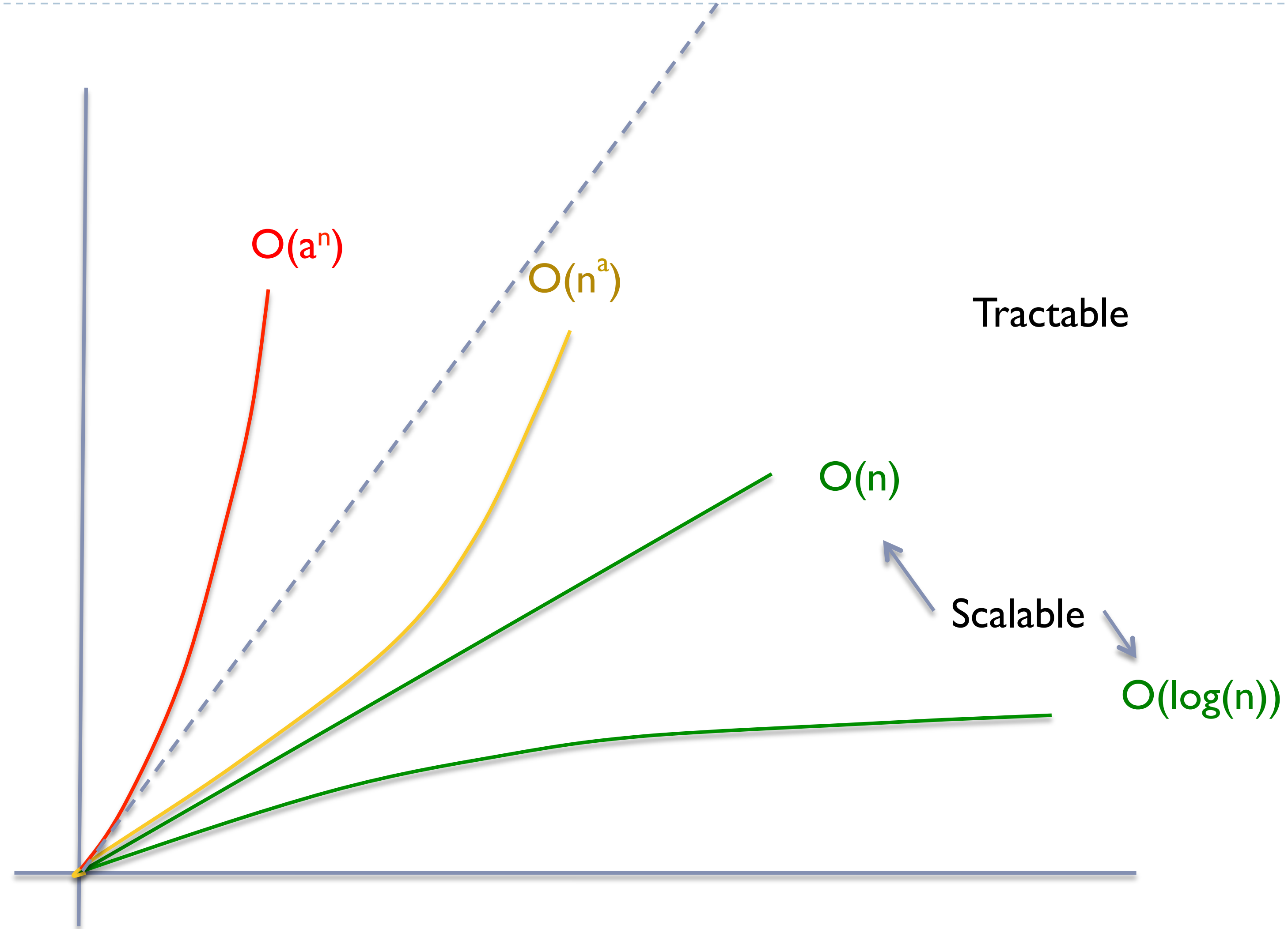
Scalability & Tractability

- A scalable algorithm has a cost with linear growth or less
- An intractible algorithm is too expensive to execute for practical problem sizes

Cost	N=10	N=100	N=1000	N=10,000	N=100,000
log2(n)	3us	6.7us	10us	13ms	17ms
n	10us	100us	1.0ms	10ms	100ms
n^2	100us	10ms	1s	1.7m	2.8h
2^n	1ms	4.0×10^{16}y			

Processing Time, assuming 10^6 operations / second

Scalability & Tractability



Example: Sequential Search

Algorithm	Worst-Case Count*
`int search(int[] source, key) {`	1
`  for (int i=0; i<source.length; i++) {`	1+n+n
`    if (key == source[i]) {`	n
`      return i;`	1
`    }`	
`  return -1;`	1
`}`	

Worst-case cost = (4 + 3n), O(n)

* as a function of n = size of the array

Example: Binary Search

```
             Algorithm                                         Worst-Case
                                                               Count*

int search(int[] source, int key, int from, int to) {         log2(n)
  if (from > to) {                                            log2(n)
    return -1;                                                1
  }
  int middle = (from + to) / 2;                               log2(n)
  if (key == source[middle]) {                                log2(n)
    return middle;                                            1
  }
  if (key < source[middle]) {                                 log2(n)
    return search(source, key, from, middle-1);               log2(n)
  }
  return search(source, key, middle+1, to);                   log2(n)
}
```

```
Worst-case cost = (2 + 7log2(n)), O(log(n))
```

Some Well-Known Sort Algorithms

Algorithm	Worst-Case Time Cost
Bubble Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Merge Sort	$O(n \log(n))$
Heap Sort	$O(n \log(n))$

Summary

- Program efficiency can be examined
  - with respect to implementation (execution cost)
  - with respect to the algorithm only (complexity)

- Common complexity classes include
  - Scalable: log(n) and n
  - Tractable: n^2 (and below, i.e., scalable)
  - Intractable: 2^n (and above)