Lecture 22: Complexity

CSCI 101

Spring 2018

Today

- Announcements
  - Next week: Course response forms. Bring computer.
  - Final exam: self-scheduled; 2 sheets of notes allowed

- Computational Complexity
  - Big-O notation to describe # of operations
  - Example: complexity of search algorithms

Time and space

Algorithm choice will determine the resources used at runtime.

Key resources:

- Time (CPU time)
- Space (memory)

Computational Complexity

A *basic operation* requires one time unit

- adding two values
- assigning to a variable
- comparing two values
- accessing a list element

How many *basic operations* does a given algorithm perform?

Traversing a List

Code to print all values in a list:

```
n = len(t)
i = 0
while i < n:
    print(t[i],end='')
    i += 1
```

How many operations does this perform?

→ $c \cdot n$ operations for list of n elements

a constant, eg, 5

Big-O notation

- Use a function to describe number of basic operations in terms of input size
- The function includes only the dominant terms, ignoring constants
- Example: list traversal is O(n) for a list of n values

Linear Search

Find a value in a list:

```
n = len(t)
i = 0
while i < n:
    if t[i] == target:
        return i
    i += 1
return -1
```

Number of operations for a list of *n* elements: O(*n*)

Binary Search

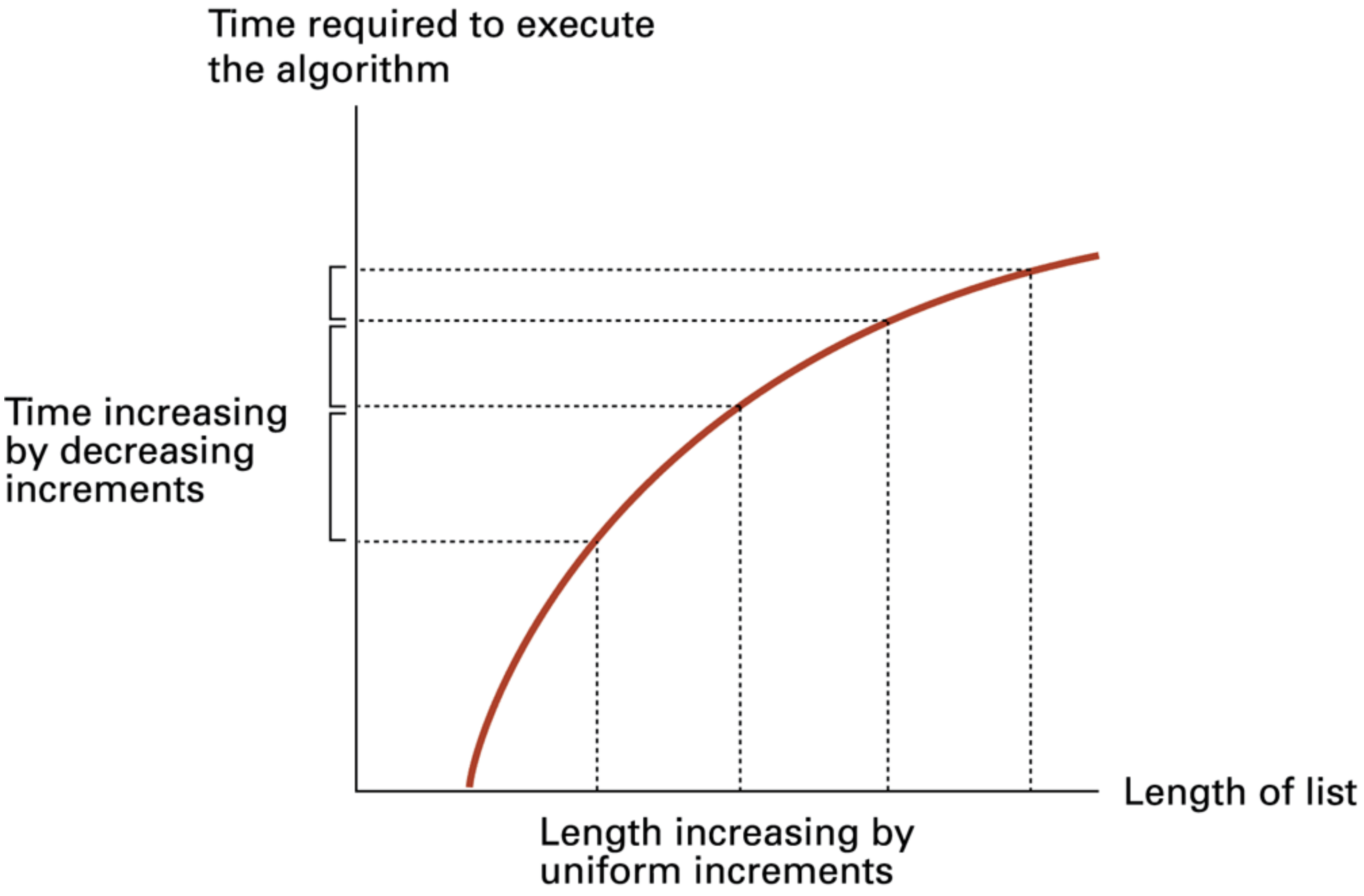
We can search faster if the list is sorted

⟶ Compare middle element to the target, then refine search to one half of list

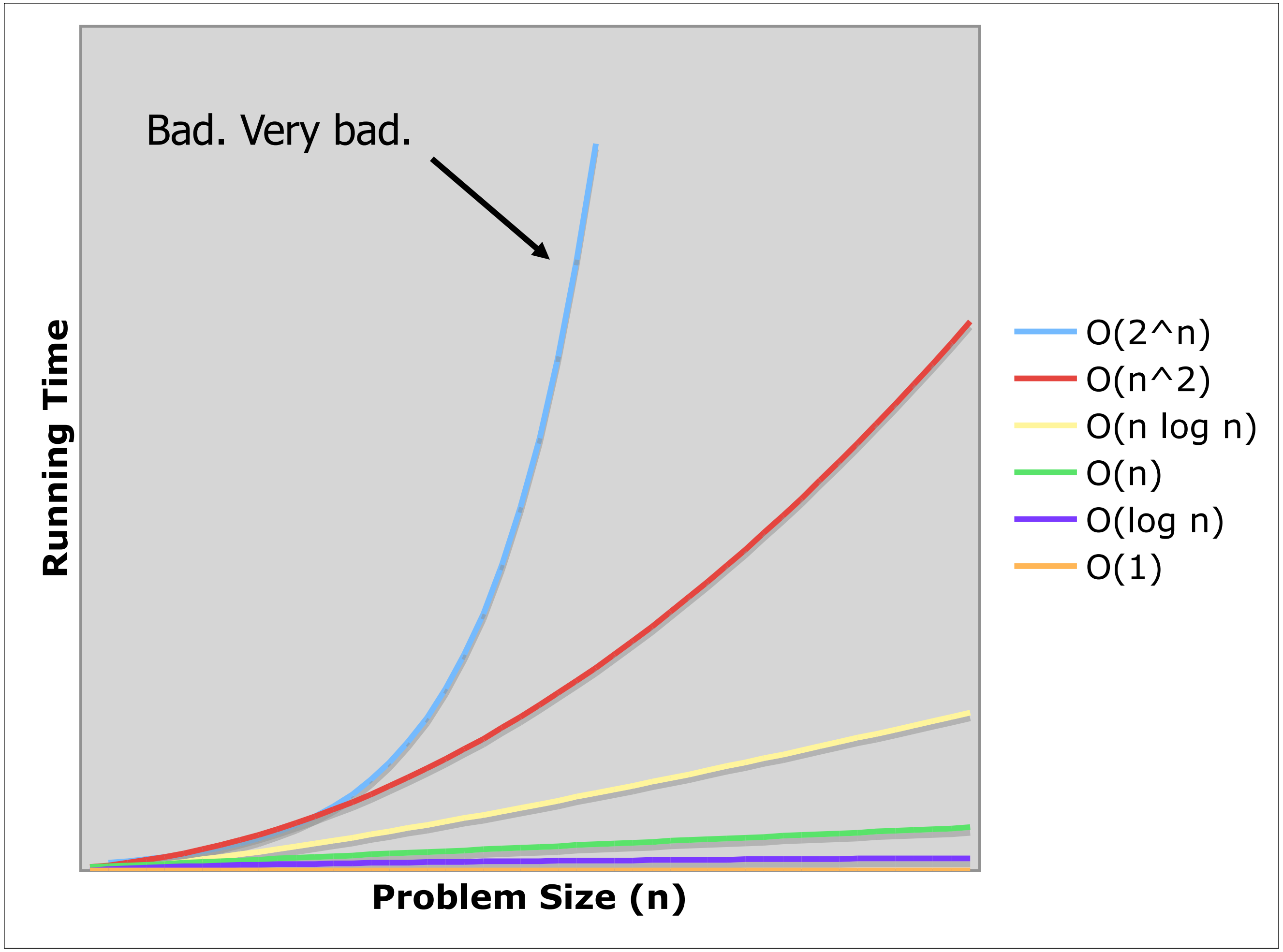
2	5	8	11	15	16	21	24	29	41	45	58	71	85	92	95

Number of operations for a list of *n* elements: $O(\log_2 n)$ or $O(\log n)$

Binary Search: O(log n)



Order of Growth



Algorithmic Analysis

If time needed…	then we say…	Examples
grows proportionally with the input size	the algorithm runs in O(n) or linear time	Linear search Compute sum of list
grows only incrementally as the input size doubles	the algorithm runs in O(log n) or logarithmic time	Binary search, Fast exponentiation
doubles with a unit increment to the input size	the algorithm runs in $O(2^n)$ or exponential time	Recursive Fibonacci Towers of Hanoi
doesn't change with the input size	the algorithm runs in O(1) or constant time	Finding max of sorted list
grows quadratically with the input size	the algorithm runs in $O(n^2)$ or quadratic time	Insertion, selection, and bubble sorts

Algorithms We Prefer

Polynomial time algorithms are desirable

- $O(1)$ [constant]
- $O(\log n)$ [logarithmic]
- $O(n)$ [linear]
- $O(n \log n)$
- $O(n^2)$ [quadratic]
- $O(n^3)$ [cubic]

Non-polynomial time algorithms are undesirable

- $O(2^n)$ [exponential]
- $O(n!)$ [factorial]