Robustness and speed

Prof. Noah Snavely
CS1114
http://www.cs.cornell.edu/courses/cs1114

Administrivia

- Assignment 1 is due next Friday by 5pm
  - Lots of TA time available (check the web)

- For grading, please get checked off by a TA, or sign up for a demo slot
  - These will be posted to CMS soon
Finding the lightstick center

- Last time: two approaches

- Both have problems...

How can we do better?

- What is the average weight of the people in this kindergarten class photo?

12 kids, avg. weight = 40 lbs
1 Arnold, weight = 236 lbs

Mean: \( \frac{12 \times 40 + 236}{13} = 55 \text{ lbs} \)
How can we do better?

- Idea: remove maximum value, compute average of the rest

\[
\text{Mean: } \frac{12 \times 40 + 236}{13} = 55 \text{ lbs}
\]

5% Trimmed mean

% A is a vector of length 100
for i = 1:5
    % 1. Find the maximum element of A
    % 2. Remove it
end

- Is it correct?
- Is it fast?
- Is it \textit{the fastest} way?
How do we define fast?

- We want to think about this issue in a way that doesn’t depend on either:
  - A. Getting really lucky input
  - B. Happening to have really fast hardware

How fast is our algorithm?

- An elegant answer exists
- You will learn it in later CS courses
  - But I’m going to steal their thunder and explain the basic idea to you here
  - It’s called “big-O notation”

- Two basic principles:
  - Think about the average / worst case
    - Don’t depend on luck
  - Think in a hardware-independent way
    - Don’t depend on Intel!
Simple example: finding the max

```matlab
function m = find_max(A)
% Find the maximum element of an array A
m = A(1);
n = length(A);
for i = 2:n
    if (A(i) > m)
        m = A(i);
    end
end
```

- How much work is this?

Big-O Notation

```matlab
function m = find_max(A)
% Find the maximum element of an array A
m = -1;
for i = 1:length(A)
    if (A(i) > m)
        m = A(i);
    end
end
```

- Let’s call the length of the array \( n \)
- The amount of work grows in proportion to \( n \)
- We say that this algorithm runs in time \( O(n) \)
- (Or that it is a \textit{linear-time} algorithm)
Another version of the trimmed mean

- Given an array of \( n \) numbers, find the \( k^{th} \) largest number in the array

Strategy:
- 1. Find the biggest number in the array
- 2. Remove it
   - Repeat \( k \) times
   - The answer is the last number you found

Performance of our algorithm

- Given an array of \( n \) numbers, find the \( k^{th} \) largest number in the array

Strategy:
- 1. Find the biggest number in the array
- 2. Remove it
   - Repeat \( k \) times
   - The answer is the last number you found

- How many operations do we need to do, in terms of \( k \) and \( n \)?
Performance of our algorithm

- How much work will we do?

  1. Examine $n$ elements to find the biggest
  2. Examine $n-1$ elements to find the biggest

  ... keep going ...

  k. Examine $n-(k-1)$ elements to find the biggest

Performance of our algorithm

- What value of $k$ is the worst case?
  - $k = n$ we can easily fix this
  - $k = n/2$

- How much work will we do in the worst case?
  1. Examine $n$ elements to find the biggest
  2. Examine $n-1$ elements to find the biggest

  ... keep going ...

  n/2. Examine $n/2$ elements to find the biggest
How much work is this?

- How many elements will we examine in total?
  $$n + (n - 1) + (n - 2) + ... + n/2$$
  $$n \text{ / 2 terms}$$
  $$= ?$$

- We don’t really care about the exact answer
  - It’s bigger than $$(n / 2)^2$$ and smaller than $$n^2$$

How much work in the worst case?

- The amount of work grows in proportion to $$n^2$$

- We say that this algorithm is $$O(n^2)$$
How much work is this?

- The amount of work grows *in proportion* to $n^2$
- We say that this algorithm is $O(n^2)$
- If it takes 10 seconds when $n = 1,000$, how long will it take when $n = 2,000$?
  - A: 20 seconds
  - B: 40 seconds

Classes of algorithm speed

- Constant time algorithms, $O(1)$
  - Do not depend on the input size
  - Example: find the first element

- Linear time algorithms, $O(n)$
  - Constant amount of work for every input item
  - Example: find the largest element

- Quadratic time algorithms, $O(n^2)$
  - Linear amount of work for every input item
  - Example: slow median method
Asymptotic analysis picture

- Different hardware only affects the parameters (i.e., line slope)
- As \( n \) gets big, the “dumber” algorithm by this measure always loses eventually

Where we are so far

- Finding the lightstick
  - Attempt 1: Bounding box (not so great)
  - Attempt 2: Centroid isn’t much better
  - Attempt 3: Trimmed mean
    - Seems promising
    - But how do we compute it quickly?
    - The obvious way doesn’t seem so great...
Questions?