

The Case for Algorithms

or "Computer Science is Not a Cookbook"

Michael Coblenz

Physics lasts > 80 years

(1925: relativity)

A car lasts 16 years.

(source: California Air Resources Board)

Canned food lasts > 2
years.

Our world...



A story

How can we beat tuna?

- Don't teach particular technologies.
- Teach how to **think**.
- Teach how to **solve problems**.
- But: in the context of current technologies.

Algorithm

- Procedure for solving a computational problem
 - sorting arrays
 - compressing data
 - rendering **The Incredibles**
 - encrypting sensitive information

Basis

- Algorithms are the basis of everything!
 - Fish behavior
 - Mail delivery
 - Human genome
 - Power distribution
 - Airline scheduling

Not dependent on...

- Human and computer language
- Hardware (mostly)
- Current trends

Current applications

- Genetics
- Web searching
- Graphics
- Databases

A standard approach

1. Define a problem
2. Give an algorithm
3. Have students implement it
4. Wash, rinse, repeat

For example

- How do you implement a stack?
- Array
 - space for n elements
 - Keep track of where the top is
 - When full, must allocate a new array & copy

What's missing?

- Why the algorithm works (prove it!)
- How fast it is (prove it!)
- A chance to think

Stacks

- push and pop: $O(1)$
- BUT ONLY IF....something.
- What's missing?

When it's full...

1. Allocate a new array, of size $f(n)$. Cost: $O(1)$
2. Copy old elements. Cost: $O(n)$
 - WARNING: this is expensive!

When it's too empty...

1. Want to save space.
2. Allocate a new array, of size $g(n)$. Cost: $O(1)$
3. Copy old elements. Cost: $O(n)$
 - WARNING: this is expensive!

A first try: $f(n) = n + 1$

- Each full insertion costs $O(n)$!
- Is this okay?

Idea

- $f(n) = 2n$
- $g(n) = n/2$
- i.e.
 - double the size when expanding
 - halve the size when contracting
- But how do we know if this works?

Amortized Analysis

- Want to calculate cost per operation
 - averaged over many operations
- Some expensive operations OK
- Want: $O(1)$ amortized cost

Accounting

- Each array operation costs \$1.
- Goal: pay constant amount for each push/pop.
- Leftover money goes in the bank to pay for future expensive operations.
- Need to figure out how much each push/pop costs.

Idea

- Every push: pay an extra \$1. Save for copying the element.
- Start half full.



One more dollar



Enough to pay for copying!

Pop

👁️ Only \$1 extra. But wait until it's 1/4 full.



○
○
○



\$ \$ \$ \$

start full

start after expansion

So THAT'S why it works!

- Proof reveals explanation for mysterious 2
- How about 3? 10?

Another Example

- Quicksort was invented in 1962.
- $O(n \log n)$
- Fastest known comparison-based sorting algorithm.
- $O(n \log n)$ seems slow. Maybe someone smart can do better.

How hard is comparison-based sorting?

- "I haven't thought of one faster than $O(n \log n)$, so there must not be a way to do it."
- WRONG APPROACH!
- Your creativity is limited.

• Need



?

Lower bounds

- Start simple.
- Is sorting $\Omega(1)$?
 - Yes. You can't answer without looking at the input at all! (why?)
- Is sorting $\Omega(n)$?
 - Yes. You need at least time n to write down the answer!

"at least as slow as..."

The rules

- Assume all elements are distinct.
- You can only compare **pairs** of elements.
- You get "greater" or "less".
- Algorithm must work for ALL arrays with distinct elements.

An Observation

- There are $n!$ permutations of n elements.
- Goal: determine which permutation you were given. (why?)
- How many operations does this take?

$n!$ permutations

- Compare 2 elements. Throw out permutations that get this pair wrong. Repeat.
- How many do we throw out each time?
 - At most half!
- Game of 20 questions

Throw out half...

- Takes $\log(n!)$ time!
- $\log(n!) < \log(n^n)$
 - $\log(n^n) = n \log n$
- Stirling's approximation: $\log(n!) \approx n \log n$

Lower bound!

- It is impossible to do better than $O(n \log n)$.
 - Not even if you're really clever!
 - Not even if we also have warp drive!
 - Not even on a Pentium 42!
- How do we do better than $O(n \log n)$?

Median

- unsorted n element array
- Find median. How slow is this?

Approaches

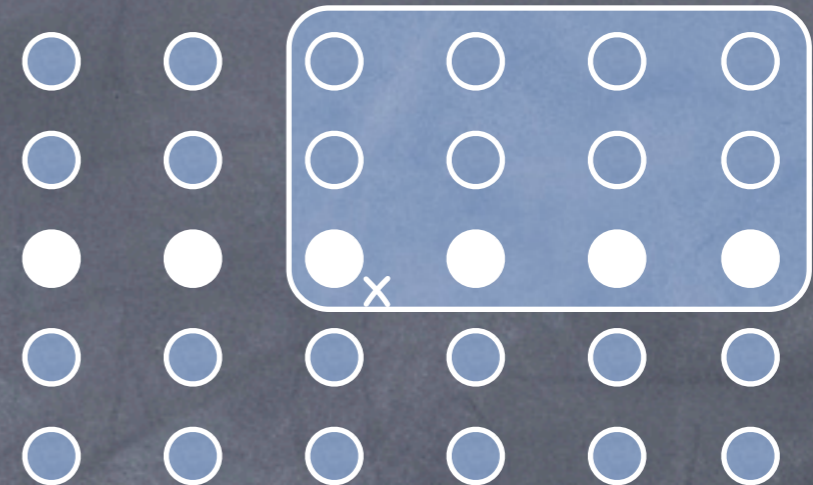
- Sort, then pick middle. Cost: $O(n \log n)$.
- Is there a better way?
- What's a lower bound?
- You have to at least look at all the elements.
 - $O(n)$

$O(n)$?!

- Yes.
- Deterministic.
- Worst-case.

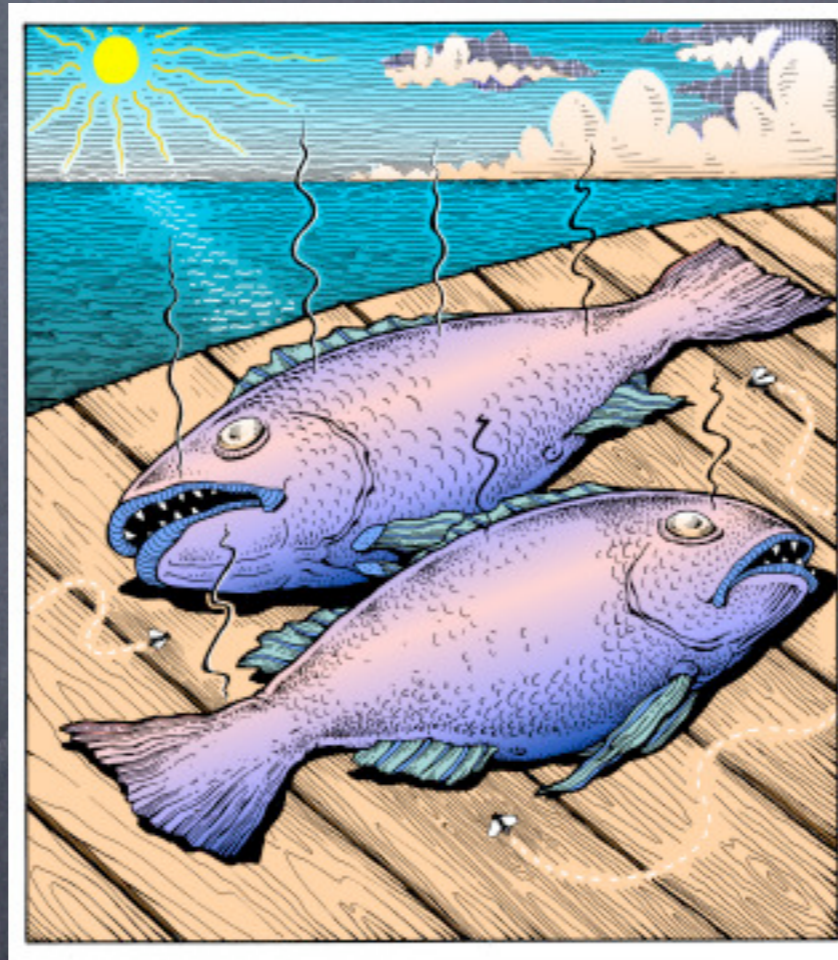
Algorithm: SELECT

at least as big as x



1. Divide into $\lceil n/5 \rceil$ groups.
2. Find median of each group.
3. Recursively find median x of these medians.
4. Partition around x .
5. Use SELECT recursively on correct partition.

$O(n)$?



Analysis

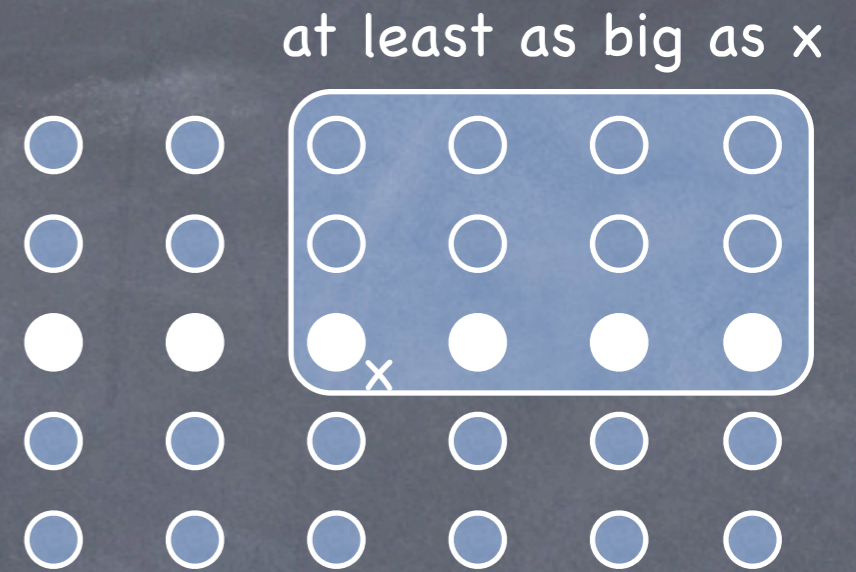
Let $T(n)$ be the cost of SELECT on n elements.

1. Divide into $\lceil n/5 \rceil$ groups. $O(n)$
2. Find median of each group.
 - Constant time per group. $O(n)$ total.
3. Recursively find median x of these medians. $T(\lceil n/5 \rceil)$
4. Partition around x . $O(n)$
5. Use SELECT recursively on correct partition.

note: SELECT can find i th smallest, not just the median

How big might it be?

How big?



- How many are bigger than median of medians x ?
- Half the medians are bigger than x .
assume n is a multiple of 5
 - Each of these groups has 3 elements bigger than x .
- $3 \left(\frac{n/5}{2} - 1 \right) = \frac{3n}{10} - 3$
- At least $\frac{3n}{10} - 3$ are bigger than x !

Analysis

- If $3n/10 - 3$ are bigger, then worst case:
 - $7n/10 + 3$ elements in biggest partition
- $T(n) \leq T(n/5) + T(7n/10 + 3) + O(n)$
- Want $T(n) \leq O(n)$. Assume true for previous iteration.
- $T(n) \leq cn/5 + 7cn/10 + 3c + an$
 - $= 9cn/10 + 3c + an$
 - $= cn + (-cn/10 + 3c + an)$
 - iff $cn/10 > 3c + an$
 - $c > 10an/(n-30)$. Use $n > 30$. $n \leq 30$: $O(1)$

So what?

- Algorithm analysis gives insight
 - but tools for analysis are forever
- power to know when you're right
- Explains details of algorithms
- Allows you to develop algorithms that **work**
- But can pick more exciting examples

Questions

- (cake-cutting?)