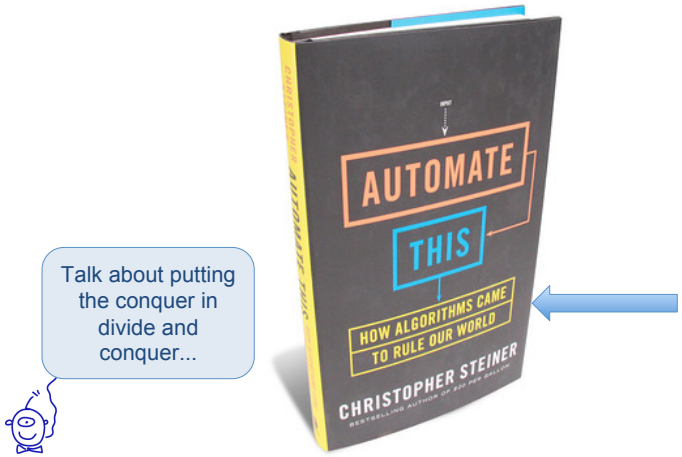
Algorithms Rule the World !?



Last time…

- Divide-and-conquer:
  - Merge sort
  - In pop culture?

- Administrivia

Recall D&C

- **Divide** the problem into a number of smaller, sub-instances of the same problem
- **Conquer** the subproblems by solving them recursively
- **Combine** the subproblem solutions to give a solution to the original problem

Wouldn't it be nice if we had a mechanism for evaluating the complexity of all such algorithms that fit into this paradigm?

Recurrence functions

- A recurrence is a function that is defined in terms of
  - One or more base cases, and
  - Itself, with smaller arguments.
- Examples:

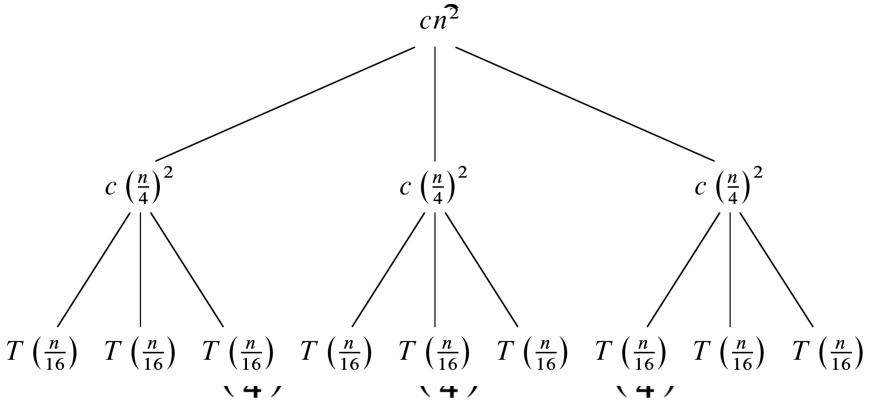
$$T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ 2T(n/2) + n & \text{if } n \geq 1\,. \end{cases} \qquad T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ T(n-1) + 1 & \text{if } n > 1\,. \end{cases}$$

Solution: $T(n) = n \lg n + n$.

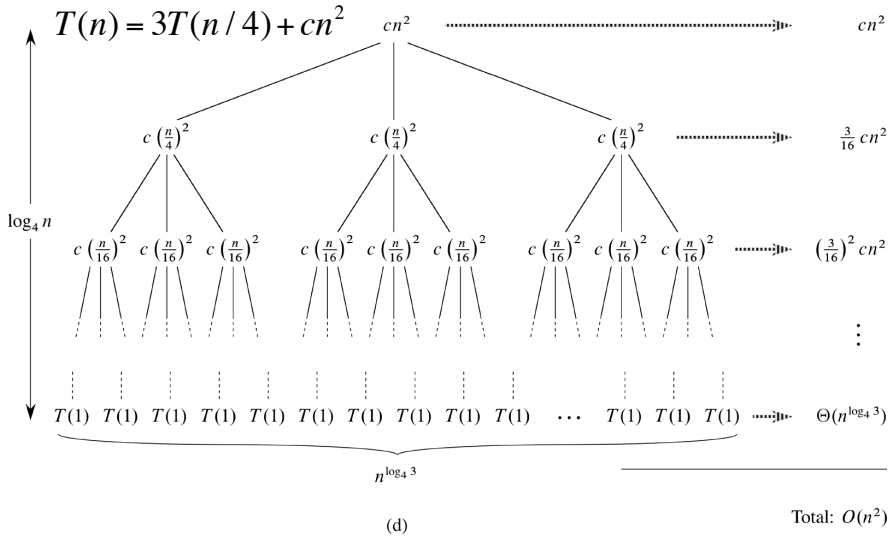
$$T(n) = \begin{cases} 0 & \text{if } n = 2\,, \\ T(\sqrt{n}) + 1 & \text{if } n > 2\,. \end{cases}$$

Recursion-tree

$$T(n) = 3T(n/4) + cn^2$$



Recursion-tree



Worksheet: Solve using recursion tree

$$T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ T(n-1) + 1 & \text{if } n > 1\,. \end{cases}$$

Solution: $T(n) = n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 2\,, \\ T(\sqrt{n}) + 1 & \text{if } n > 2\,. \end{cases}$$

Solution: $T(n) = \lg \lg n$.

Substitution Method

- You can inductively verify / prove (a guess for) a solution by substituting it back into the recurrence formula.
- Recursion trees are a good way of generating a good estimate.

$$T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ 2T(n/2) + n & \text{if } n > 1\,. \end{cases}$$

Substitution Method for Asymptotics

- Assume $T(n) = O(1)$ for sufficiently small *n*
  - Why?
  - No need to worry about base cases
- Name the constant in the additive term
- For Θ, must show upper (O) and lower bounds (Ω) separately!
  - Can use different constants

Example: Merge sort

$$T(n) = 2T(n/2) + \Theta(n)$$

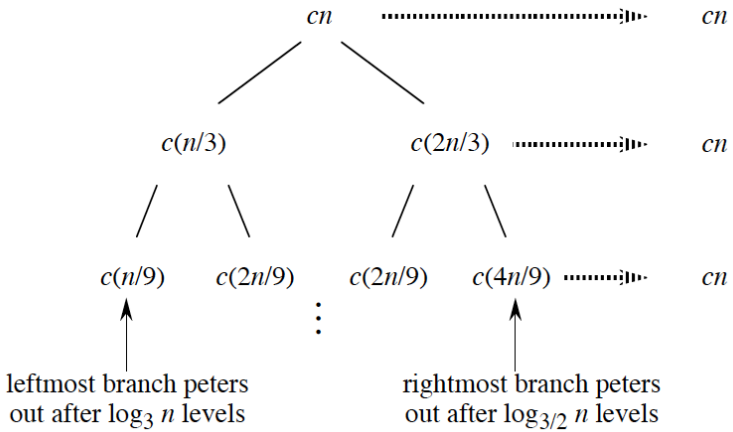
$$2T(n/2) + O(n)$$

$T(n) \le 2T(n/2) + cn$ for some positive constant c

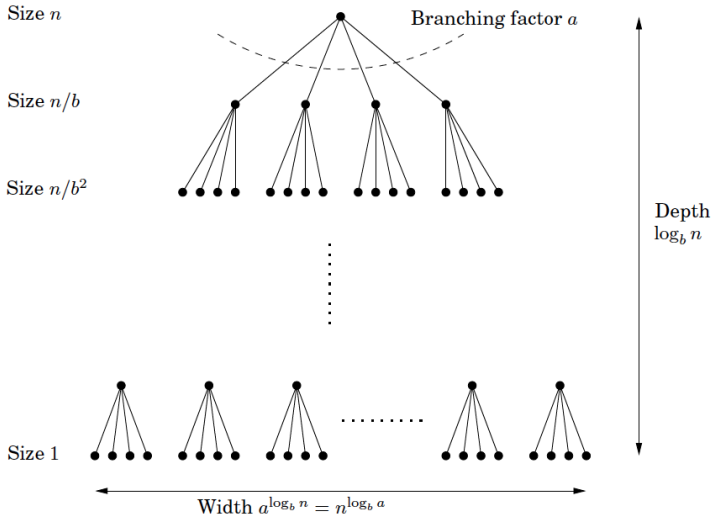
Guess: $T(n) \le dn \lg n$ for some positive constant d.

When to substitute

- Recursion trees can get messy…

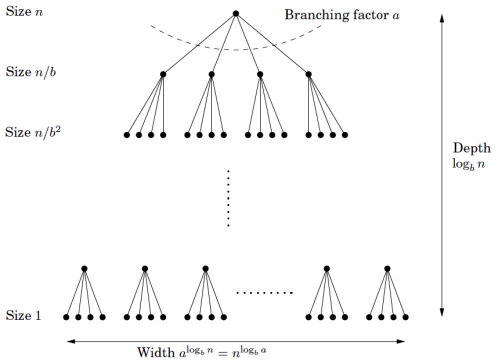


The structure of D&C problems



A General Recurrence Function

$$T(n) = aT(\lceil n/b \rceil) + O(n^d)$$



Master theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ *for some constants* $a > 0$, $b > 1$, *and* $d \geq 0$,

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \,. \end{cases}$$

Full Proof: Section 4.6

Proving the Master Theorem

- For convenience, assume n is a power of b
- Note: size of problem decreases by a factor of *b* with each level of recursion, reaches base case after $\log_b n$ levels
- Branching factor *a*
- k^{th} level of the tree has a^k subproblems of size n/b^k

Proving the Master Theorem (cont)

- k^{th} level of the tree has a^k subproblems of size n/b^k:

$$a^k \times O\left(\frac{n}{b^k}\right)^d = O\left(n^d\right) \times \left(\frac{a}{b^d}\right)^k$$

- As *k* increases, geometric series with ratio $\left(\frac{a}{b^d}\right)$

Proving the Master Theorem: Case 1:

$$O\left(n^d\right) \times \left(\frac{a}{b^d}\right)^k$$

$\left(\frac{a}{b^d}\right) < 1$ --- series is decreasing, sum given by **first** term / level in the tree:

$$O\left(n^d\right)$$

Proving the Master Theorem: Case 2:

$$O\left(n^d\right) \times \left(\frac{a}{b^d}\right)^k$$

$\left(\frac{a}{b^d}\right) > 1$ --- series is increasing, sum given by **last** term / level in the tree:

$$n^d \left(\frac{a}{b^d}\right)^{\log_b n} = n^d \left(\frac{a^{\log_b n}}{\left(b^{\log_b n}\right)^d}\right) = a^{\log_b n} = a^{(\log_a n)(\log_b n)} = n^{\log_b a}$$

Proving the Master Theorem: Case 3:

$$O\left(n^d\right) \times \left(\frac{a}{b^d}\right)^k$$

$\left(\frac{a}{b^d}\right) = 1$ --- each of the $O\left(\log_b n\right)$ terms in the series are equal to

$$O\left(n^d\right)$$

Master theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ *for some constants* $a > 0$, $b > 1$, *and* $d \geq 0$,

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \,. \end{cases}$$

Full Proof: Section 4.6

Integer Multiplication

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$

$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R.$$

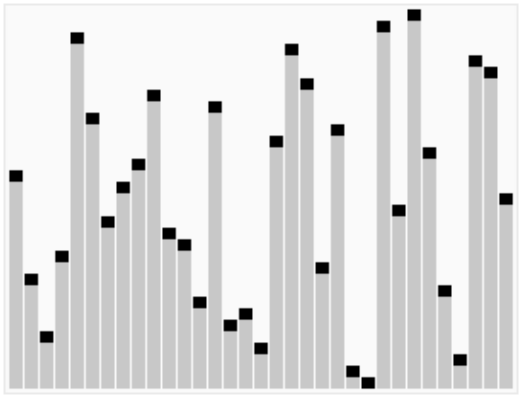
$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R.$$

$$T(n) = 4T(n/2) + O(n). \longrightarrow O(n^2)$$

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

$$T(n) = 3T(n/2) + O(n). \longrightarrow O(n^{\log_2 3})$$

Quicksort



Source: Wikimedia

Quicksort: the players

Tony Hoare

From Wikipedia, the free encyclopedia

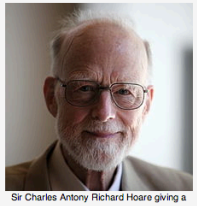
Sir Charles Antony Richard Hoare (born 11 January 1934),[1] commonly known as **Tony Hoare** or **C. A. R. Hoare**, is a British computer scientist. He developed the sorting algorithm Quicksort in 1960. He also developed Hoare logic for verifying program correctness, and the formal language Communicating Sequential Processes (CSP) to specify the interactions of concurrent processes (including the dining philosophers problem) and the inspiration for the occam programming language.

Contents [hide]
1 Biography
2 Quotations
3 Awards
4 Books
5 References
6 Further reading
7 External links

Biography [edit source | edit beta]

Born in Colombo, Ceylon (now Sri Lanka) to British parents, he received his Bachelor's degree in Classics from the University of Oxford (Merton College) in 1956. He remained an extra year at Oxford studying graduate-level statistics, and following his National Service in the Royal Navy (1956–1958). While he studied Russian, he also studied computer translation of human languages at the Moscow State University in the Soviet Union in the school

Sir Charles Antony Richard Hoare



Sir Charles Antony Richard Hoare giving a conference at EPFL on 20 June 2011

Born	11 January 1934 (age 79) Colombo, British Ceylon
Residence	Cambridge
Fields	Computer Scientist
Institutions	Elliott Brothers Queen's University Belfast Oxford University

Quicksort

- Java system sort = Mergesort

- Unix sort command = Quicksort

Quicksort

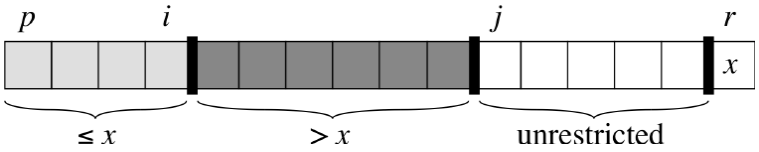
To sort items in *A*

- select last item to be pivot *r*,
- divide remaining items into two subgroups: those smaller than *r* and those larger than *r*
- Recurse on each subgroup

Quicksort Loop Invariants

Loop invariant:

1. All entries in $A[p\,..\,i]$ are $\leq$ pivot.
2. All entries in $A[i+1\,..\,j-1]$ are $>$ pivot.
3. $A[r]$ = pivot.



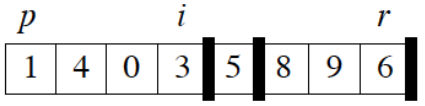
Quicksort

```
QUICKSORT(A, p, r)
  if p < r
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q − 1)
      QUICKSORT(A, q + 1, r)

Initial call is QUICKSORT(A, 1, n).
```

```
PARTITION(A, p, r)
  x = A[r]
  i = p − 1
  for j = p to r − 1
      if A[j] ≤ x
          i = i + 1
          exchange A[i] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

Quicksort in action



$A[r]$:	pivot
$A[j\,..\,r-1]$:	not yet examined
$A[i+1\,..\,j-1]$:	known to be > pivot
$A[p\,..\,i]$:	known to be ≤ pivot

Worksheet: Correctness

- Initialization:
- Maintenance:
- Termination

Quicktime runtime

Divide: O(n)

Conquer: T(A_L)+T(A_R)

Merge: O(1)

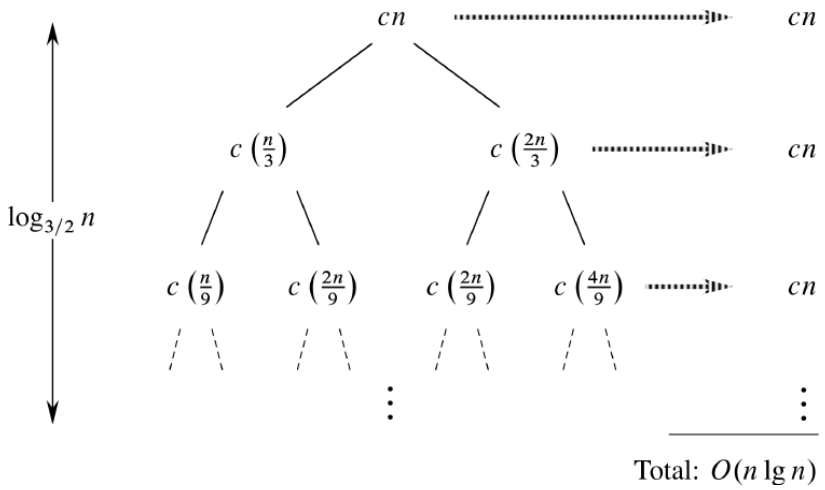
Worst-case:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \,. \end{aligned}$$

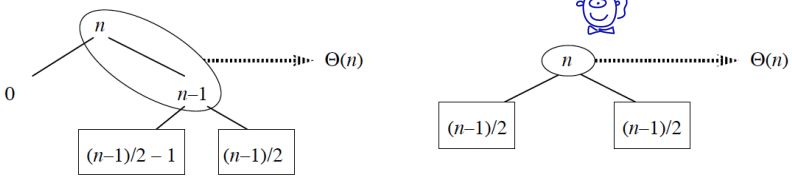
Best case:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \,. \end{aligned}$$

Proportional splits



Not all that can go wrong will...



Alternating between best/worst case adds only constant factor to tree depth

Average case analysis

- We'd like a way to describe what typically happens, i.e., what we expect to happen on average
- Turns out for quicksort, average case is similar to best case.

Avoiding the worst case: randomization

```
RANDOMIZED-PARTITION(A, p, r)
  i = RANDOM(p, r)
  exchange A[r] with A[i]
  return PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)
  if p < r
      q = RANDOMIZED-PARTITION(A, p, r)
      RANDOMIZED-QUICKSORT(A, p, q − 1)
      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

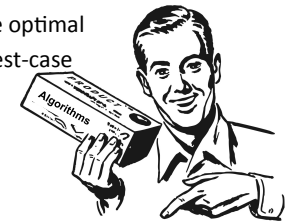
http://www.sorting-algorithms.com/quick-sort

Average-case analysis of Quicksort

See Chapter 5 for more!

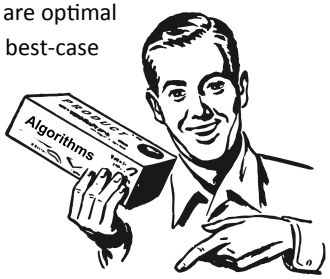
Bounding Algorithms vs. Problems

- Bounding Algorithms
  - Best-case, average-case, worst-case
  - For each: upper bound (O), lower bound (Ω), tight bound(θ)
- Bounding problems
  - Establish lower bounds for solving general problem
  - Why do we care?
    - We want to show that our algorithms are optimal
    - E.g., algorithm's worst-case = problem best-case



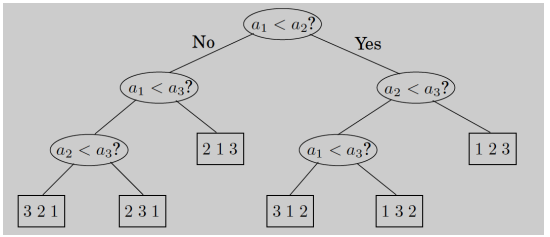
Lower-bounding sort

- Bounding problems
  - Establish lower bounds for solving general problem
  - Why do we care?
    - We want to show that our algorithms are optimal
    - E.g., algorithms worst-case = problem best-case
- How long does it take to sort?
  - Insertion, Merge, Quick
    - $\Omega(n \log n)$
  - Can we do better?
    - $\Omega(n)$ to examine input



Comparison-based search

- Decision tree:



- Comparisons form a binary tree
- Each permutation of *n* input elements must appear as a leaf
- Depth--number of comparisons on the longest path--represents worst-case complexity

Bounding comparison-based search

- Observations
  - Number of permutation (leaves) for *n* elements: $n!$
  - Binary tree of depth *d* has at most: 2^d leaves
  - $2^d \geq l \geq n!$
  - $n! \geq (n/2)^{(n/2)}$ Why? $n! = 1 \cdot 2 \cdot \cdots \cdot n$
  - $n! > (n/e)^n$ (Stirling's approximation)
- Any decision tree sorting *n* elements has depth $\Omega(n \log n)$
  - Depth *d* $\geq \log(n!)$
    $\geq \log(n/2)^{(n/2)}$
    $= \frac{n}{2} \log(n/2)$
    $= \frac{n}{2} \log n - \frac{n}{2} \log 2$
    $= \Omega(n \log n)$

Coming up…

- Classic D&C: Strassen's Matrix Multiplication
- Beating n log n sorts
- Order in this array!