Lec05-asymptotics

KT 2.1 - 2.2

Thursday, September 5, 2024 9:52 PM

What does it mean for an algorithm to be fast?

Who's a faster runner? Usain Bolt - 100m in 9.58 s, Berlin (2009)

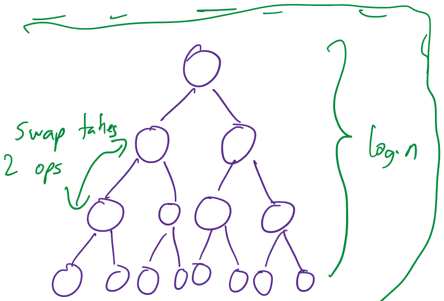
Eliud Kipchoge - 26.2 miles in 2:01:09, Berlin (2022)

- Need a formal definition of algorithmic efficiency to avoid vagueness
- Something concrete and falsifiable
- We care about scaling to large problems
  - ↳ small problems don't take too long even for inefficient algs

Proposal: count the exact number of operations (done by a computer)

Ex. Heap insert takes (at worst) $\underbrace{1}_{\text{find leaf}} + \underbrace{2}_{\text{swap 2 pts}} \underbrace{\log n}_{\text{height of heap}} = O(\log n)$

Heap delete takes $\underbrace{1}_{\text{find leaf}} + \underbrace{2}_{\text{swap with leaf}} + \underbrace{1}_{\text{delete leaf}} + \max(\underbrace{2 \log n}_{\text{sift up}}, \overbrace{\underbrace{2}_{\text{find smaller child}} \cdot \underbrace{2 \log n}_{\text{swaps}}}^{\text{sift down}}) = O(\log n)$



As $n \to \infty$, for large problems, log n grows, but 4 doesn't, so runtime of above is dominated by log n term

Instead of $n^2 + 4n + 2$, we just write $O(n^2)$

Def. (O) (upper bound) A runtime $T(n)$ is $O(f(n))$ if $\exists$ constants $n_0 \geq 0$, $c > 0$ s.t.

$$\underbrace{T(n) \leq c f(n)}_{\text{running time is bounded by}} \quad \underbrace{\forall n \geq n_0}_{\text{for all large enough instances}}$$

running time is bounded by a constant multiple of $f(n)$ — for all large enough instances

$O(\cdot)$ upper bounds the runtime

Def. (Ω) (lower bound) $T(n)$ is $\Omega(f(n))$ if $\exists$ constants $n_0 \geq 0$, $\varepsilon > 0$ s.t.

$$T(n) \geq \varepsilon f(n) \quad \forall n \geq n_0$$

$\Omega(\cdot)$ lower bounds the runtime.

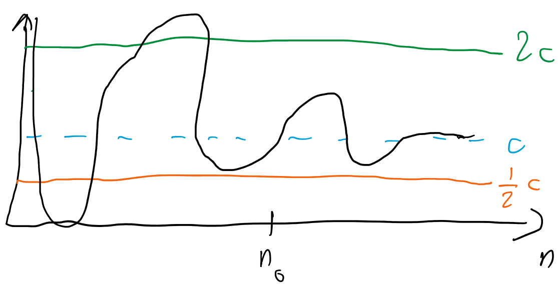
Def. (Θ) (tight bound) $T(n)$ is $\Theta(f(n))$ iff $T(n)$ is $O(f(n))$ and $\Omega(f(n))$

normally we are referring to worst-case example. Sometimes will be faster, but there are cases where it takes this long, but no longer.

Asymptotic limit

Thm (Theta) If $\lim_{n\to\infty} \frac{T(n)}{g(n)} = c$ for constant $c > 0$, then $T(n) = \Theta(g(n))$.

proof. $\exists\, n_0$ s.t. $\frac{c}{2} \leq \frac{T(n)}{g(n)} \leq 2c$ for all $n \geq n_0$ by def. of limit.



Therefore $T(n) \leq 2c\, g(n)$ for $n \geq n_0$ $\Rightarrow$ $T(n) = O(g(n))$

Also $T(n) \geq \frac{c}{2} g(n)$ for $n \geq n_0$ $\Rightarrow$ $T(n) = \Omega(g(n))$ $\boxtimes$

Linear time $O(n)$

- Find maximum in list A.

```
max = A[1]
for i=2 to n:
    if A[i] > max, then
```

takes n iterations for n items, constant # ops per iteration

for $i = 1$ to n:
  if $A[i] > max$, then
    set $max = A[i]$

takes n iterations for n items, constant # ops per iteration

- merging two sorted lists A_1, A_2 — each time, compare bottom item of both lists + take min.

- Finding connected components of forest (graph made up of trees)
  - Let $C = \{s\}$, for an arbitrary node s.
  - Let Q = list of all edges adj to s.
  - Repeat until $Q = \emptyset$:
    - Let $e = \{x, y\}$ be the first edge in Q.
    - Remove e from Q (dequeue)
    - WLOG, $x \in C$, $y \notin C$.
    - $C = C \cup \{y\}$
    - $Q = Q$ + [list of edges adj to y except e]

$G = (V, E)$
$|V| = n$
$|E| = m$ but $m < n$ because forest

every node/edge gets added only once because no cycles
$\Rightarrow O(n)$

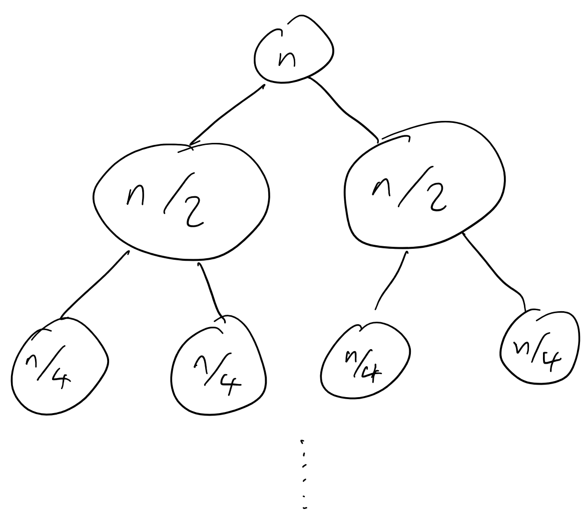
$O(n \log n)$ time — common in sorting. Why?

n

$n = \frac{n}{2} + \frac{n}{2}$

$n = \frac{n}{4} \cdot 4$

log n levels

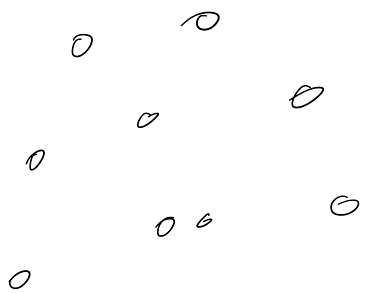


Sorting time $T(n)$
sorting for half the list $T(n/2)$
merging takes $O(n)$ time
$\Rightarrow T(n) = 2T(n/2) + n$

Linear amount of work per level
$\log n$ levels.
(Bottom level pairwise comparison)

$O(n^2)$ time — quadratic time

Ex. What is the smallest distance b/t a pair of pts? (brute force)



$\underbrace{\binom{n}{2}}_{\#\text{ pairs}} = \frac{n(n-1)}{2} = O(n^2)$ pairs

---

$O(n^3)$ time – cubic time

Ex. Matrix multiplication (schoolbook)

$$\text{row } i \begin{bmatrix} \\ a_{i1} \; a_{i2} \; \cdots\cdots \; a_{in} \\ \\ \end{bmatrix} \begin{array}{c} \text{col } j \\ \begin{bmatrix} b_{1j} & \\ b_{2j} & \\ \vdots & \\ b_{nj} & \end{bmatrix} \end{array} = \begin{bmatrix} \\ c_{ij} \\ \\ \end{bmatrix}$$

$A \in \mathbb{R}^{n^2}$ $\qquad B \in \mathbb{R}^{n^2}$ $\qquad C = \mathbb{R}^{n^2}$

$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ $\;=\;$ n multiplications per entry

n^2 entries

$\Rightarrow O(n^3)$

$O(n^k)$ – larger polynomials e.g. k-nested for loops

Ex. Independent set of size k. – no two nodes are adjacent

Given graph with n nodes, find one or say none exist

For every subset S of k nodes $\quad \} \; O(n^k)$ subsets $\binom{n}{k}$

If S is an independent set, $\quad k^2$

return S.

return failure

$O(2^n)$ – Exponential time

Ex. Largest independent set in graph

There are 2^n subsets to check via brute force

Ex. Largest independent set in graph

There are 2^n subsets to check via brute force.

- $O(\underbrace{n^2}\underbrace{2^n}_{\text{each subset}})$

checking if any edge b/t pairs of nodes

Sublinear time

- don't even need to look at all inputs
- must be able to query points w/o reading through

Ex. binary search on sorted list $O(\log n)$