Lec05-asymptotic-analysis

KT 2.1–2.2

Tuesday, September 5, 2023 9:02 PM

What does it mean for an algorithm to be fast?

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

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

- 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 for even inefficient algs

Proposal: count the exact number of operations.

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

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

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.

$$T(n) \leq c f(n) \quad \forall\, n \geq n_0$$

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

$O(\cdot)$ says the longest possible time an algorithm can take.

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

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

$\Omega(\cdot)$ says the shortest possible time an alg can take.

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

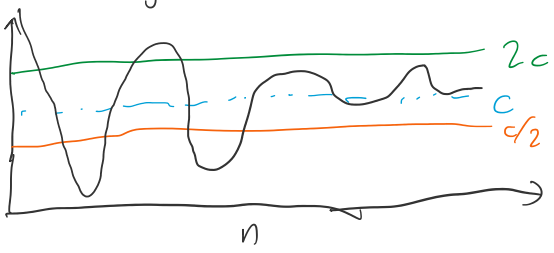
Def. (Θ). $T(n)$ is $\Theta(f(n))$ iff $T(n)$ is $O(f(n))$ and $\Omega(f(n))$.
(tight bounds)

still refers to worst-case example. Sometimes will be faster, but there are bad cases where it takes that 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. $\frac{c}{2} \le \frac{T(n)}{g(n)} \le 2c$ for all $n \ge n_0$ by def of limit.



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

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

Linear time $O(n)$

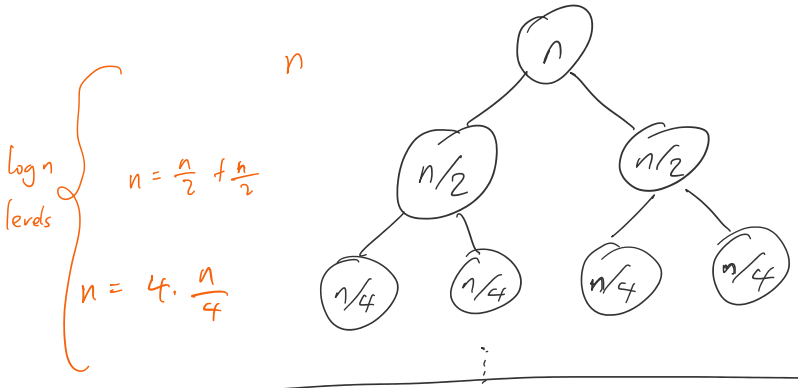
- Find maximum in list A.

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

} takes n iterations for n items, constant number of operations per iteration

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

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



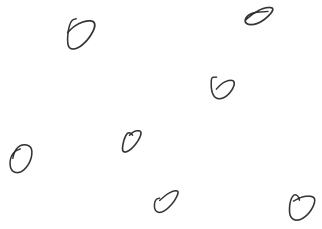
Sorting time $T(n)$

Sorting for half the list $T(n/2)$

$\Rightarrow T(n) = 2\,T(n/2) + n$

$O(n^2)$ time - quadratic time

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



$\binom{n}{2} = \frac{n(n-1)}{2} \approx O(n^2)$ pairs

$O(n^3)$ time - cubic time

Ex. matrix multiplication (standard)

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

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

$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ = n multiplications for 1 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.

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

For every subset S of k nodes:

If S is an independent set

return S

return Failure

} $O(n^k)$ subsets $\binom{n}{k}$

Exponential Time $O(2^n)$

Ex. Largest independent set in graph.

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

- $O(n^2 2^n)$
  - (each subset
  - checking if any edges between 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 $O(\log n)$