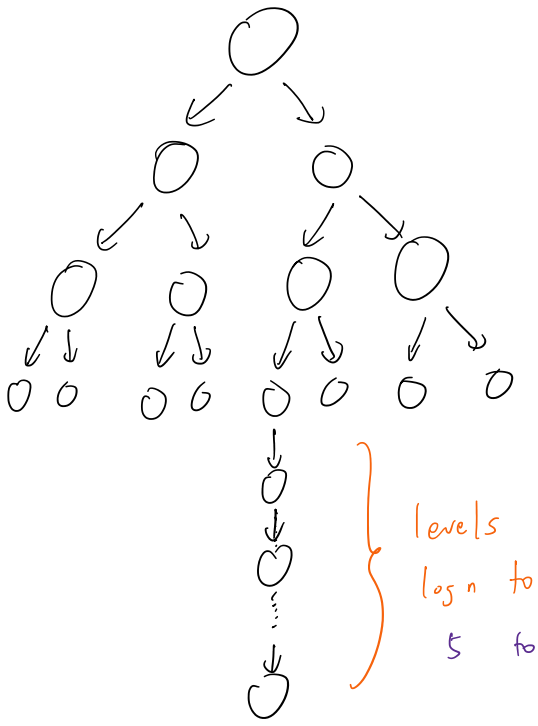
Lec14-averaged-analysis

Sunday, September 22, 2024 2:34 PM

Worst-case runtime What's the worst that can happen?



24 nodes

15 in balanced complete tree

9 in long linear branch

O(n)

Worst-case: find goes to bottom (13)

Average-case: find is faster because don't have to go to bottom (but still O(n) in this case)

But what are we averaging over? Possible inputs? A probability distribution?

Ex. Looking at average value over

Rolling 10-sided die many times vs. shuffling & dealing deck of 10 cards 1-10

randomized value amortized value

⇒ both have an expected average of 5.5.

Randomized: can get average of 10 if unlucky.

Average is over "random" chance.

Amortized: can never get worst answer every time, though can occasionally.

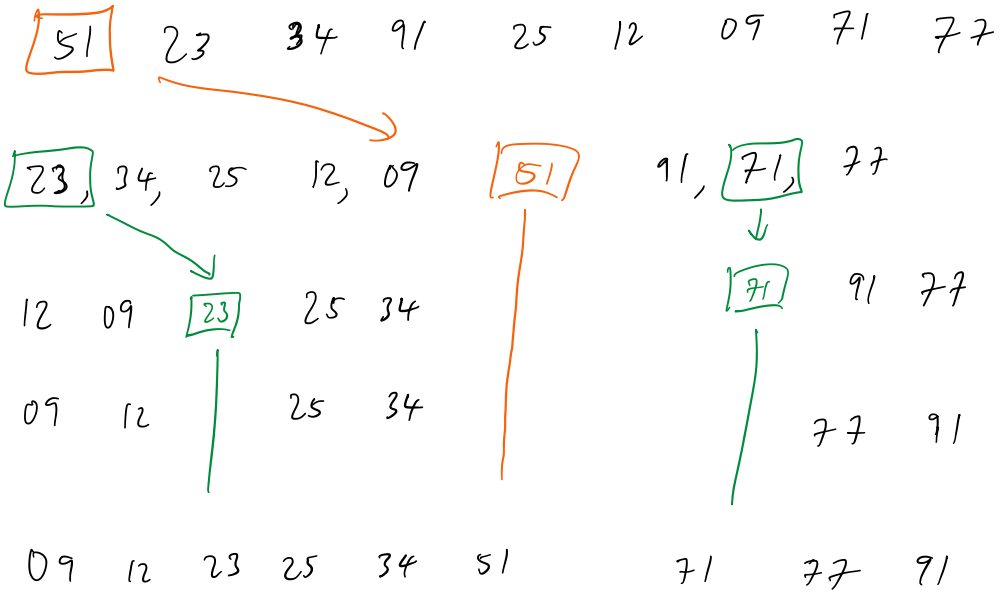
Average is over a constrained sequence of events

In-class demo: 10-sided die vs. deck of cards

Expected-time complexity of randomized alg:

Given a known probability distribution we impose via "coin flips", what is the average amount of time an algorithm takes to run.

Ex. Quicksort: Given an unsorted list, pick a random item as pivot, and divide into two smaller lists, one where everything is smaller, and one where everything is larger, & recurse.



Runtime: On average, each division creates two "half-size" lists.

Need $O(\log n)$ divisions.

Each iteration takes $O(n)$ comparisons to divide up set items.

$\Rightarrow O(n \log n)$ time (expected).

Worst-case: Always "accidentally" pick smallest item as pivot

Then need $O(n)$ division $\Rightarrow O(n^2)$ time.

Amortized time-complexity

- Average cost of operation over a series of operations
  - some ops costly, some cheap

total area of m bars bounded by some area function $f(m, n)$

m: # ops, n: # elements

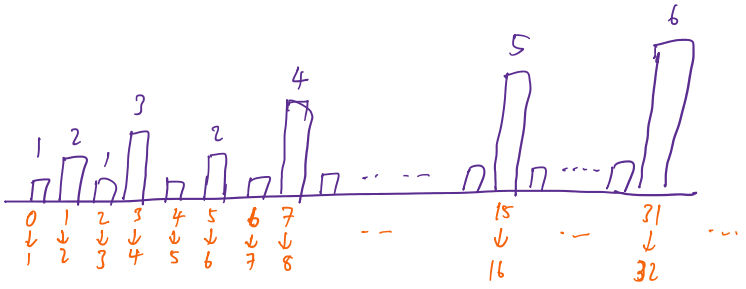
If $f(m,n) = O(m \log n)$, each operation takes $O(\log n)$ amortized time.

Ex. Binary counter: Cost to flip a bit is \$1

0: 00000000 ↓ \$1
1: 00000001 ↓ \$2
2: 00000010 ↓ \$1
3: 00000011 ↓ \$3
4: 00000100
!
33: 00100001

Worst-case cost to increment counter to k is \$ $\log k$, since might need to flip all bits.

Amortized average case ?



Claim: Cost to increment counters from 1 to k is only \$ $2k$.

Method: Prepay for $1 \to 0$ flips when we do a $0 \to 1$ flip.

i.e. when we flip a counter $0 \to 1$, pay \$1 to flip, and pay \$1 to store dollar on counter.

Then when we flip it back from $1 \to 0$, we have already paid for the flip back.

proof. On each increment, only one counter goes from $0 \to 1$.

Any other changing counters go from $1 \to 0$, so have already been paid for.

$\Rightarrow$ incrementing to k takes \$ $2k$ dollars.

$\Rightarrow$ amortized cost of bit flip is \$2, not \$ $\log k$.

Alt proof. Counter 1 flips every increment, so k times

Counter 2 flips every 2nd inc, so $\frac{k}{2}$ times

" 3 " " 4th inc, so $\frac{k}{4}$ times

$\frac{k}{8}$ times

$\vdots$

+)

$2k$ flips total.