6.006 Recitation 10

Agenda

- lect review
  - lower bound
  - counting sort.

Overhead

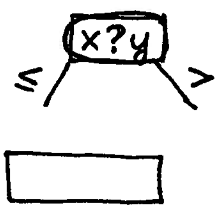
- Quiz
  - Friday = review
  - Wednesday = optional
  - problems soon
  - excercises in text
- Pset 3 out : work in grps!

Lower Bounds on Sorting

- if this hurts your brain : don't worry about it.

= Decision Tree

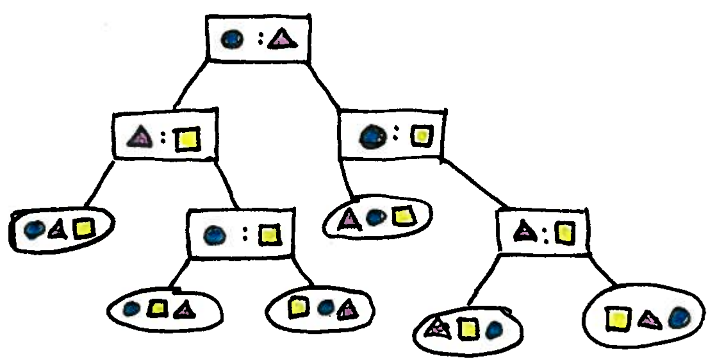
- each node is a comparison → 2 possibilities → binary tree.
- each leaf is a possible termination of the algorithm.
- a traversal from root to leaf is ONE execution of the alg.



↑ permutation of input array.

example: insertion sort 3 elts. (construct for students)

permutations:



note: all permutations are ∈ leaf set of decision tree.

math

$n!$ = # permutations of input

h = height of decision tree = runtime of algorithm.

nodes in tree of height $h \le 2^h$

min # nodes in a correct comparison sort alg = $n!$

$$n! \le 2^h$$

$$h \ge \lg(n!) = n\lg(n)$$

so $n\lg(n)$ is a lower bound on the runtime of a comparison-based sort.

COUNTING SORT

idea: don't compare them!

assumption: integers only.

- use the values as indecies into an auxillary array.

input : [ ] len=n — $x \in$ array then $x \in \mathbb{Z}_k$

aux : [ ] len=k = counts # of occurences of each int.

↓

[ ] = location in end array to put each elt. ((cumulative sum) of counts array)

output : [ ] len=n.

example: (step·by·step)

2	0	1	1	4	0	2	1

→

0	1	2	3	4
2	3	2	0	1

↓

0	1	2	3	4
2	5	7	0	8

oops too high! sub 1.

←

0	1	2	3	4	5	6	7
0	0	1	1	1	2	2	4

- really when going from (input, aux) → output we traverse the input array in reverse.
  - why? STABLE = if two values are "equal" and x comes before y in input, then x before y in output

- runtime:
  1. go through input = A — n
     a. for each elt – incr aux[A[i]]
  2. Compute cumulative sum of aux — k
  3. go through input A — n
     a. for each elt – find loc. in output using aux
     b. copy elt into output.
     c. decr aux[A[i]]

  $\Rightarrow \Theta(2n+k)$

FAST! (probably)

what if k is large? → RADIX!

Quick Probability Review

- probability $P(dice=4) = 1/6$
- expected valve $E[\text{dice roll}] = \sum_{i=1}^{6} i \frac{1}{6} = \frac{1}{6}(1+2+3+4+5+6) = 3.5$

$E[\text{2 rolls}] = \sum_{i=1}^{12} i\, P(\text{2 rolls} = i) \leftarrow$ hard.

- linearity! $= E[\text{roll1} + \text{roll2}] = E[R1] + E[R2] = 7$

in general... $E[\Sigma(\cdots)] = \Sigma(E[\cdots])$

- indicator random variables

$I_A = \begin{cases} 1 & A \text{ occurs} \\ 0 & \text{else.} \end{cases}$ A is an event.

$E[I_A] = Pr(A)$ (proof: $E[I_A] \overset{\Sigma}{=}$ valve · prob $= 1\,P(A) + 0(P(\sim A)) = P(A)$

ex: expected # of 6s in k rolls.

$I = \sum_{i=1}^{k} I_i$ where I_i = got a 6 on the i^{th} roll.

$E[I] = E\left[\sum_{i=1}^{k} I_i\right] \underset{\text{linearity.}}{=} \sum_{i=1}^{k} E[I_i] = k \frac{1}{6}$

Coding

Good Practises

A write comments first
1. help you
2. help others help you.
3. you will forget what it does no matter how obvious

B use small, simple pieces
- break into fcns if necessary
1. easier to do
2. easier to understand
3. easier to test.

C tests first then code.
1. understand corner cases before you run into them.
2. can run as soon as code written to see if it works.

D Step away from your code sometimes: fresh eyes are good.

Debugging

goal: isolate the problem

B & C help w/ this ALOT

- use asserts / prints

Testing

- **TRY** to break the code
  - use corner cases (0, 1, -1 ... ∞)
  - give invalid inputs ← if robustness matters.
  - sample the problem space well