

## Intro

Administrivia.

- Signup sheet.
- prerequisites: 6.046, 6.041/2, ability to do proofs
- homework weekly (first next week)
- collaboration
- independent homeworks
- grading requirement
- books.
- **question:** scribing?

Randomized algorithms: make random choices during run. Main benefits:

- speed: may be faster than any deterministic
- even if not faster, often simpler (quicksort)
- sometimes, randomized is best
- sometime, randomized idea leads to deterministic algorithm

Distinguish average-case analysis

- Probabilistic analysis assuming random input
- randomized algorithms do not assume random inputs
- so analyses are more applicable

We don't really use random numbers. But randomized algorithms break patterns we don't know are there.

- deterministic algorithm: works well except a few specific cases.
- But those are the ones you will encounter (Murphy)!

- randomized: almost always works well on any case
- but sometimes does bad on any case, so risky for life-threatening errors.

Course objective:

- Randomization is a general technique. Applies to all areas of CS.
- Underlying it is a common set of tools.
- Goal is to give familiarity with those tools so you can apply them to your own problems.
- To present tools, we draw applications from many areas of CS: data structures, geometric algos, graph algos, parallel and distributed, number theory.
- Because so many, only a brief taste of each.
- But sufficient to go on alone.

Basic methodologies.

- Avoiding adversarial inputs
  - sorted quicksort list
  - a kind of random reordering (geometry—BSP)
  - hashing to same buckets
  - online algorithms
  - **note:** “adversarial” may mean “well structured” i.e. natural
- fingerprinting/verification
  - generate short random fingerprints for things
  - faster than comparing things
  - almost every fingerprint works
  - so a random one works
- random sampling. graph algs, computational geometry, median

- fast way to find “typical” members
- solve representative subproblem fast
- extrapolate to solution of original problem
- load balancing
  - randomization spreads things out uniformly
  - parallel algs, routing, hashing
- symmetry breaking
  - random decisions keep everyone from doing the same thing
  - ethernet
  - deadlocks avoidance in distributed systems (MUST randomize)
- Probabilistic existence proofs
  - thought experiment
  - prove an object is build with positive probability
  - guarantees object exists
  - makes search for algo worthwhile.

Today: 2 really basic principles:

- linearity of expectation
- product of event probabilities (independence)

Then some fundamental ideas:

- Kinds of randomized algorithms
- a bit of complexity

## Quicksort

Items  $S_1, \dots, S_n$  to be sorted

- suppose could pick middle element:

$$T(n) = 2T(n/2) + O(n) = O(n \log n)$$

works since divides into much smaller subproblems

- picking middle is hard. But an almost middle element is OK.
- pick random element. “probably” near middle and divides problem in two
- bound expected number of comparisons  $C$
- $X_{ij} = 1$  if compare  $i$  to  $j$
- **linearity of expectation:**  $E[C] = \sum E[X_{ij}]$
- $E[X_{ij}] = p_{ij}$
- Consider smallest recursive call involving both  $i$  and  $j$ .
- pivot must be one of  $S_i, \dots, S_j$ . all equally likely
- $S_i$  and  $S_j$  get compared if pivot is  $S_i$  or  $S_j$
- probability is at most  $2/(j - i + 1)$  (may have outer elements)
- analysis:

$$\begin{aligned} \sum_{i=1}^n \sum_{j>i} p_{ij} &\leq \sum_{i=1}^n \sum_{j>i} 2/(j - i + 1) \\ &= \sum_{i=1}^n \sum_{k=1}^{n-i+1} 2/k \\ &\leq 2 \sum_{i=1}^n \sum_{k=1}^n 1/k \\ &\leq 2nH_n \end{aligned}$$

(Define  $H_n$ , claim  $O(\log n)$ .)

$$= O(n \log n).$$

- analysis holds for every input, doesn't assume random input
- we proved expected. can show high probability
- how did we pick a random elements? Depends on model.
- algorithm always works, but might be slow.

## BSP

- linearity of expectation. hat check problem
- Rendering an image
  - render a collection of polygons (lines)
  - painters algorithm: draw from back to front; let front overwrite
  - need to figure out order with respect to user
- define BSP.
  - BSP is a data structure that makes order determination easy
  - Build in preprocess step, then render fast.
  - Choose any hyperplane (root of tree), split lines onto correct side of hyperplane, recurse
  - If user is on side 1 of hyperplane, then nothing on side 2 blocks side 1, so paint it first. Recurse.
  - time=BSP size
- sometimes must split to build BSP
- how limit splits?
- autopartitions
- random auto

- analysis
  - $index(u, v) = k$  if  $k$  lines block  $v$  from  $u$
  - $u \dashv v$  if  $v$  cut by  $u$  auto
  - probability  $1/(1 + index(u, v))$ .
  - tree size is (by linearity of  $E$ )

$$n + \sum_u 1/index(u, v) \leq \sum_u 2H_n$$

- result: **exists** size  $O(n \log n)$  auto
- gives randomized construction
- equally important, gives **probabilistic existence proof** of a small BSP
- so might hope to find deterministically.

## MinCut

- the problem
- contraction
- conditionally independent events
- give/analyze
- repetition for better success probability (independent events)
- faster implementation later

## Monte Carlo vs. Las Vegas

- turn LV to MC by truncating
- turn MC to LV by certifying.
- if can't certify, dangerous!