Contents

% Chapter 04 — Randomized Algorithms % CS161 Reader (Plotkin F10) %

1 Chapter 04 — Randomized Algorithms

Randomization is a *proof technique disguised as an implementation trick*. The key discipline is to state **exactly** what is random, what is adversarial, and what kind of guarantee you are proving.

Primary patterns: expectation (indicators + linearity), amplification (repeat/boost), induction (recurrences).

Executable witnesses: `cs161lab.algorithms.sorting.rand_quicksort`, `cs161lab.algorithms.sorting.qui`
`cs161lab.algorithms.mincut.karger`.

1.1 4.1 Model and guarantee types

We analyze algorithms in the randomized RAM model: the algorithm may draw independent random bits at any step.

1.1.1 Las Vegas vs Monte Carlo

- **Las Vegas**: always correct; randomness affects running time (Randomized QuickSort).
- **Monte Carlo**: fixed running time; randomness affects correctness (Karger's min-cut).

1.1.2 Common guarantee traps

Expected-time bounds do **not** automatically imply "fast with high probability." When you need high probability, use explicit tail bounds or amplification (repetition plus a correctness filter or "best-of" selection).

1.2 4.2 Linearity of expectation and indicator variables

1.2.1 Theorem 4.1 (Linearity of expectation)

For any random variables $X_1, \ldots, X_m$ (independence not required),

$$E\Big[\sum_{i=1}^{m} X_i\Big] = \sum_{i=1}^{m} E[X_i].$$

Proof. Expand $E[\cdot]$ by definition (sum/integral) and swap summations/integrals. $\square$

1.2.2 Lemma 4.2 (Indicators)

If I is an indicator (0/1) for an event E, then $E[I] = P(E)$.

Proof. $E[I] = 0 \cdot P(I = 0) + 1 \cdot P(I = 1) = P(I = 1) = P(E)$. $\square$

This is the standard CS161 move: reduce expected counts to sums of event probabilities.

1.3 4.3 Randomized QuickSort

1.3.1 Algorithm

Given n distinct keys: 1. Pick a pivot uniformly at random from the subarray. 2. Partition around the pivot. 3. Recurse on both sides.

Executable witness: `cs161lab.algorithms.sorting.rand_quicksort`
Trace events: `pivot`, `partition`.

1.3.2 Theorem 4.3 (Expected comparisons)

Let C_n be the number of comparisons. Then

$$E[C_n] = 2(n+1)H_n - 4n = O(n \log n),$$

where $H_n = \sum_{k=1}^{n} \frac{1}{k}$ is the n-th harmonic number.

1.3.2.1 Proof (indicator-variable proof)

Label keys in sorted order $z_1 < z_2 < \cdots < z_n$. For each $i < j$, define $I_{ij} = 1$ iff QuickSort compares z_i and z_j at some point. Then

$$C_n = \sum_{i<j} I_{ij} \quad \Rightarrow \quad E[C_n] = \sum_{i<j} E[I_{ij}] = \sum_{i<j} P(I_{ij} = 1).$$

Consider the set $S_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$. Keys z_i and z_j are compared **iff** the first pivot chosen from S_{ij} is one of the endpoints (z_i or z_j). If the first pivot in S_{ij} is some z_k with $i < k < j$, the recursion separates z_i and z_j forever.

The first pivot drawn from S_{ij} is uniformly distributed among its $|S_{ij}| = j - i + 1$ elements, so

$$P(I_{ij} = 1) = \frac{2}{j - i + 1}.$$

Therefore,

$$E[C_n] = \sum_{i<j} \frac{2}{j-i+1} = \sum_{\ell=2}^{n} \sum_{i=1}^{n-\ell+1} \frac{2}{\ell} = \sum_{\ell=2}^{n} \frac{2(n-\ell+1)}{\ell} = 2(n+1)(H_n-1)-2(n-1) = 2(n+1)H_n-4n.$$

□

1.4 4.4 Randomized Selection (QuickSelect)

QuickSelect finds the k-th smallest element by partitioning around a random pivot and recursing on one side.

Executable witness: `cs161lab.algorithms.sorting.quickselect`
Trace events: `pivot`, `partition`, `shrink`, `found`.

1.4.1 Theorem 4.4 (Expected linear time)

The expected running time of QuickSelect is $O(n)$.

1.4.1.1 Proof (good pivot with constant probability)

Partitioning costs cn. A pivot is **good** if its rank lies in the middle half (between $n/4$ and $3n/4$), which happens with probability at least 1/2. With a good pivot, the recursion continues on size at most $3n/4$. With a bad pivot, size is at most $n-1$. Hence,

$$E[T(n)] \le cn + \frac{1}{2}E[T(3n/4)] + \frac{1}{2}E[T(n-1)].$$

A standard induction shows this solves to $E[T(n)] \le an$ for a sufficiently large constant a. □

1.5 4.5 Amplification and the union bound

1.5.1 Theorem 4.5 (Union bound)

For events $E_1, \dots, E_m$,

$$P\Big(\bigcup_{i=1}^{m} E_i\Big) \le \sum_{i=1}^{m} P(E_i).$$

1.5.2 Amplification lemma

If an independent trial succeeds with probability at least $p > 0$, repeating t times yields failure probability at most $(1-p)^t \le e^{-pt}$. To get failure at most δ, choose $t \ge \frac{1}{p}\ln(1/\delta)$.

1.6 4.6 Karger's randomized min-cut

Karger's algorithm is Monte Carlo: one run can fail, but it is fast, and repetition makes failure negligible.

Executable witness: `cs161lab.algorithms.mincut.karger`
Trace events: `contract`, `self_loop`, `trial`, `best`, `result`.

1.6.1 Algorithm (Random Contraction)

While more than 2 supernodes remain: 1. pick a uniformly random edge, 2. contract its endpoints, 3. delete self-loops. Return the cut induced by the final 2 supernodes.

1.6.2 Theorem 4.6 (One-run success probability)

For an n-vertex multigraph with min-cut value λ, one run returns a min-cut with probability at least $\frac{2}{n(n-1)}$.

1.6.2.1 Proof

Fix a particular minimum cut C of size λ. The run succeeds if no contraction ever contracts an edge crossing C. When there are k supernodes left and we have not crossed C, the cut still has exactly λ crossing edges. Also, the number of edges m_k satisfies $m_k \geq k\lambda/2$ because the minimum degree is at least λ (otherwise we'd have a smaller cut). Thus,

$$P(\text{avoid } C \text{ at stage } k) = 1 - \frac{\lambda}{m_k} \geq 1 - \frac{\lambda}{k\lambda/2} = 1 - \frac{2}{k}.$$

Multiply from $k = n$ down to 3:

$$P(\text{avoid } C \text{ for all contractions}) \geq \prod_{k=3}^{n} \left(1 - \frac{2}{k}\right) = \prod_{k=3}^{n} \frac{k-2}{k} = \frac{2}{n(n-1)}.$$

□

1.6.3 Corollary 4.7 (Repetition schedule)

After t independent runs and taking the best cut,

$$P(\text{all fail}) \leq \left(1 - \frac{2}{n(n-1)}\right)^t \leq \exp\left(-\frac{2t}{n(n-1)}\right).$$

To make failure $\leq \delta$, it suffices to take $t = \Theta(n^2 \log(1/\delta))$.

1.7 4.7 What can go wrong

- Running a Monte Carlo algorithm without amplification (and then believing it).
- Assuming tail bounds from expectations without proof.
- Non-uniform pivot selection (implementation bias).
- PRNG seeding mistakes that destroy independence assumptions.

1.8 Appendix — Trace events as proof witnesses

- QuickSort: `pivot`, `partition` support the indicator proof narrative (who gets compared).
- QuickSelect: `shrink` is a termination witness; the good-pivot argument predicts frequent shrinkage.
- Karger: `contract` and `trial` let you empirically validate the repetition schedule.