AAD Course Project – Randomized Algorithms

Team Singularity

Members: Adithya, Aryama, Sahid, Sushil, Yashas

Repository: `https://github.com/Sushil2006/singularity_aad_project/`

Abstract

Randomness powers some of the fastest and most elegant algorithms known. In this project we study multiple randomized paradigms through formal proofs, code implementations, and benchmarks across wide-ranging test cases. We highlight when randomness beats deterministic lower bounds, how error can be driven exponentially low, and where these ideas solve real-world problems on real-world data. The goal is to motivate, implement, and empirically validate randomized algorithms with rigor and ambition.

- **Definition and model:** A randomized algorithm draws unbiased random bits to guide its internal choices. Runtime and output become random variables; guarantees are stated in expectation or with high probability.
- **Why randomness helps:**
  - **Break worst-case structure:** Random choices defeat adversarial input orders and pathological patterns.
  - **Simplicity with strong bounds:** Hashing, sampling, and randomized rounding often yield cleaner code and clean expected-time guarantees.
  - **Performance on real-world datasets:** Massive graphs, streams, and high-dimensional datasets—mirroring real-world data—admit fast randomized sketches and samplers where exact deterministic methods are infeasible.
  - **Provable speedups:** When deterministic lower bounds target worst case only, randomness can improve expected or smoothed complexity.
- **Las Vegas vs. Monte Carlo:**
  - **Las Vegas:** Always correct; randomness affects runtime (e.g., randomized QuickSort, randomized incremental constructions). Guarantee: exact output with expected (and often tail) bounds on time.
  - **Monte Carlo:** Fixed resource budget; output may err with small probability (one- or two-sided). Error is reduced exponentially by independent repetitions and majority/threshold rules (e.g., Miller–Rabin, randomized min-cut).
- **Complexity classes for randomness:**
  - **RP / coRP:** One-sided error; RP accepts yes-instances with probability at least 1/2 and never accepts no-instances; coRP is the complement.

  - **BPP:** Two-sided error bounded away from 1/2 (e.g., 1/3); amplification drives error to $2^{-\Omega(k)}$ with k repetitions.
  - **ZPP:** Zero-error (Las Vegas) with expected polynomial time; ZPP = RP $\cap$ coRP.
  - **PP / BQP:** PP allows majority acceptance with unbounded two-sided error; BQP is the quantum analogue of BPP with bounded two-sided error on a quantum computer.
  - **Other classes:** MA and AM capture randomized proof systems; RL vs. L address randomized logspace. These formalize how randomness affects power under time/space constraints.

- **Error management and amplification:**
  - **Independent repetition:** Re-run and aggregate outputs (e.g., majority) to exponentially reduce failure probability.
  - **Confidence vs. cost:** Amplification trades a multiplicative work factor for exponentially smaller error, enabling user-chosen confidence.

- **Takeaways:**
  - **Power of randomness:** Simplifies algorithms, thwarts worst-case inputs, and yields strong expected or with-high-probability guarantees.
  - **Formal lenses:** Las Vegas vs. Monte Carlo and classes (RP, BPP, ZPP, BQP, etc.) formalize correctness and resource trade-offs.
  - **Tunable confidence:** Amplification converts modest per-run guarantees into very high confidence with predictable cost.