Comp 480/580 — Assignment #1

Dev Sanghvi - ds221

Rice University
Date: 09/30/2025

Constants

- **Global seeds**: `SEED_COEFFS = 580123`, `SEED_KEYS = 20250916`, `SEED_BLOOM = 137`.
- **Prime** $P = 1{,}048{,}573$.
- **Fixed coefficients** (drawn once with `SEED_COEFFS` and then *frozen*):

$$a = 716{,}663, \quad b = 625{,}113, \quad c = 32{,}912, \quad d = 480{,}811.$$

1 Testing Hash Functions

Goal Check for the avalanche behavior: for 31 input bits and 10 output bits, estimate the probability $P[\text{output bit } i \text{ flips} \mid \text{input bit } j \text{ flips}]$ for each of four hash families:

$$
\begin{aligned}
h_1(x) &= ((ax + b) \bmod P) \bmod 1024 \quad \text{(2-universal)},\\
h_2(x) &= ((ax^2 + bx + c) \bmod P) \bmod 1024 \quad \text{(3-universal)},\\
h_3(x) &= ((ax^3 + bx^2 + cx + d) \bmod P) \bmod 1024 \quad \text{(4-universal)},\\
h_4(x) &= \texttt{murmurhash3\_32}(x; \text{seed} = 137) \bmod 1024.
\end{aligned}
$$

I generated $N = 5000$ independent 31-bit positive integers x, flip each input bit $j \in \{0, \dots, 30\}$ to form $x \oplus 2^j$, compute $y = h(x), y' = h(x \oplus 2^j)$, and mark whether bit $i \in \{0, \dots, 9\}$ changed in $y \oplus y'$. This yields a 10×31 matrix of empirical probabilities for each hash.

Implementation details All random sources (x, coefficients) use fixed seeds noted above. MurmurHash3 uses its `scikit-learn` implementation.

Summary statistics Let A denote the 10×31 matrix of probabilities for a hash. I got mean(A), the average absolute deviation from 0.5 (AAD), and the min/max entry as follows:

Hash	mean(A)	AAD from 0.5	min(A)	max(A)
2-universal	0.5352	0.2231	0.0612	0.9692
3-universal	0.4997	0.0056	0.4804	0.5224
4-universal	0.5000	0.0055	0.4776	0.5170
MurmurHash3	0.5001	0.0060	0.4772	0.5204

2univ_linear_heatmap.png

(a) 2-universal (linear)

3univ_quadratic_heatmap.png

(b) 3-universal (quadratic)

4univ_cubic_heatmap.png

(c) 4-universal (cubic)

murmurhash3_heatmap.png

(d) MurmurHash3

Figure 1: Avalanche heatmaps (10×31). Color scale is centered at 0.5 (black), so lighter indicates deviation from 0.5.

Interpretation As expected, h_2, h_3, and MurmurHash3 exhibit near-ideal avalanche (entries close to 0.5 with small spread). The 2-universal linear hash is *not* avalanche: some output bits change almost deterministically for certain input-bit flips, while others rarely change. This is consistent with lower-independence families offering weaker bit-mixing, while higher-degree polynomials and practical non-cryptographic hashes like MurmurHash3 provide better diffusion.

2 Counting Turtle Confidence

Goal Using Chebyshev's inequality, I (i) give an explicit "constant×std" band for the sample mean $\bar{M}$ of overlap counts, (ii) show how to choose R so that the relative error $|\bar{M} - \mathbb{E}[M]| \leq f\,\mathbb{E}[M]$ holds with failure probability at most 0.05, (iii) translate the band into an interval for $\hat{n} = \frac{k_1 k_2}{\bar{M}}$, and (iv) note when estimation is hard.

Setup I repeat the same experiment R times and observe the i.i.d. counts $M_1, \ldots, M_R$. Let

$$\bar{M} = \frac{1}{R}\sum_{i=1}^{R} M_i, \qquad \mu \triangleq \mathbb{E}[M] = \frac{k_1 k_2}{n}$$

The usual estimator of population size is $\hat{n} = \dfrac{k_1 k_2}{\bar{M}}$.

Chebyshev band ("constant × std") Because $\mathrm{Var}(\bar{M}) = \mathrm{Var}(M)/R$, Chebyshev gives, for any $\delta \in (0, 1)$,

$$\Pr\big[\,|\bar{M} - \mu| \geq a\,\big] \;\leq\; \frac{\mathrm{Var}(M)}{R\,a^2}.$$

Choosing

$$a(\delta, R) = \sqrt{\frac{\mathrm{Var}(M)}{R\,\delta}} = \frac{1}{\sqrt{\delta}}\underbrace{\sqrt{\frac{\mathrm{Var}(M)}{R}}}_{\mathrm{std}(\bar{M})}$$

ensures $\Pr[|\bar{M} - \mu| \leq a] \geq 1 - \delta$. For the assignment's $\delta = 0.05$,

$$a = 4.4721\sqrt{\frac{\mathrm{Var}(M)}{R}}$$

Exact model vs. binomial approximation With sampling *without* replacement, M is hypergeometric:

$$M \sim \mathrm{Hypergeometric}(n,\, k_1,\, k_2), \quad \mathbb{E}[M] = k_2\frac{k_1}{n}, \quad \mathrm{Var}(M) = k_2\frac{k_1}{n}\Big(1 - \frac{k_1}{n}\Big)\frac{n - k_2}{n - 1}$$

Since $\frac{n-k_2}{n-1} \leq 1$, the binomial variance $k_2 p(1-p)$ with $p = \frac{k_1}{n}$ is an *upper bound* on the true variance. Thus, bands and R-requirements derived under the binomial model are *conservative* for the exact model.

How many repetitions R for a target relative error f with failure ≤ 0.05? Impose $|\bar{M} - \mu| \leq f\,\mu$ with failure at most δ. Set $a = f\,\mu$ and solve:

$$\delta \;\geq\; \frac{\mathrm{Var}(M)}{R\,(f\mu)^2} \quad\Longrightarrow\quad R \;\geq\; \frac{\mathrm{Var}(M)}{\delta\, f^2\, \mu^2}$$

Under the binomial upper bound (conservative),

$$R \;\geq\; \frac{k_2\, p(1-p)}{\delta\, f^2\, (k_2 p)^2} = \frac{1-p}{\delta\, f^2\, k_2\, p}\,, \qquad p = \frac{k_1}{n}$$

At $\delta = 0.05$ this specializes to $R \geq \dfrac{1-p}{0.05\, f^2\, k_2\, p}$.

Translating to an interval for $\hat{n}$ Since $g(m) = \frac{k_1 k_2}{m}$ is decreasing on $(0, \infty)$, the event $\bar{M} \in [\mu - a, \mu + a]$ with $\mu > a$ implies

$$\hat{n} \in \left[\frac{k_1 k_2}{\mu + a}, \frac{k_1 k_2}{\mu - a} \right]$$

Using $\mu = \frac{k_1 k_2}{n}$ this can be written as

$$\hat{n} \in \left[\frac{n}{1 + \frac{an}{k_1 k_2}}, \frac{n}{1 - \frac{an}{k_1 k_2}} \right]$$

In practice n is unknown; I use the plug-in $\hat{\mu} = \bar{M}$ (and $\hat{p} = \bar{M}/k_2$) inside a to produce a data-driven CI.

When is estimation hard? If $p = \frac{k_1}{n}$ is very small (few overlaps), then $\mu = k_2 p$ is small and $1/\bar{M}$ is unstable; Chebyshev is also loose in that regime. Practically, I either increase k_1, k_2 to raise μ or increase R via the formula above (or both).

3 Inequalities: Linear Probing with 5-independence

With load factor $\alpha = m/n = 1/3$, and assuming the following bound for the expected search cost in linear probing (given in the problem):

$$\mathbb{E}[\text{cost}] \;=\; \mathcal{O}(1) \sum_{s=1}^{\lfloor \log_2 n \rfloor} 2^s \cdot \Pr\left[B_s \geq 2\,\mathbb{E}[B_s]\right],$$

Where B_s counts the number of inserted keys that hash into a fixed *interval of length* 2^s (a contiguous block of table positions), under a *5-independent* hash function.

Goal Prove this sum is bounded by a constant (independent of n).

Setup Fix an interval I_s of length 2^s. Let

$$B_s \;=\; \sum_{i=1}^{m} X_i, \quad X_i = \mathbf{1}\{h(\text{key}_i) \in I_s\}, \quad p_s = \Pr[X_i = 1] = \frac{2^s}{n}$$

Then $\mu_s \triangleq \mathbb{E}[B_s] = mp_s = \alpha\, 2^s$ and $\mathrm{Var}(B_s) = mp_s(1 - p_s) \leq \mu_s$.

Fourth moment under 5-independence (statement & sketch). *Statement.* Under 5-independence, mixed moments up to order four factorize, and

$$\mathbb{E}\Big[(B_s - \mu_s)^4\Big] \;\leq\; C_1\, \mu_s \;+\; C_2\, \mu_s^2 \;=\; \mathcal{O}(\mu_s + \mu_s^2),$$

for absolute constants C_1, C_2 (when $\mu_s \geq 1$; the $\mu_s < 1$ case is handled separately below).

Sketch. Let $Y_i = X_i - p_s$ so $B_s - \mu_s = \sum_{i=1}^m Y_i$ with $\mathbb{E}[Y_i] = 0$, $\mathbb{E}[Y_i^2] = p_s(1 - p_s)$, and $\mathbb{E}[Y_i^4] \leq C_0 p_s$. Expanding and using that odd mixed moments vanish,

$$\mathbb{E}\left[(B_s - \mu_s)^4\right] = \sum_i \mathbb{E}[Y_i^4] + 6\sum_{i<j} \mathbb{E}[Y_i^2 Y_j^2],$$

and 5-independence yields $\mathbb{E}[Y_i^2 Y_j^2] = \mathbb{E}[Y_i^2]\mathbb{E}[Y_j^2] = p_s^2(1 - p_s)^2$. Hence

$$\mathbb{E}\left[(B_s - \mu_s)^4\right] \;\leq\; C_0 m p_s + 6\binom{m}{2} p_s^2 \;=\; \mathcal{O}(\mu_s + \mu_s^2), \quad \mu_s = mp_s = \alpha 2^s.$$

Markov on the 4th power gives

$$\Pr[B_s \geq 2\mu_s] \;\leq\; \frac{\mathbb{E}[(B_s - \mu_s)^4]}{\mu_s^4} \;\leq\; \frac{C}{\mu_s^2} \;=\; \frac{C}{\alpha^2\, 2^{2s}}$$

where $\mu_s = mp_s = \alpha 2^s$. Markov's inequality on the 4th power then gives $\Pr[B_s \geq 2\mu_s] \leq \mathbb{E}[(B_s - \mu_s)^4]/\mu_s^4 \leq C/\mu_s^2$, yielding the summable bound in the next step.

Summation From the fourth–moment bound we obtained

$$\Pr[B_s \geq 2\mu_s] \;\leq\; \frac{C}{\mu_s^2} \;=\; \frac{C}{(\alpha 2^s)^2} \;=\; \frac{C}{\alpha^2\, 2^{2s}}$$

Plugging into the given sum,

$$\sum_{s=1}^{\lfloor \log_2 n \rfloor} 2^s \cdot \Pr[B_s \geq 2\mu_s] \;\leq\; \sum_{s=1}^{\infty} 2^s \cdot \frac{C}{\alpha^2\, 2^{2s}} \;=\; \frac{C}{\alpha^2} \sum_{s=1}^{\infty} 2^{-s} \;=\; \frac{C}{\alpha^2} \cdot 1 \;=\; \mathcal{O}\Big(\frac{1}{\alpha^2}\Big)$$

Since the load factor $\alpha = m/n$ is fixed by the assignment (here $\alpha = \frac{1}{3}$), the right-hand side is a numerical constant independent of n; in fact it equals $9C$ when $\alpha = \frac{1}{3}$.

For the very small-mean regime $\mu_s = \alpha 2^s < 1$ (i.e., $2^s < 1/\alpha$), a crude bound $\Pr[B_s \geq 2\mu_s] \leq \Pr[B_s \geq 1] \leq \mu_s$ gives the partial contribution

$$\sum_{s:\,\mu_s<1} 2^s \cdot \mu_s \;\leq\; \sum_{s\leq s_0} 2^s \cdot (\alpha 2^s) \;=\; \alpha \sum_{s\leq s_0} 2^{2s} \;=\; \alpha \cdot \mathcal{O}(2^{2s_0}) \;=\; \alpha \cdot \mathcal{O}((1/\alpha)^2) \;=\; \mathcal{O}\Big(\frac{1}{\alpha}\Big),$$

where $s_0 = \lfloor \log_2(1/\alpha) \rfloor$ is constant when α is constant. Thus both parts (of large and small μ_s) are bounded by constants depending only on α, and the expected search cost is $\mathcal{O}(1)$ for $\alpha = \frac{1}{3}$.

4 Implement and Test Bloom Filters

Goal

- **Warmup:** (i) A hash factory mapping integers to a power-of-two range, and (ii) a `BloomFilter` class that takes (n, c) and stores bits in a packed bitmap (no boolean arrays). Build a 10,000-key membership set and, for targets $c \in \{0.01, 0.001, 0.0001\}$, report *Theoretical FP* and *Real FP* together with (R, k).
- **Extended:** Parse unique URLs, insert all N, and with policy $k = \lfloor 0.7R/N \rfloor$ vary $R \in \{2^{19}, 2^{20}, 2^{21}, 2^{22}\}$ to (i) report empirical FP and memory, (ii) plot FP vs. memory (use $R/8$ on the x-axis), and (iii) compare memory against a Python `set` and the theoretical $R/8$, with a brief comment.

Warmup

Setup Membership: 10,000 unique integers from [10,000..99,999]; test: 1000 non-members + 1000 true members. For each target c, I round R to a power of two; the "Theoretical c" below is computed *after* rounding R.

Target c	R (bits)	k	Theoretical c	Empirical c	Memory (bytes)
0.01	131,072	9	1.84141×10^{-3}	0.002	16,441
0.001	262,144	18	3.3908×10^{-6}	0	32,825
0.0001	262,144	18	3.3908×10^{-6}	0	32,825

Table 1: Warmup results (membership 10^4, test 1000+1000). Theoretical FP is the model prediction after rounding R to a power of two.

Extended

Setup I parsed the AOL file to unique URLs, inserted all N into a Bloom filter, and varied $R \in \{2^{19}, 2^{20}, 2^{21}, 2^{22}\}$ with the required policy $k = \lfloor 0.7R/N \rfloor$. Then it was evaluated on 1000 sampled members and 1000 random strings as negatives. For each (R, k), I report the *empirical* FPR and Bloom bitmap size (via `sys.getsizeof`), alongside the theoretical bit budget $R/8$. I also measured a Python `set` containing the same universe (constant across R).

R (bits)	k	Emp. FPR	BF bytes (measured)	Theory $R/8$ (bytes)
524,288	1	0.503	65,593	65,536
1,048,576	1	0.301	131,129	131,072
2,097,152	3	0.081	262,201	262,144
4,194,304	7	0.006	524,345	524,288

Table 2: Extended memory comparison.

Python `set` footprint (independent of R): 16,777,432 bytes.

Measurement note (Python set). For the Bloom bitmap, I reported the exact container size via `sys.getsizeof(bytearray)`, which equals $R/8$ plus a small fixed overhead. For the Python `set`, I intentionally reported the *container-only* size taken from `sys.getsizeof(set)`, which does *not* include the memory of the stored elements (e.g. strings) orheir internal allocations. This choice is conservative *in favor of the set*; the true deep footprint (for example, by `pimpler.asizeof` or summing `sys.getsizeof` over all elements) is much larger for

$N \approx 3.78 \times 10^5$ URLs. However, the Bloom vs. theory comparison is exact because it concerns only the bitmap bits.

Memory Usage Comparison

- *Bloom vs theory:* Measured bitmap size matches $R/8$ up to a small constant overhead of ≈ 57 bytes in every setting (measured $- R/8 = 57$).
- *Bloom vs Python hashtable:* At R=4,194,304 the Bloom bitmap uses 524,345 bytes vs. `set` at 16,777,432 bytes ($\sim 32\times$ smaller); the ratio widens further at smaller R.
- *Accuracy–memory tradeoff:* With $k = \lfloor 0.7R/N \rfloor$, empirical FPR falls from 0.503 to 0.006 as R increases, while memory scales linearly with R (the $R/8$ line).

fpr_vs_memory.png

(a) FPR vs memory ($R/8$).

Figure 2: Extended evaluation with policy $k = \lfloor 0.7\, R/N \rfloor$.

Conclusions Empirical FP decreases rapidly as R grows and tracks theory. The packed bitmap's measured size agrees with $R/8$ up to small object overhead, while a Python `set` is orders of magnitude larger for this N.

Appendix A: How to Run

Environment (Python $\geq$ 3.9).

1. **(Recommended) Create a virtual environment**

   ```
   python3 -m venv env

   # This is MacOS Command. Do equivalent if on windows.
   source env/bin/activate
   ```

2. **Install dependencies**

   ```
   python -m pip install --upgrade pip
   pip install -U numpy pandas matplotlib scikit-learn
   ```

Dataset If you have the AOL file, place `user-ct-test-collection-01.txt` in the working directory (or a `data/` subfolder). The Q4 script will attempt to detect it automatically when `--extended` is used. If the file is absent, the extended run is skipped.

Commands.

- **All results:** `python3 main.py --all`
- **Only Q1 (hash avalanche):** `python3 main.py --q1`
- **Only Q4 (Bloom tests):** `python3 main.py --q4` (add `--extended` to run the AOL dataset sweep if the file is present)

Outputs (where to find things).

- **Q1:** `outputs/q1/`
  - Heatmaps: `2univ_linear_heatmap.png`, `3univ_quadratic_heatmap.png`, `4univ_cubic_heatmap.png`, `murmurhash3_heatmap.png`
  - Deviation heatmaps: corresponding `*_dev_heatmap.png`
  - CSVs: `*_avalanche.csv`; summary: `summary.txt`
- **Q4:** `outputs/q4/`
  - Warmup table dump: `Results.txt` and `results.csv`
  - Memory reports: `memory.txt` (warmup), `extended_memory.txt` (AOL)
  - Extended sweep: `extended.csv` (AOL metrics)
  - Plots: `fpr_vs_memory.png`

Reproducibility. All scripts are deterministic given the fixed seeds listed in the paper's "Constants" section. Running the commands above will regenerate the CSVs and figures in the same paths.

Appendix B: Submitted Code Artifacts

Per the instructions, all code is provided as compressed archives rather than embedded listings here.

- **Source zip:** `src.zip` containing the files `main.py`, `q1_avalanche.py`, `q4_bloom.py`, `q1_heatmaps.py`, `q4_plots.py`.
- **Outputs zip:** `outputs.zip` containing all CSVs, text summaries, and figures produced by running the scripts.