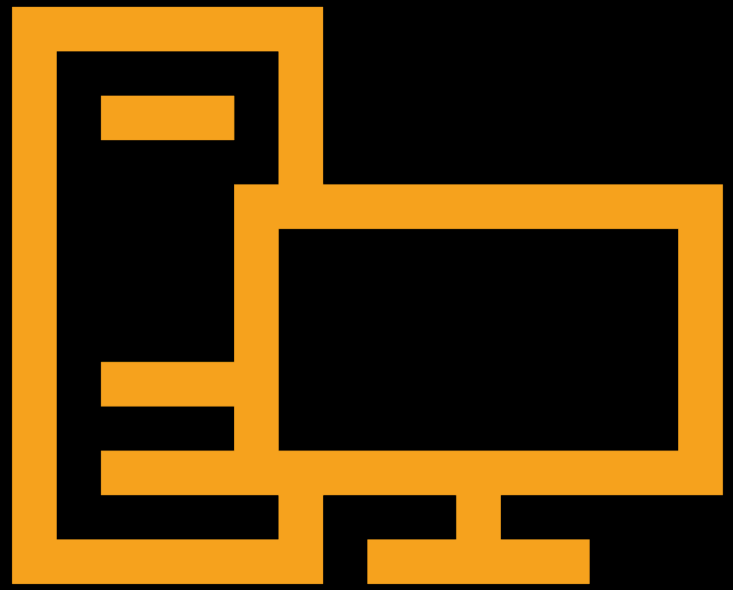
APPLIED ALGORITHMS

Lecture 9: Assignment 2, Probability, and Hashing



ADMIN

- Assignment 2 grades and comments pushed to your repo
- Mini-midterm due 10PM tomorrow

QUESTION

Which would you rather have:

- Two assignments: 1 assignment due next Wednesday, one assignment out on Wednesday and due the Wednesday after Spring Break (I would assume you'd work on it for ~1 week total: 3 days before break and 3 days after)
- Just one larger assignment due next Friday

SOME MINI-MIDTERM CLARIFICATIONS

Problem 2. Let s be the value of `SHIFT` used in your code. This means that there are 2^s buckets.

Assume that each of the 2^s buckets contains $O(n/2^s)$ elements.[3] In the External Memory model, how many I/Os does this algorithm take? (Please give your answer in terms of s, n, M, and B. If you need to make any assumptions—such as $n > B$—please state them explicitly.)

Answer should be parameterized by s. (Should not use the value of SHIFT in your code as a constant—I'm asking how the I/Os change depending on SHIFT)

or `SHIFT` to minimize the number of I/Os incurred
model?[4] Please give your answer in terms of n, M,
e.)

SOME MINI-MIDTERM CLARIFICATIONS

Problem 2. Let s be the value of `SHIFT` used in your cod
buckets.

Assume that each of the 2^s buckets contains $O(n/2^s)$ ele
model, how many I/Os does this algorithm take? (Please
n, M, and B. If you need to make any assumptions—such
explicitly.)

Solution.

Problem 3. What value should you use for `SHIFT` to minimize the number of I/Os incurred by the algorithm in the external memory model?[4] Please give your answer in terms of n, M, and B. (You may not need to use all three.)

Solution.

Asks you to use the external memory model to predict the best value of SHIFT. This may not line up with practice!

ANY OTHER MINI-MIDTERM QUESTIONS?

- Note that the best value of SHIFT may not be too interesting (depends on your implementation). I want you to implement the algorithm and see what works best for you!

ASSIGNMENT 2

- I know that this class is hard and that not everyone has the same background
- I am grading very generously on assignments because of this
- But I can't give partial credit if you don't hand anything in

- Please hand in answers to all questions, and hand in code!

ASSIGNMENT 2

- Resources for C
- Textbooks! (Very good.). On reserve (for 237) in library

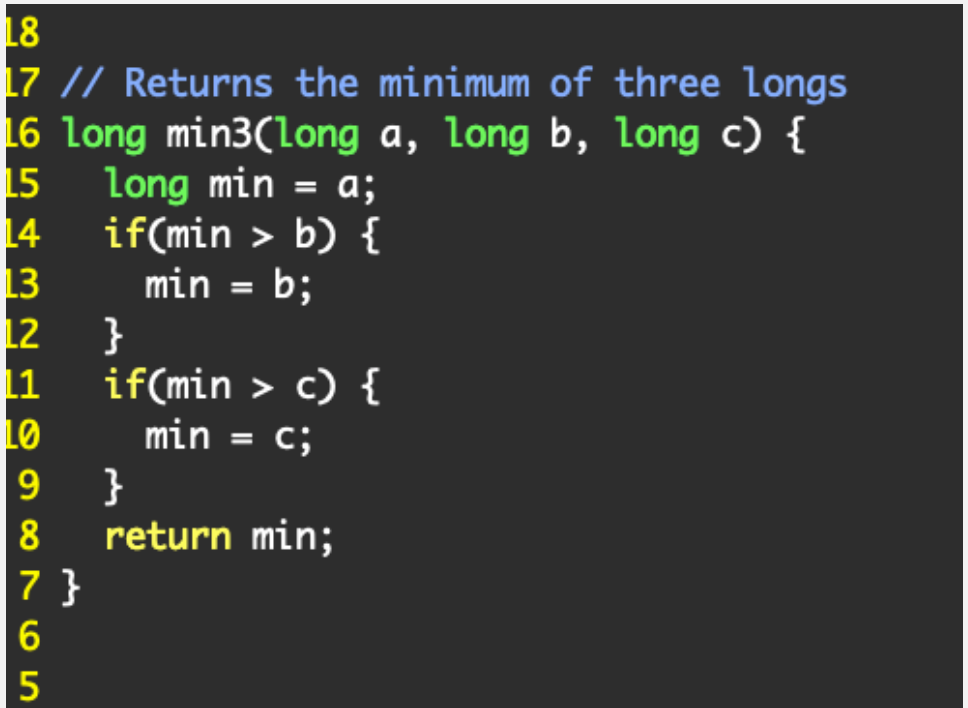
- Also: come talk to me, ask classmates, etc.
- I'm looking into more formal options (not sure what's possible)

ASSIGNMENT 2

- Code: lots of great ideas
- Unfortunately not all of them led to speedups
  - In some cases two people implementing the same idea got different results
- Some particularly effective ideas:
  - Iterative version (no recursion, track the stack manually)
  - Space-inefficient base case
  - Keep old rows of table to avoid recomputations

ASSIGNMENT 2

- By far most expensive operation is min-of-3
- I believe that this is the best way to do it:



```
18
17 // Returns the minimum of three longs
16 long min3(long a, long b, long c) {
15   long min = a;
14   if(min > b) {
13     min = b;
12   }
11   if(min > c) {
10     min = c;
 9   }
 8   return min;
 7 }
 6
 5
```

ASSIGNMENT 2

- By far most expensive operation is min-of-3
- I believe that this is the best way to do it:
- That said, many of the fastest submissions did not do it that way and the assembly does not do anything special so who knows

ASSIGNMENT 2

I/Os for edit distance (space-inefficient algorithm, no backtracking)

- We fill out the table one row at a time
- How many I/Os does it take to fill out a row?
  - For each B elements, need to bring in at most 6 blocks
- So a row of length n takes $O(\frac{n}{B} + 1)$ I/Os if $M \geq 6B$
- Overall I/Os?
  - m rows, so $O(m\frac{n}{B} + m)$ I/Os

ASSIGNMENT 2

- Problem 3 (Extra credit): edit distance in $O\left(\frac{nm}{MB} + \frac{n+m}{B}\right)$ I/Os
- Idea: tiling!
- What size do we want our tile to be?
  - Want to be able to solve a tile-sized subproblem in cache
  - If two strings have size $O(M)$, can solve in $O(\frac{M}{B})$ I/Os
- Visualization on board

ASSIGNMENT 2

- Problems 4,5,6: Dynamic programming
- Tip for dynamic programming: you want to answer three questions
  - What does each entry of the table represent?
  - What is the base case? What exactly is the value of each base case?
  - How can I fill in each entry of the table using the other entries of the table and the input?

ASSIGNMENT 2

- Problem 4: Bookshelf problem
- Let cost(i, j) be the cost of putting book i on shelf j (can calculate in $O(k)$ time)
- Entry (i, j) of the table represents the cost of the minimum solution that places books $b_1, \ldots, b_i$ on shelves $s_1, \ldots, s_j$, where book b_i must be on shelf s_j
- Base cases: $(i, 1)$ for all i is $\sum_{i'=1}^{i} cost(i', 1)$; (j, j) for all j is $\sum_{j'=1}^{j} cost(j', j')$

Can skip

ASSIGNMENT 2

- Problem 4: Bookshelf problem
- Let cost(i, j) be the cost of putting book i on shelf j (can calculate in $O(k)$ time)
- Entry (i, j) of the table represents the cost of the minimum solution that places books $b_1, \ldots, b_i$ on shelves $s_1, \ldots, s_j$, where book b_i must be on shelf s_j
- To place book b_i on shelf s_j, need b_{i-1} to be on s_j or s_{j-1}
- Entry (i, j) = min ($(i-1, j), (i-1, j-1)$) + cost (i, j) (need $i \geq j$)

ASSIGNMENT 2

- Problem 4: Bookshelf problem
- Can fill in table row-by-row. Each table entry takes $O(k)$ time
- Problem 5: Can we use linear space?
- Yes. Just keep two rows, as in edit distance

ASSIGNMENT 2

- Problem 6: Recover actual book placements in linear space
- Idea: Hirschberg's!
- Every assignment of books $b_1, \ldots b_i$ to shelves $s_1, \ldots s_j$ can be split into two parts for some k:
  - An assignment of $b_1, \ldots, b_k$ to $s_1, \ldots, s_{j/2}$
  - An assignment of $b_{k+1}, \ldots, b_i$ to $s_{\frac{j}{2}+1}, \ldots, s_j$
- Can find the cost of every possible split in $O(nmk)$ time, $O(m)$ space. Need to run second half backwards so that all assignments are given with one call
- Remaining analysis is same as Hirshberg's

ANY QUESTIONS?

- Assignment 2, mini-midterm?

- I have office hours today 3-4

CUCKOO HASHING

- Pagh, Rodler 2005
- Lookup time is $O(1)$ in the *worst case!*
- Insert/delete is $O(1)$ in expectation
  - $O(\log n)$ worst case (same caveat as before)

CUCKOO HASHING INVARIANT

- Have two hash functions h_1, h_2
- Table of size cn with $c = 2$
- Invariant: item x is stored either at slot $h_1(x)\%\ cn$, or at slot $h_2(x)\%\ cn$
- I'll talk about inserts in a second
- How do we query?
- How much time does that take?

CUCKOO HASHING INSERTS

- Let's put a new item x into our hash table. How?
- Easy case: if slot $h_1(x)\%\ cn$ or $h_2(x)\%\ cn$ is free, can just store x
- What if they're both full?
- Answer: pick one of the slots. Kick the item stored there out; store it using its other hash
  - Hence "cuckoo"

Cuckoos kick other birds' eggs out of the nest, replacing them with their own

CUCKOO HASHING EXAMPLE

Table:

CUCKOO HASHING EXAMPLE

Table:

Insert new item: **33**. $h_1(33) = 0, h_2(33) = 3$

CUCKOO HASHING EXAMPLE

Table:

CUCKOO HASHING EXAMPLE

Table:

Insert new item: **49**. $h_1(49) = 3, h_2(49) = 1$

CUCKOO HASHING EXAMPLE

Table:

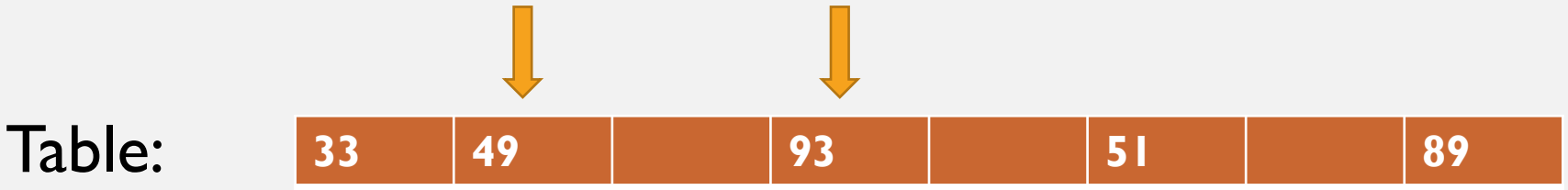
Insert new item: 49. $h_1(49) = 3, h_2(49) = 1$

CUCKOO HASHING EXAMPLE

Table:

Let's choose to kick out 5, replacing it with 49

CUCKOO HASHING EXAMPLE



Now we need to find a place for 5

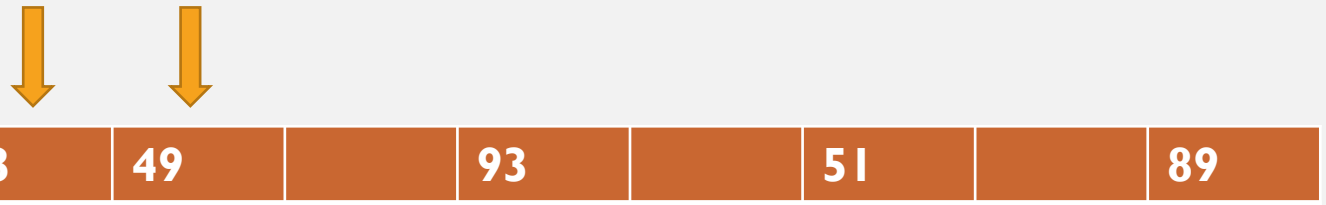
CUCKOO HASHING EXAMPLE

Table:

Reinsert item: 5. $h_1(5) = 0, h_2(5) = 1$

CUCKOO HASHING EXAMPLE

Table:



We have to kick out 33. (Don't want to loop back)

CUCKOO HASHING EXAMPLE

Table:

Reinsert item: 33. $h_1(33) = 0, h_2(33) = 2$

CUCKOO HASHING EXAMPLE

Table:

5	49	33	93		51		89

Reinsert item: **33**. $h_1(33) = 0, h_2(33) = 2$

CUCKOO HASHING EXAMPLE

Table:

Done!

CUCKOO HASHING ANALYSIS

- Insert: expected number of swaps is $O(1)$
- Largest number of swaps is $O(\log n)$

- Wait a minute…does this always work?
  - No.

CUCKOO HASHING FAILURE

Table:

$$h_1(33) = 0, h_2(33) = 1$$
$$h_1(5) = 0, h_2(5) = 1$$
$$h_1(17) = 0, h_2(17) = 1$$

We can't maintain the invariant!!!

CUCKOO HASHING FAILURE

- [PR'05]: the probability of failure is only $O(1/n)$
- What do we do if we fail???
  - Pick new hash functions and start from scratch
  - (Ouch)

CUCKOO HASHING

- Advantages?
  - Great worst-case performance on queries
  - Only two cache misses on queries
  - Fairly simple
- Disadvantages?
  - Rebuilds are a huge issue!
  - Two cache misses can be much worse tha[illegible]inear probing
  - On inserts, every swap of an ele[illegible]ent is another cache miss
  - Space usage is not great

You can avoid this by storing a constant number of elements in each slot (say 4)

EXPECTATION

- Expectation is like a weighted average, in the context of probability

$$\text{Expectation} = \sum_{outcomes\ o} Probability(o) * cost(o)$$

- Example: I roll a die; if there's a 6 I win $60. What is my expected winnings?
- $0 * \left(\frac{1}{6}\right) + 0 * \left(\frac{1}{6}\right) + 0 * \left(\frac{1}{6}\right) + 0 * \left(\frac{1}{6}\right) + 0 * \left(\frac{1}{6}\right) + 60 * \left(\frac{1}{6}\right) = 10$

LINEARITY OF EXPECTATION

- If a random variable $X = X_1 + X_2$, then $E[X] = E[X_1] + E[X_2]$
- This is ALWAYS true. Don't need X_1 and X_2 to be independent!

HASH FUNCTION

- We need a function that will decide what slot each item goes in
- Generally: start with a function with large output, then take `% tablesize`
  - Can lead to slight issues unless output is very large—be careful!
- What do we want out of a hash function?
  - Quick to compute
  - Small space to store
  - Random enough that we get small chains/small buckets

HASH FUNCTION DISCUSSION

- Fully-random hash functions are very unreasonable
  - Take forever to evaluate, and/or take tons of space
- Practice often uses simpler functions

HASH FUNCTIONS IN JAVA

- Does anyone know how Java .hashCode() hashes a 64 bit Long?
  - (What is your guess?)
- Answer: `return x ^ (x >> 32);`
- Is this going to work well for hashing?

MULTIPLY-SHIFT HASHING

- The hash you used on Mini-midterm 1
- How fast is it?
- How good is it?
  - Answer: pretty good! Let's say the output is from $0, \ldots, n-1$
  - If you choose the integer you multiply by at random, $\Pr\big(h(x) = h(y)\big) = 1/n$
  - Let's look at an element that hashes to a given value. How many collisions in expectation if we hash n elements?
  - $\sum 1/n = 1.$ (wow!)

MULTIPLY-SHIFT HASHING

- Expectation is that one extra element hashes to each bucket
- Why is this not always good enough?
  - Average is not always good!
  - For example: might have one bucket of size $\sqrt{n}$, rest of size 1
  - Good on average, but some queries are very slow!

MURMURHASH

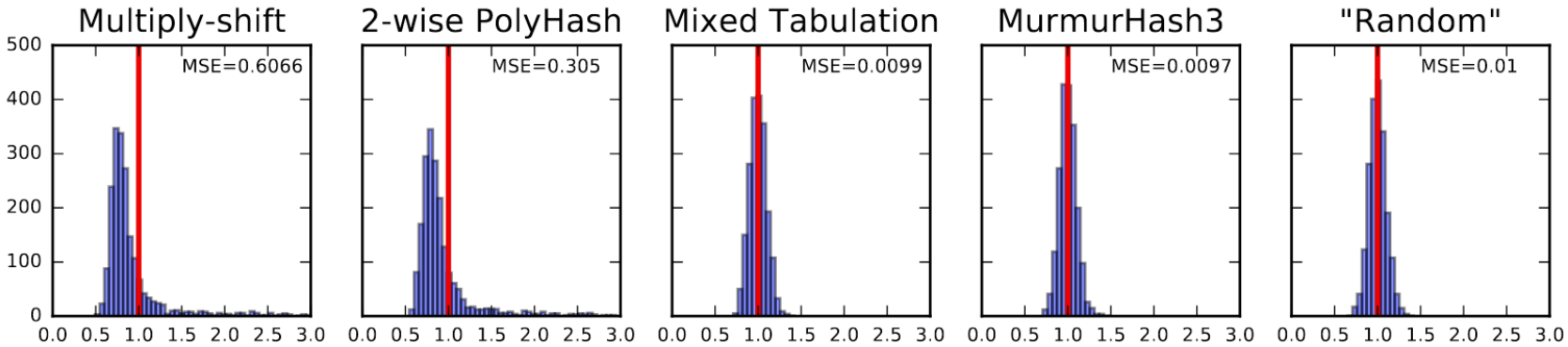
- Popular implementation that does a little more work than multiply-shift
- Two basic operations: XOR, and "rotate"
  - Rotate is like a shift, but when bits "fall off" they are replaced on the other side
  - Can be implemented with two shifts and a bitwise OR

- Code

MURMURHASH

- Compared to multiply-shift it's definitely slower
- Is it better???
- Theoretically: doesn't necessarily even give constant-sized buckets in expectation

MURMURHASH



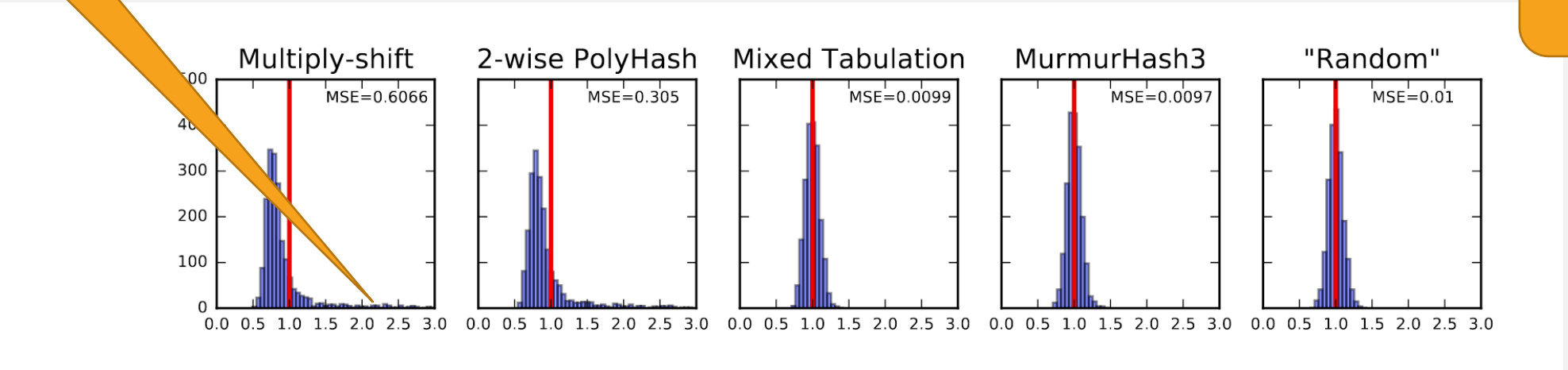
Average of *square* of bucket sizes. Data is an intentionally bad (albeit reasonable) case

From "Practical Hash Functions for Similarity Estimation and Dimensionality Reduction" by Dahlgaard, Knudsen, Thorup NeurIPS 2017

MURMURHASH

Long tail!
(= bad)

Murmurhash looks exactly like random

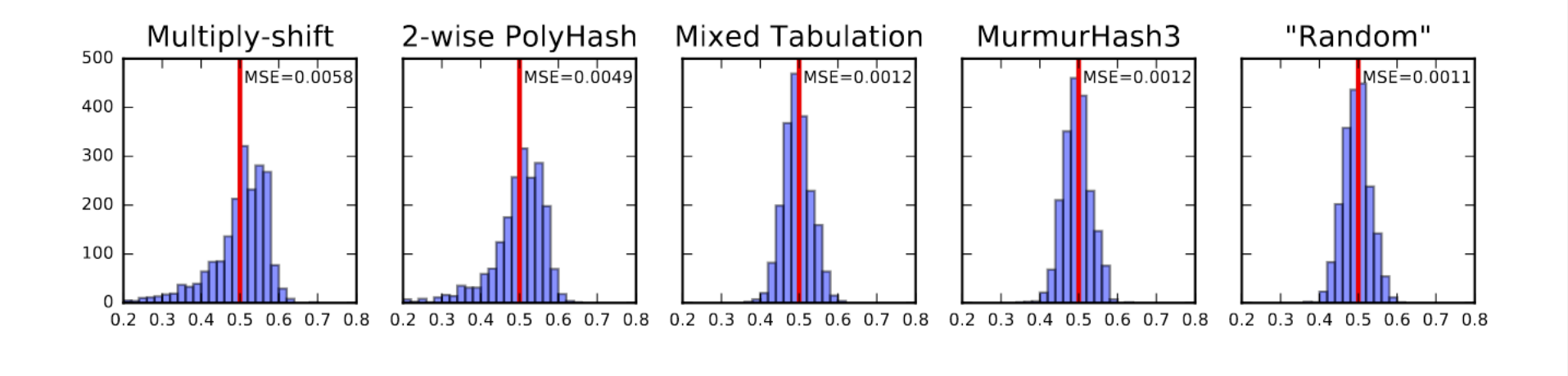


Average of *square* of bucket sizes. Data is an intentionally bad (albeit reasonable) case

From "Practical Hash Functions for Similarity Estimation and Dimensionality Reduction" by Dahlgaard, Knudsen, Thorup NeurIPS 2017

MURMURHASH

- Does this actually impact anything?
- From same paper: Yes. Let's say we use hashing to estimate how many elements two sets share in common



MURMURHASH

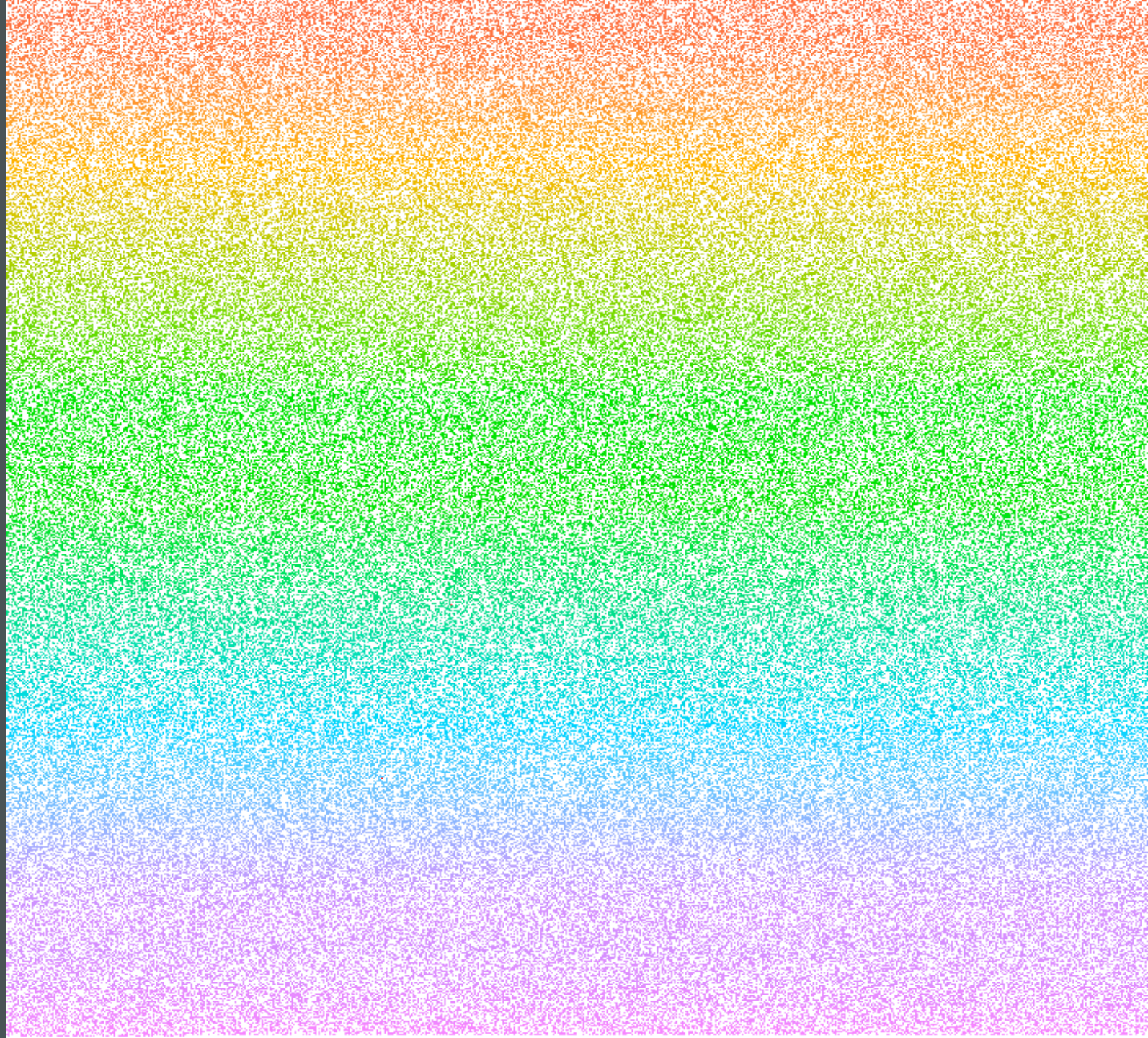
- Much more resilient than multiply-shift to more-difficult statistical tests (beyond average case)
- One more example: let's say we hash "number strings": `"1", "2", … "216553"`
- (Cool experiment from https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed)
- (I wouldn't normally cite stackexchange but this is really cool)

SDBM (A POPULAR HASH)

- You can see big clusters and big gaps when hashing number strings!
- (SDBM works pretty well on most inputs though)

MURMURHASH

- Doesn’t have this issue



ROTATE AND XOR

- Many other PRNGs and hashes under this paradigm
- Is moving bits around like this randomly enough?
- No! Example: "SuperFastHash"
- From https://softwareengineering.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed
- If you hash all English words, Murmurhash has 6 collisions
- SuperFastHash has 85

CAN WE DO BETTER?

- All of these hash functions rely on the input being "random" enough to do consistently well
- What if we REALLY want evenly-spread elements?
- What if the security of our application depends on evenly-spread elements?
  - You may want to be resilient to an attacker
  - I.e. timing attacks

CRYPTOGRAPHY

- We know how to encode stuff pretty well
- It's basically impossible to decode---information about the output tells us nothing about the input whatsoever
- Doesn't that mean it's about as "random" as it can get?
  - Yep.
  - So why don't we use it?

CRYPTOGRAPHIC HASH FUNCTIONS

- Easily obtainable, work very well
- Important for security
- Downside: generally high cost
- Options?
  - MD5: broken
  - SHA1: broken
  - SHA256, SHA3: OK for now (many other options: BLAKE2 etc.)

SPEED COMPARISON (32 BIT)

Benchmarks

The benchmark uses SMHasher speed test, compiled with Visual 2010 on a Windows Seven 32-bit box. The reference system uses a Core 2 Duo @3GHz

Name	Speed	Quality	Author
xxHash	5.4 GB/s	10	Y.C.
MurmurHash 3a	2.7 GB/s	10	Austin Appleby
SBox	1.4 GB/s	9	Bret Mulvey
Lookup3	1.2 GB/s	9	Bob Jenkins
CityHash64	1.05 GB/s	10	Pike & Alakuijala
FNV	0.55 GB/s	5	Fowler, Noll, Vo
CRC32	0.43 GB/s †	9	
MD5-32	0.33 GB/s	10	Ronald L.Rivest
SHA1-32	0.28 GB/s	10	

Note †: SMHasher's CRC32 implementation is known to be slow. Faster implementations exist.

From https://github.com/Cyan4973/xxHash

THURSDAY

- Start applications of randomness to small-space data structures
- In meantime: work on midterm.
- Good luck!