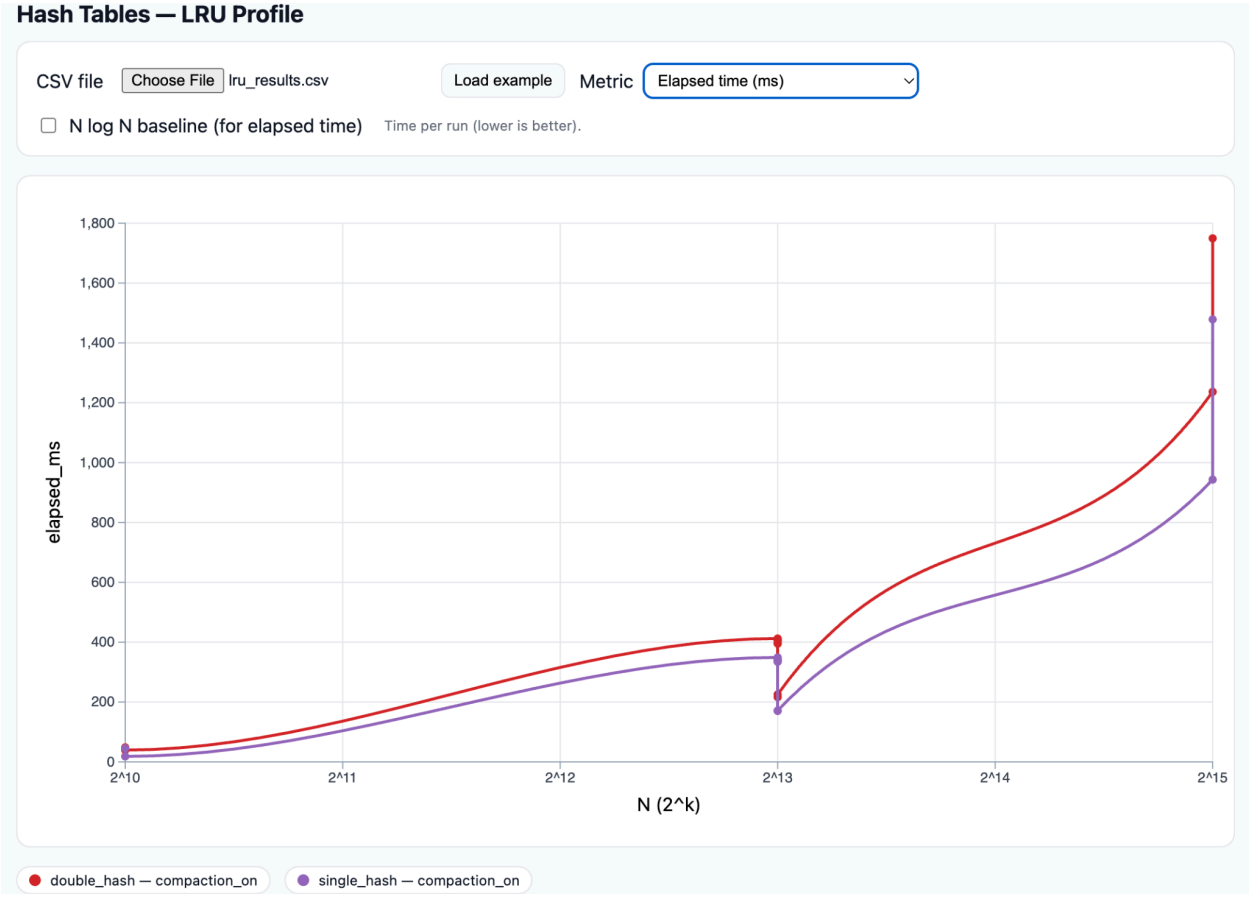
Initial Thoughts:

Given what we know about hashing on both single probing and double hashing and the runtimes for both inserts and deletes being at best O(1) if there is no initial collision and O(n) if there is a collision and possible subsequent collisions, we can infer that the only factor that would separate the two's real world runtime's is the number of operations that are needed. Since single probing only inserts at the next available position, it tends to create clusters until the table is, in theory, full. At the same time, double hashing jumps all over the table, in which the likelihood to create a mass amount of clusters is greatly decreased at the cost of more operations than that of single probing as the table becomes more and more full.

With all this in mind, it can be expected to see that single probing will be faster at the lower N's even when you take into account a heavy delete centered table, but as N grows, double hashing will more than likely overtake it since while it's slower as the table gets more full, the deletes come into play and free up the table, which can result in less operations in theory for double hashing if it were to get lucky everytime and had at most 1 collision.



After running the program with N being 2^15, we can observe that our initial hypothesis was only half correct; the graph shows that double hashing, no matter the value of N, is always more, but it is apparent that between 2^13 and 2^14, the gap starts increasing at a considerable pace, with the greatest being right before 2^15. This can be attributed to the fact that there possibly were not enough deletes to free use space to allow double hashing to get fewer collisions each insert. As for the seemingly random drops in elapsed at (N)2^13 and (N)2^15, this could just be that these values are arguably more stable integers and can be operated with at a greater capacity, therefore compaction might occur at a greater strength here.

Timing and Costs:

Results for Single and Double Probing:

```
==== SINGLE PROBING ====
   42598 table size:
    3298 cells marked as deleted.
   32768 active cells.
    6532 available elements.
   32768 maximum number of values in the table ever.
 9479178 total probes.

  308127 inserts.
  275359 deletes.
       0 lookups.
       0 full scans.
      14 compactions.

      15% ratio of available elements.
      76% load factor.
      84% effective load factor.
       7% tombstone fraction.

16.2458 average number of probes (single probing, compaction on).
```

```
==== DOUBLE PROBING ====
   42598 table size:
    5547 cells marked as deleted.
   32768 active cells.
    4283 available elements.
   32769 maximum number of values in the table ever.
 5125071 total probes.

  308127 inserts.
  275353 deletes.
       0 lookups.
      36 full scans.
      17 compactions.

      10% ratio of available elements.
      76% load factor.
      89% effective load factor.
      13% tombstone fraction.

8.78363 average number of probes (double probing, compaction on).
```

After comparing the average number of probes of single probing (16.2458) and double probing (8.78363) to the graphed values, it's interesting to see that the numerical values of average probes do not tell the whole story and should not be trusted as true predictors of which is actually faster. This can be easily explained when observing that the number of full scans and compactions for single probing were 0 and 14 respectively, while double probing demonstrated 36 full scans and 17 compactions, a significant jump from that of single probing. This can point to double hashing, still expressing that it is a costly operation to ensure a smaller number of clusters and cluster size.

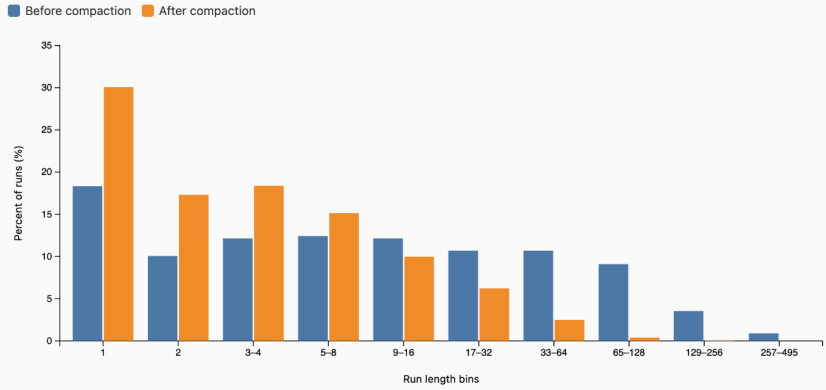
Spatial locality also plays a big part in terms of cost, specifically, the memory locality of the table itself. Single probing operates by taking single "steps" each collision, making the spatial locality accessing the direct neighbor address, making it the fastest possible travel, while double hashing can "jump" anywhere from a single memory address over to N addresses over. This also contributes to the cost of performing double hashing in the long run as the table size both increases and begins to get full.

In terms of compactions, it overall created more overhead, but the difference in the number of compactions was small in the grand scheme of things, specifically 14 compactions vs 17 compactions, and really doesn't have a massive effect. Since the percentage of tombstones single probing accumulated was 7% while double probing was 15%, this points to a stressed structural integrity.

For the latency and throughput of single probing and double probing, the story also remains the same, and can be expressed by calculating their respective values. For single probing, latency is 583480 ops_total / 1500ms, which equals 389 ops/ms (throughput). Double probing shows 583480 ops_total / 1750ms (throughput), which comes out to 333 ops/ms. For latency, the single probing is 1500ms / 583480 ops_total, which is 0.00257 ms/op, while double probing has 1750ms / 583480 ops_total, which is 0.00299 ms/op. In both latency and throughput, double probing boasts worse results, which is another factor that points to it being an overall costly operation.

Average load factor was surprising the same value for both, at 76%, which doesn't point to any indication of it having played a role in the timings, but when observing the values for effective loads, single probing has an 84% peak while double probing shows to have had an 89% peak, being yet another contributing factor to the overall results of both runs.

Before and After Compactions:



Before compaction — summary	
Runs counted	1,441
Bits scanned	42,498
Mean run length	28.02
Median run length	7
p90 run length	82
p95 run length	124
Max run length	495

After compaction — summary	
Runs counted	5,240
Bits scanned	42,598
Mean run length	6.25
Median run length	3
p90 run length	15
p95 run length	24
Max run length	149

Tip: The parser ignores all non-0/1 characters in the map blocks. It reads the first non-empty block after the header as “before” and the second as “after”.

Compaction’s (orange) role in the distribution of work results in the majority of work having been done with only one run, specifically in bin 1, shown in the bar graph above, boasting a high 30% of all runs. On the other hand, before compaction (blue), we saw a far more even distribution of work across the program’s runtime, with below 20% for bin 1 and an average between 10% and 15% for all subsequent bins. This suggests that compaction and the structure as a whole play a key role in the program’s runtime performance.

Before compaction had run up to 495 probs long, while after compaction saw that decrease to around 149 probes, which plays a huge role in performance and shows that it did indeed break up a considerable amount of clusters. This tracks with the results since tombstones’ and effective loads’ percentages went down, and improved performance considerably.