Solutions

Marking: **M1** a correct method, **A1** the correct answer, **B1** an independent mark.

Each answer shows the steps, then the **result**.

2.1

bucket 5: 15, 25, 35; bucket 7: 7, 17

the longest chain has 3 keys

2.2

14 → slot 0; 21 → 0 is taken → slot 1

9 → slot 2

16 → 2 is taken → slot 3

2.3

now 600/800 = **0.75**

after: 1600 buckets, 601/1600 ≈ **0.38**

2.4

only 9 of the 1000 buckets are ever used

each chain holds about $n/9$ names

so a lookup is $\Theta(n)$, not $O(1)$

3.1

(a)

```
def has_pair(a, target):
    seen = set()
    for x in a:
        if target - x in seen:
            return True
        seen.add(x)
    return False
```

a set of values seen so far; test `target - x` before adding `x`; one pass

(b) 3: need 3, not seen; 8: need −2; 5: need 1, not seen yet; 1: need 5, which is seen, so `True`

(c) pairs: $n(n-1)/2 \approx \mathbf{5 \times 10^9}$; the set version: about 10^5 steps

3.2

(a) 21 hashes to slot 0; slot 0 is now empty, so the search stops there and says "not found", although 21 is in slot 1

(b) mark slot 0 as deleted instead of empty; a search steps past a tombstone, and an insert may reuse it

(c) tombstones make probe sequences long, so searches slow down although few keys are stored; rebuilding removes them

3.3

(a) $10^6/0.75 \approx 1.33 \times 10^6$ buckets; ×8 B ≈ **10.7 MB**

(b) every URL lands in one chain, so each lookup is $O(n)$ and the server slows to a crawl; use a keyed, random hash (e.g. SipHash with a secret seed), so attackers cannot predict collisions

(c) hash table: about 1 step per lookup on average; tree: about $\log_2 10^6 = 20$ steps, but sorted order and a guaranteed worst case

(d) a hash table for the lookups, which happen all the time; the daily sorted list can be made by sorting that site's keys once a day, in $O(k \log k)$, which is cheap; a tree would slow every lookup to save one daily job

exercises.lol exercises.com.cn