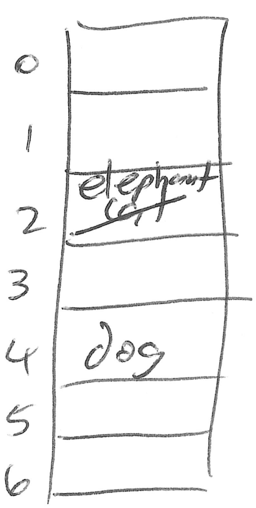
Hash Tables

Previous Data Structs: Insert, Delete, Search $\Rightarrow O(\lg n)$
w/balanced binary search tree (AVL)

hash table ~~=~~ is an array. Most simple version
max n items $\Rightarrow$ size n.

hash function $\Rightarrow$ input anything we would store
Output integer in btw 0 and $n-1$
(many output are fixed length bitstrings)
many-to-one (multiple inputs could map same output)
good hash function is such that the prob
$H(x) = H(y)$ is roughly $\frac{1}{n}$ in all cases
where $x \neq y$.
compute quickly!

Insert x, calculate $H(x)$, put x in index $H(x)$
Search x, calculate $H(x)$, go to index and see if x is there.



$H(cat) = 2$
$H(dog) = 4$
$H(elephant) = 2$
↳ PROBLEM as written ~~elep~~ elephant
will overwrite ~~cat~~ !!!
"eat"

Collision when 2 items have
same hash value.

LOSSY

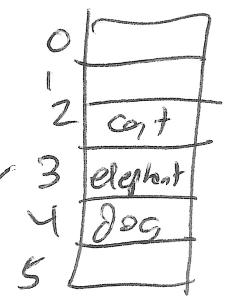
Strategies to keep all data

1. Linear Probing
2. Quadratic Probing

] effectively don't support delete.

* 3. Separate Chaining Hashing

→ if we get a collision on insert at index i, then go to $i+1$, $i+2$, ... 0, 1, 2, .. until an empty slot is found



H(elephant) = 2 → full
→ 3

Search? ⟹ slower, now we need to ~~do~~ go until 1st empty slot to prove something isn't in table.
Deletion becomes really tricky

→ CLUSTERING is an issue w/ linear probing
Probability a cluster grows increases quickly.
Run-time dependent cluster sizes.

Quadratic Probing

if $H(x) = i$ indexes I look at to ~~insert~~ insert

are $i, i+1, i+4, i+9, i+16, i+25, \ldots$ on

1 3 5 7 9

$idx \mathrel{+}= (2 * i + 1)$ how to advance idx you're looking at.

$idx \mathrel{\%}= n;$

With quadratic probing strategy what if we repeat table slots quickly?

length of table = prime #

odd prime

$idx, idx+1, idx+4, idx+9, \ldots \quad idx+k^2 \quad \% \quad p$

Prove that 1st $\frac{p-1}{2}$ items on list are unique

Pf by contradiction

assume to the contrary that and $0 \le i < j \le \frac{p-1}{2}$

$$\cancel{idx} + i^2 \equiv \cancel{idx} + j^2 \pmod{p}$$

$$(i^2 - j^2) \equiv 0 \pmod{p}$$

$$(i-j)(i+j) \equiv 0 \pmod{p}$$

$$p \mid (i-j)(i+j)$$

$$\Rightarrow \underbrace{p \mid (i-j)}_{\text{not poss}} \quad \text{or} \quad p \mid (i+j)$$

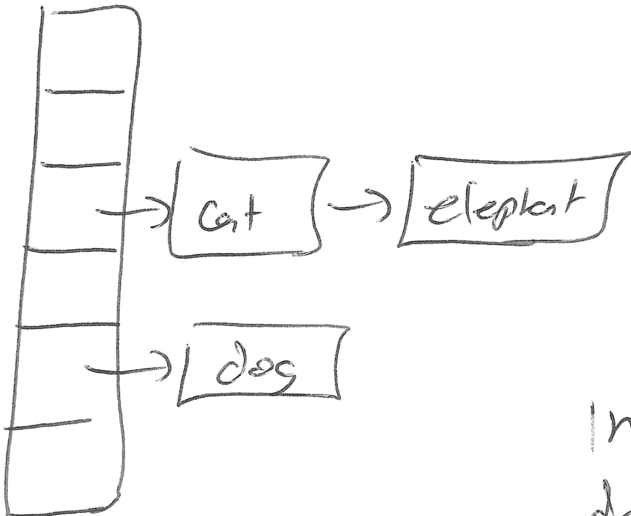
not poss
is neg
but $> -p$.

$i+j > 0$
$i+j \le p-2 < p$
not poss

$\boxed{\text{if } p \in \text{Prime} \quad p \mid (ab) \quad \text{then } p \mid a \vee p \mid b}$

w/ Quadratic probing make table at least twice as big as max # ~~ele~~ elements

Linear Chaining Hashing



w/good hash func max list size is expected to be $O(\lg(\lg(n)))$, I Think

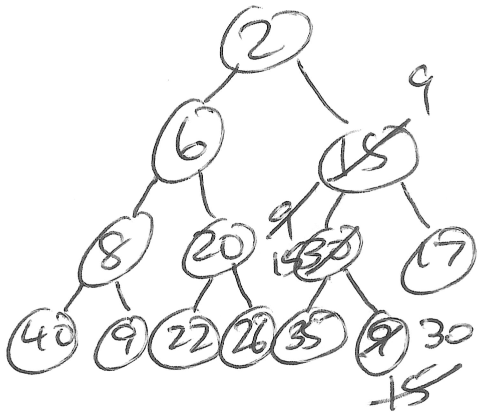
Insert to front

delete

Binary Heaps

① Insert
② Delete Min } O(lg n) time

Complete Binary Tree adheres to the heap node property



Complete bin tree:
all levels completely filled in except last, last must L → R.

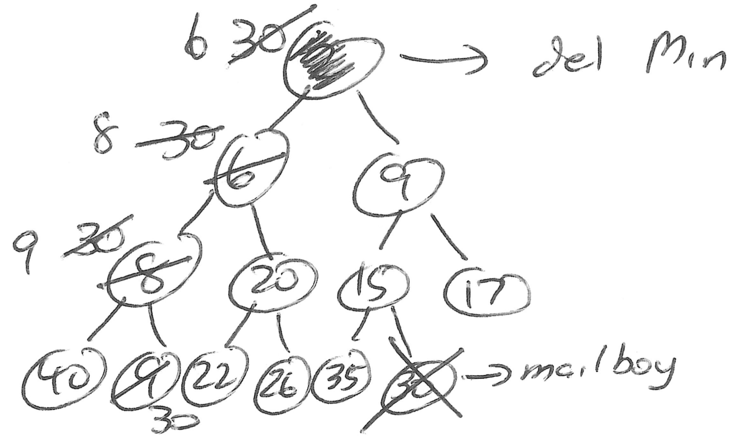
heap node prop: for each node it's the minimum value in its subtree

① Insert
② Delete Min
③ Create Heap from n unsorted items

Insert (x)

1) Place x in 1st empty slot, maintaining complete tree property. O(lg n)

2) Call PercolateUp(x).



delMin

1) locate min save will return
2) copy val last slot into root
   maintains complete bin tree structure.
3) PercolateDown (root)

Store

0	1	2	3	4	5	6	7	8	9	10	11	12
	6	8	9	9	20	15	17	40	30	22	26	35

root idx → 1

leftchild(idx) → 2 * idx

rightchild(idx) → 2 * idx + 1

parent (idx) → idx/2