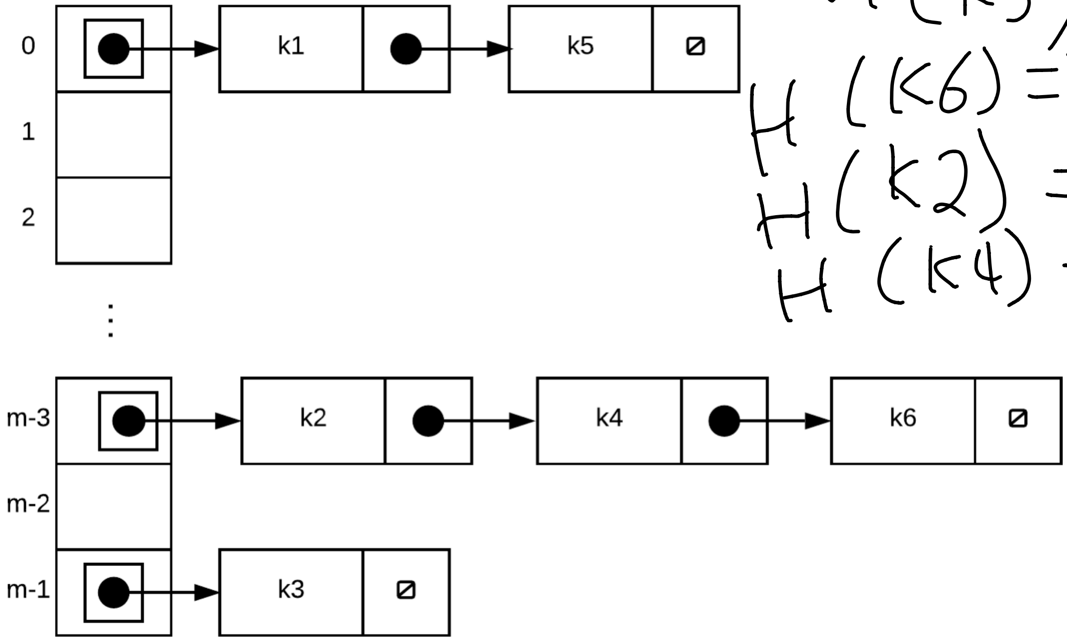
Hash Tables - Cost?

n - number of key-value pairs stored in hash table
m - number of slots in the array
U - universe of all possible keys

$H(K1) = \emptyset$
$H(K5) = \emptyset$
$H(K6) = m-3$
$H(K2) = m-3$
$H(K4) = m-3$



Chaining:

insert(k,v)

- find the correct linked list
- search for k within the linked list,
- add new node or modify existing if k is found

search(k)

- find the correct linked list
- search for k within the linked list

delete(k)

- find the correct linked list
- search for k within the linked list
- remove a node from list

Worst case: $\theta(n)$

But not likely to occur (every item would have to collide)

Average case:

SUHA - **S**imple **U**niform **H**ashing **A**ssumption

Any key is equally likely to hash to any slot

Consequence: Expected number of items in each bucket will be the same.

Let alpha represent the load factor which is also the expected number of items in each bucket under SUHA

alpha = n/m

For us to search unsuccessfully would involve searching to the end of the linked list. Thus, the expected length of the linked list determines how many elements we need to look through

The expected length of each list = alpha

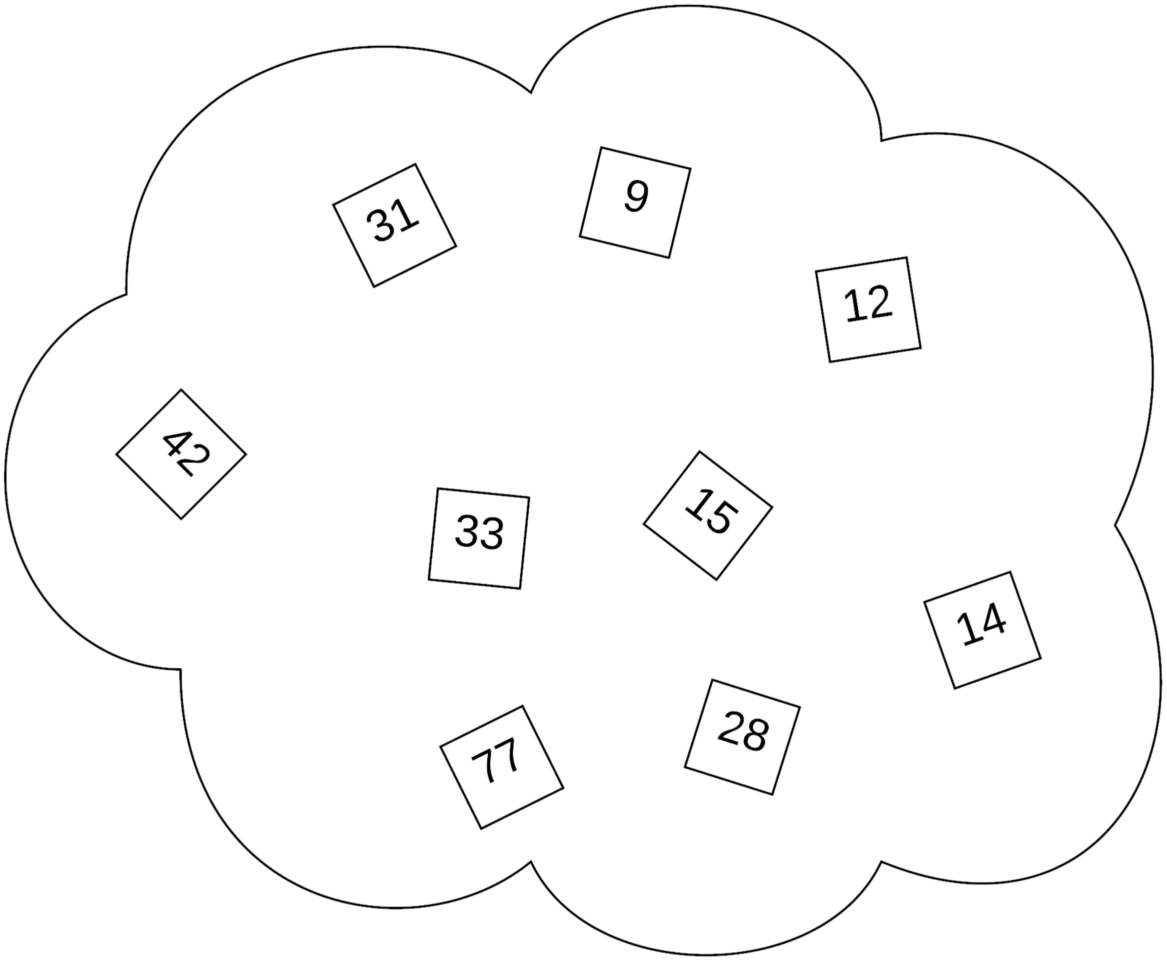
Hash function:

\

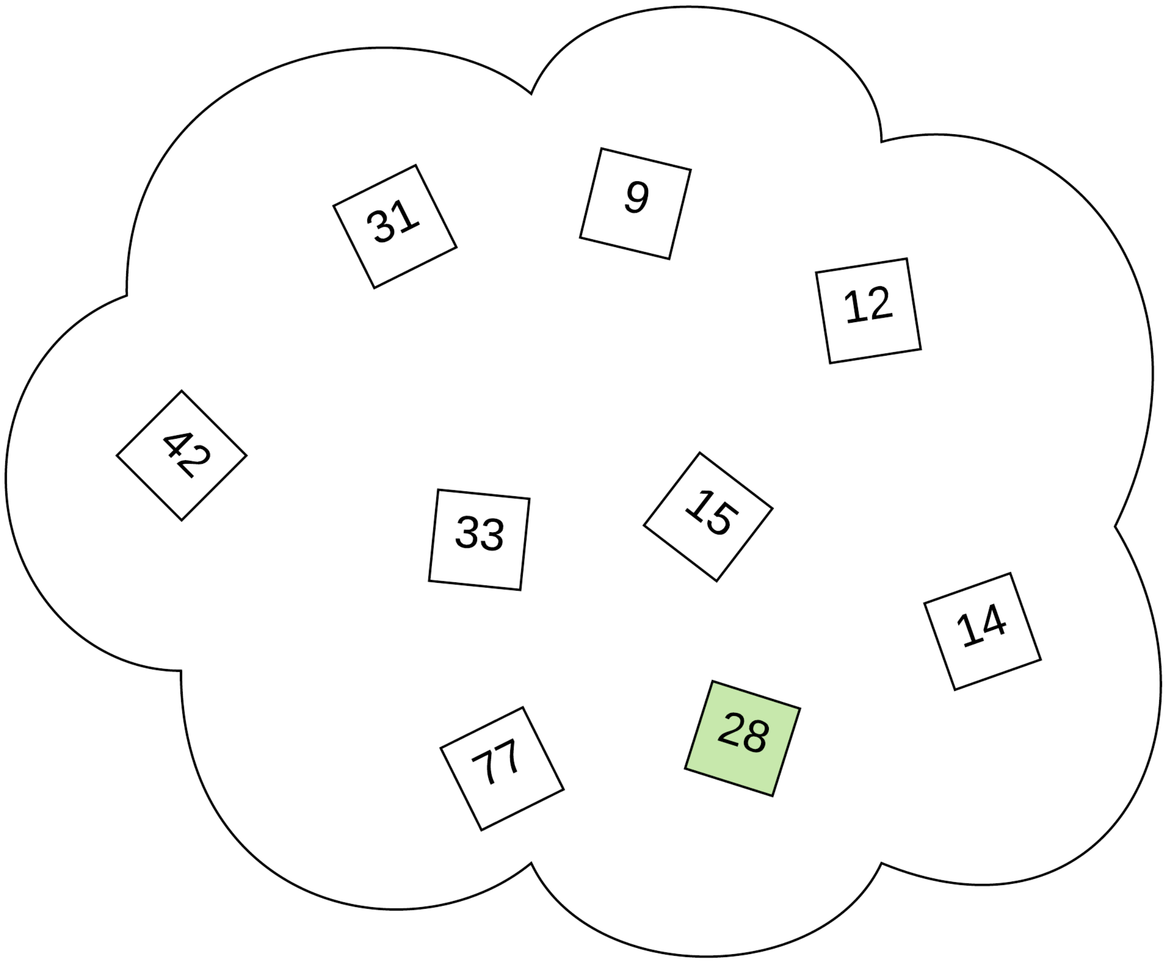
- hash(x) - efficient to compute
- hash(x) - every bit of x is used for calculating hash value
- hash(x) - spreads out the value (over all valid indexes evenly)

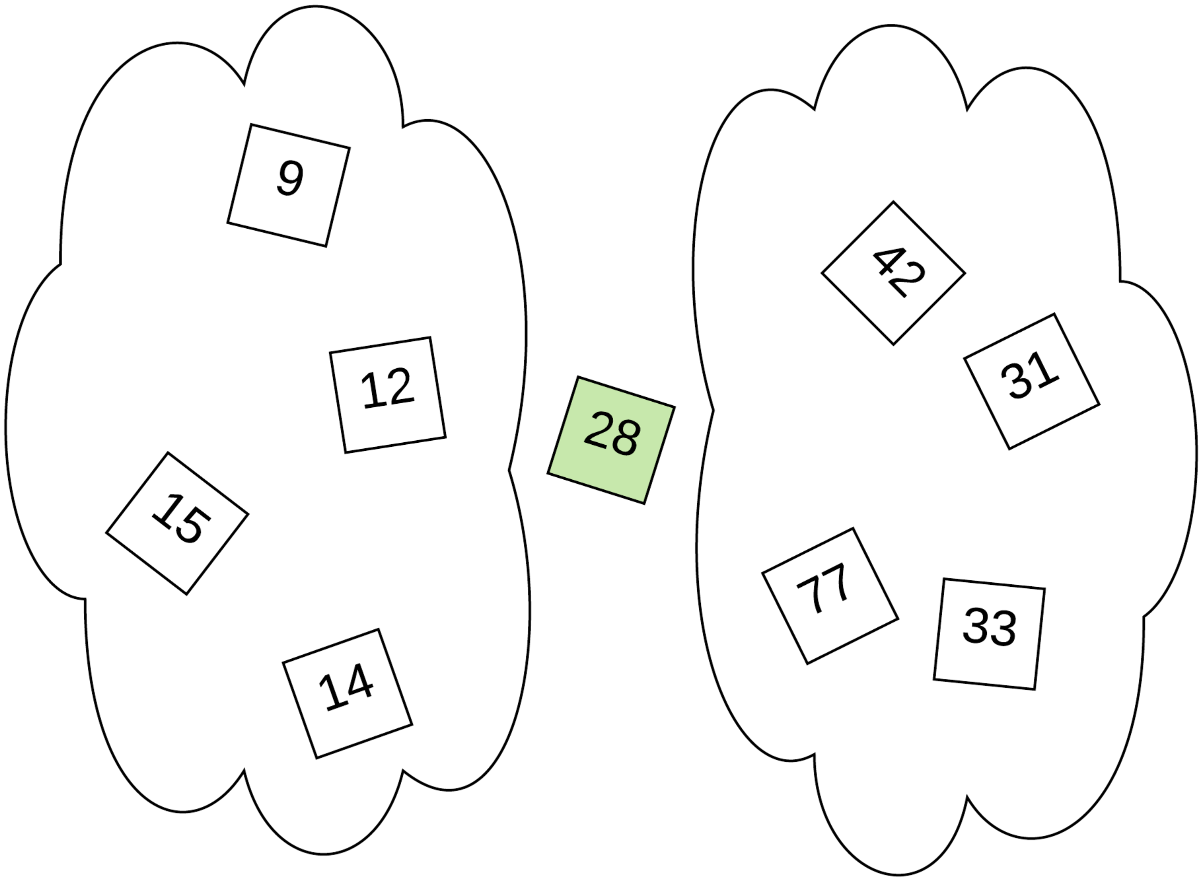
Hard to get all 3.

Quicksort



Pick a pivot

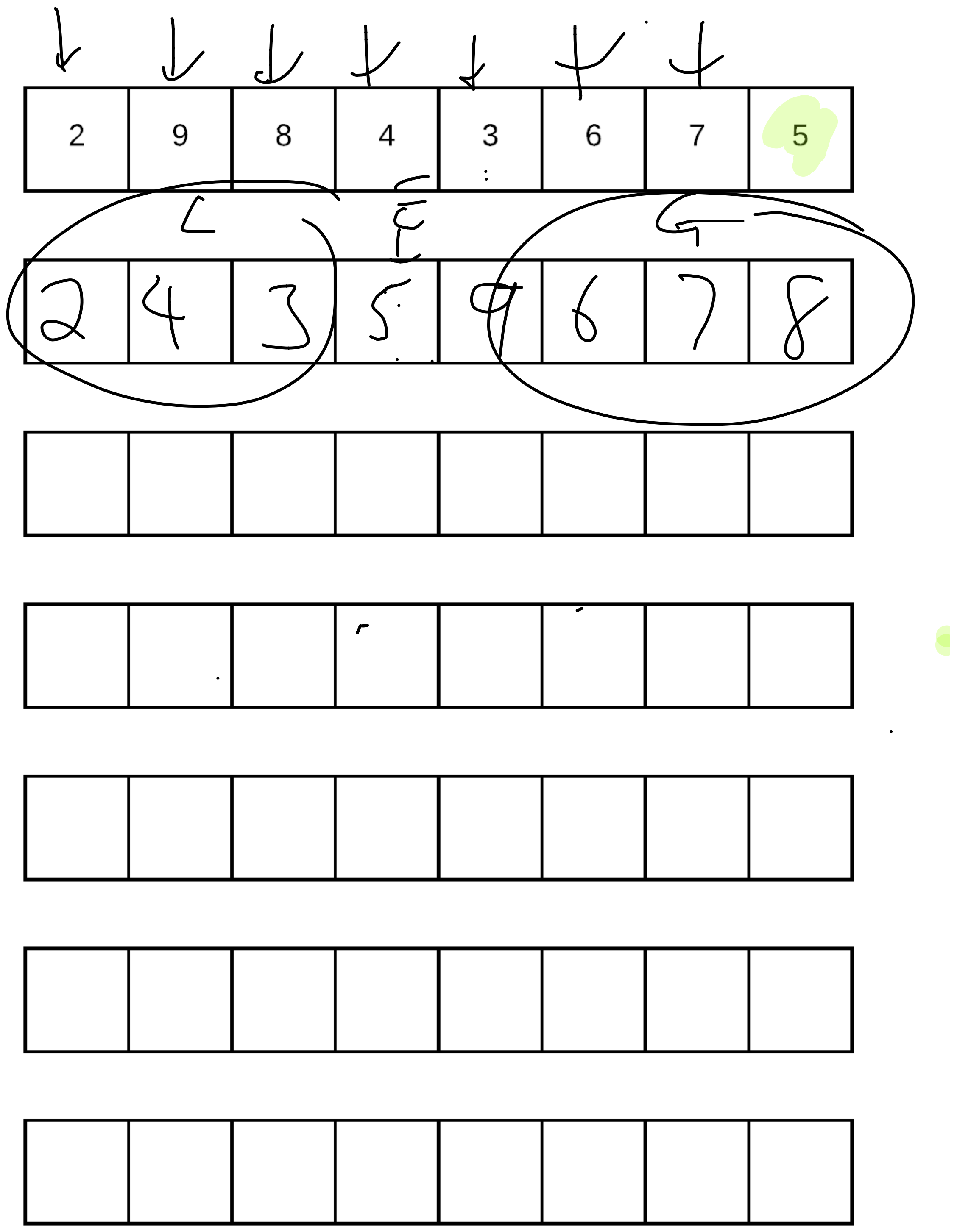



9
12
15
14
28
42
31
77
33

General Algorithm:

Given an array S
- select pivot
- partition array into 3 pieces
  - L - less than pivot
  - E - equal to than pivot
  - G - greater than pivot
- return [Quicksort(L) + E + Quicksort(G)]

For now, lets choose pivot = last number in S


2 9 8 4 3 6 7 5
L
E
G
2 4 3 5 9 6 7 8

Worst case analysis

Upperbound

- each element in S is picked at most once to be pivot
- each can at most be compared at most to every other element once
- thus each pair is compared at most once
-
- there are $\binom{n}{2}$ pairs $\frac{n!}{(n-2)!(2!)} = \frac{(n)(n-1)}{2!}$
-
- therefore T(n) is in O(n^2)

But quicksort is quicker than that... Average case next class

-