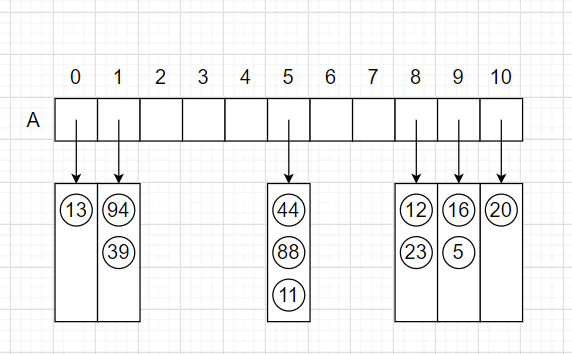
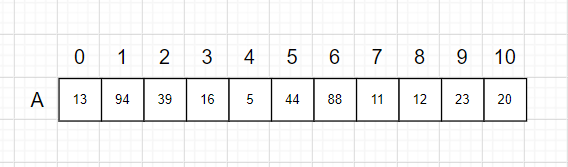
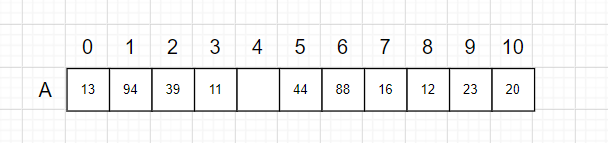
1. Draw the 11-entry hash table that results from using the hash function, h(i) = (3i+5) mod 11, to hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, assuming collisions are handled by chaining.



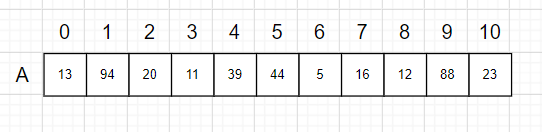
1. What is the result of the previous exercise, assuming collisions are handled by linear probing?



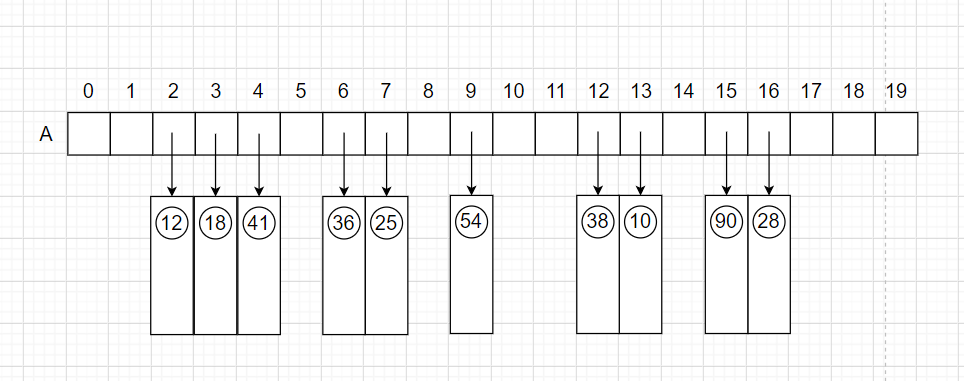
1. Show the result of Exercise R-10.9, assuming collisions are handled by quadratic probing, up to the point where the method fails.



1. What is the result of Exercise R-10.9 when collisions are handled by double hashing using the secondary hash function h(k) = 7 − (k mod 7)?



1. Show the result of rehashing the hash table shown below into a table of size 19 using the new hash function h(k) = 3k mod 17.



1. Suppose that each row of an n×n array A consists of 1’s and 0’s such that, in any row of A, all the 1’s come before any 0’s in that row. Assuming A is already in memory, describe a method running in O(n log n) time (not O(n2 ) time!) for counting the number of 1’s in A.

An augmented binary search can be conducted on each row.

Below, the binary\_search() looks to return the index at which the first 0 in the row is found.

* Within a basic for-loop that passes in each row of the nxn array, the binary search will return the index of the first zero, which implicitly gives the number of 1's in the row thanks to the nature of pythonic indexing.
* This works if the row is all 1's (effectively returns len(row)) or all 0's (returns 0).
* Binary Search runs in O(log*n*)
* B\_Search is called n times (thanks to the square n x n array)
* Hence, the algorithm runs in O(n log*n*)