**BUCKET SORTING ALGORITHM**

Bucket Sort is a sorting algorithm that divides the unsorted array elements into several groups called buckets.

● Scatter-Gather approach

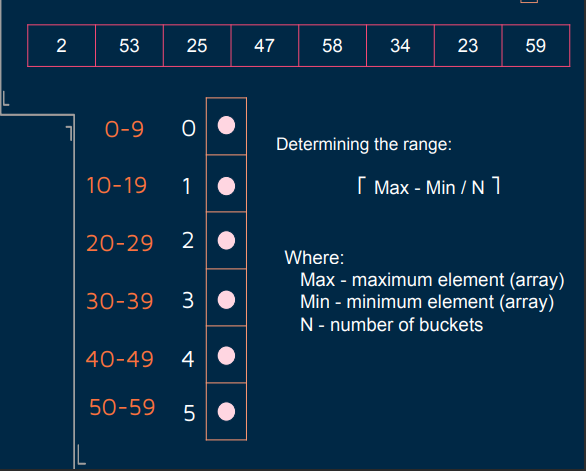
● Out-of-place algorithm

● Both stable and unstable sort

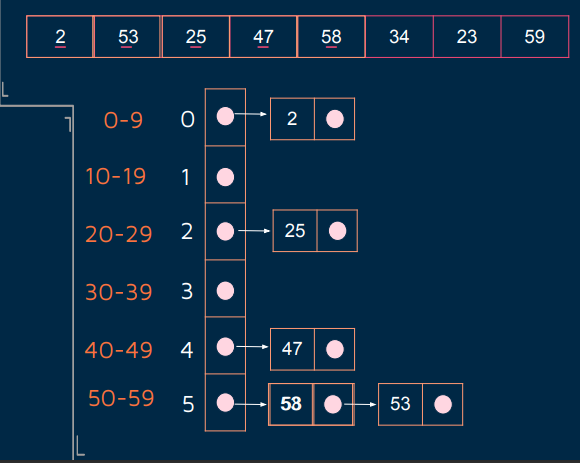
|  |  |
| --- | --- |
| **In-place** | **Out-of-place** |
| ●Transforms the input without using any extra memory.  ● Amount of memory required must not be dependent on the input size and should be constant. | ● the extra space used by an out-of-place algorithm depends on the input size. |

**ALGORITHM**

1. Create and empty array of size n
   1. Max – min / N

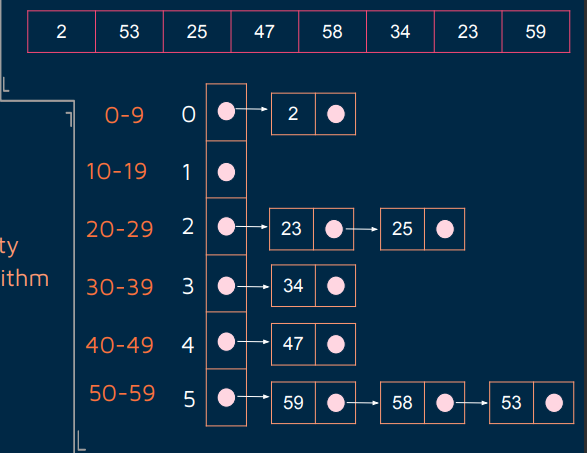


1. Loop through the original array and put each array element in a bucket

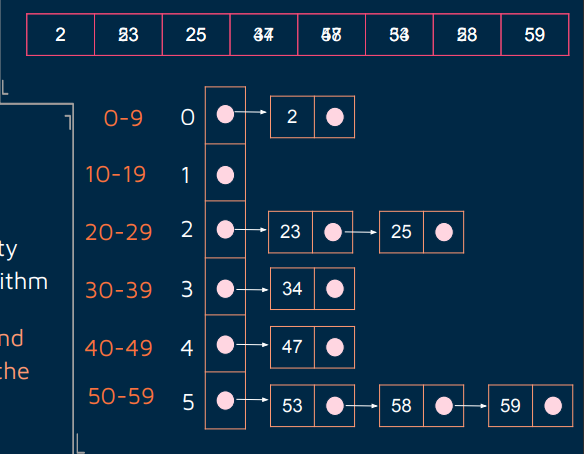


(Data structure similar to open hashing)

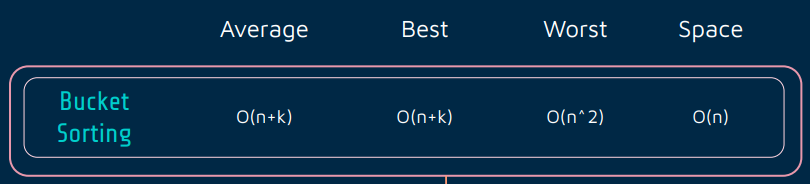
1. Sort each of the non-empty buckets using sorting algorithm



1. Visit buckets in order and put all elements back into the original array



**TIME AND SPACE COMPLEXITY**

****

**VARIATIONS**

**Postman’s Sort**

○ An algorithm that takes advantage of hierarchical structures of elements

○ Similar to radix sort

**Histogram Sort**

○ Checks the number of elements that will be in each bucket using a count array

○ Also known as counting sort

**Proxmap Sort**

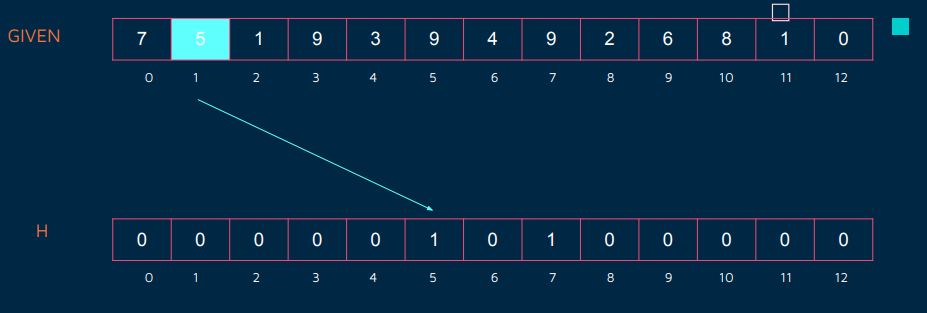
○ Uses a "map key" function that preserves a partial ordering on the keys

**Shuffle Sort**

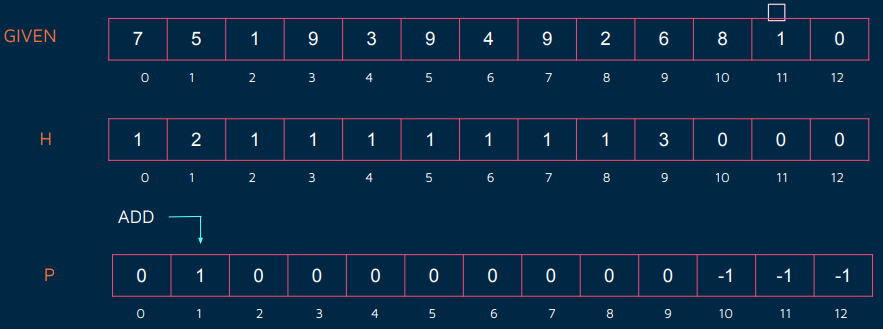
○ Begins by removing the first 1/8 of the n items to be sorted, sorts them recursively, and puts them in an array

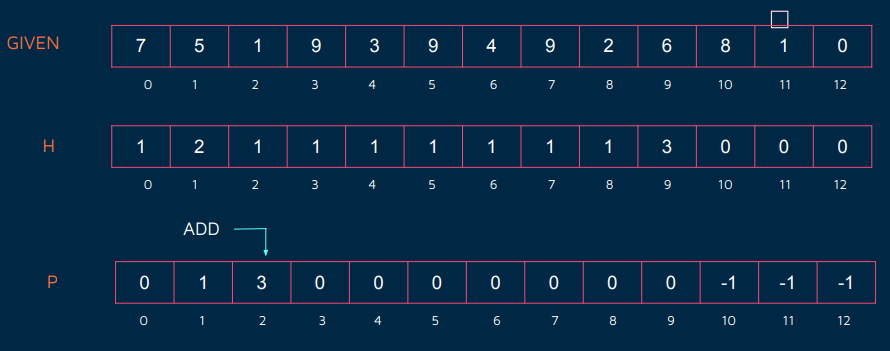
**SIMULATION OF PROXMAP SORT**

1. Determine how many keys will map to the same subarray, using an array of "hit counts" called H – similar to histogram in counting sort.

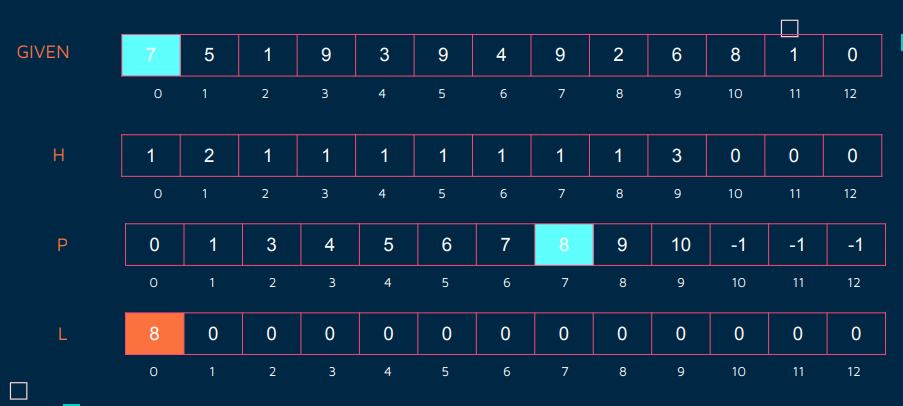


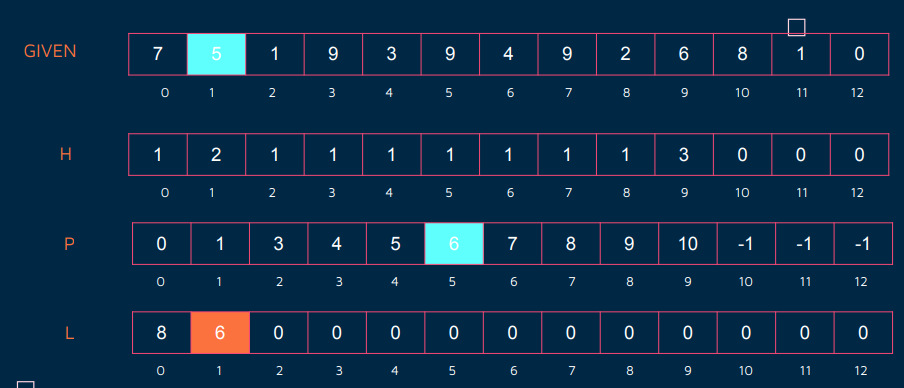
1. Determine where each subarray will begin in the destination array so that each bucket is exactly the right size to hold all the keys that will map to it, using an array of "proxmaps" called P. – elements in H that have no hit count are -1 in P.



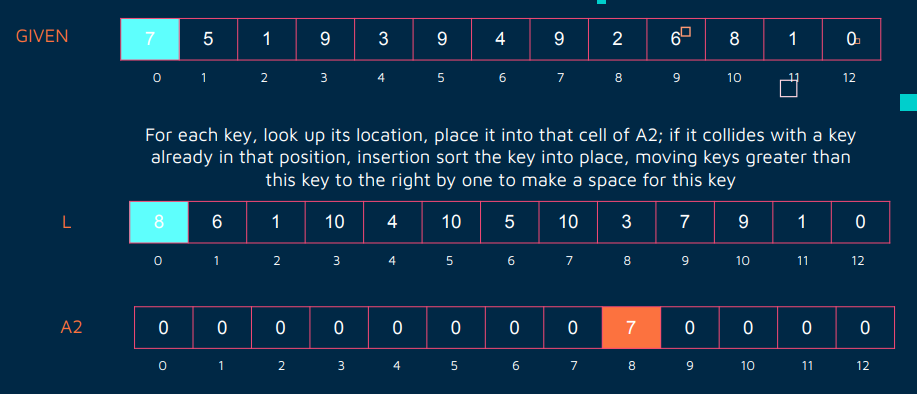


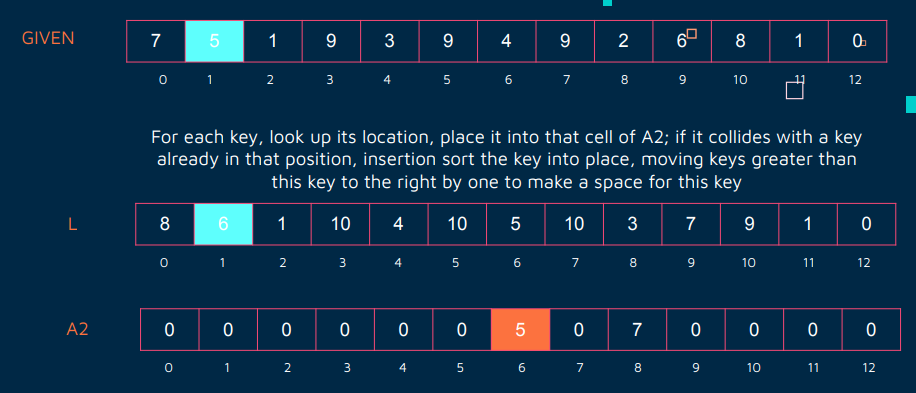
1. For each key, compute the subarray it will map to, using an array of "locations" called L

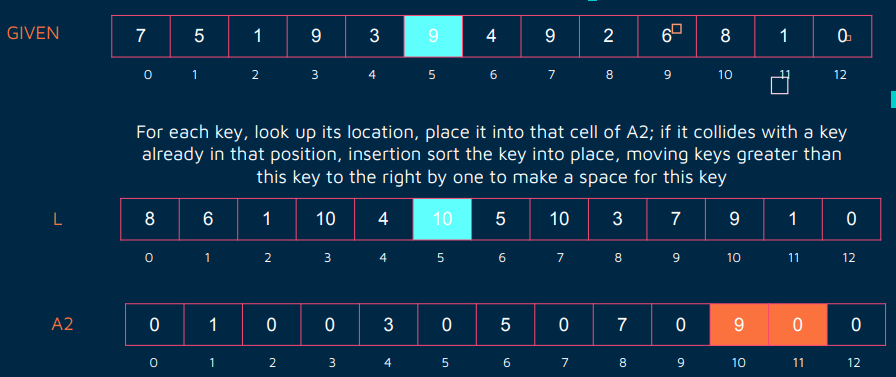




1. For each key, look up its location, place it into that cell of A2; if it collides with a key already in that position, insertion sort the key into place, moving keys greater than this key to the right by one to make a space for this key.

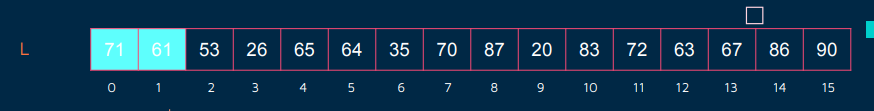
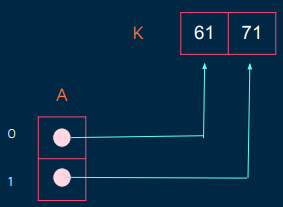




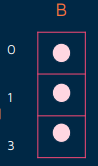


**SHUFFLE SORT SIMULATION**

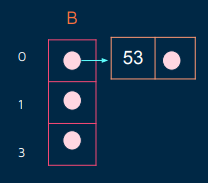
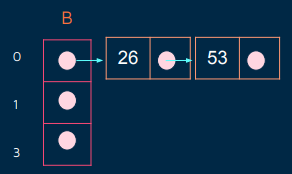
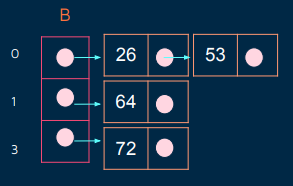
1. Determine **n** number of elements in array. Let K be a list of the first n/8 elements of L. Sort K recursively



1. Let A be an array of n/8 pointers to elements, and set them to the elements of K



1. Let B be an array of n/8+1 empty lists. These correspond to the lists which are in between, precede and proceed the elements in K
2. For the remaining elements in L, append each to the appropriate list in B as determined by a binary search in A. Recursively sort elements listed in B



1. Construct the original list by appending the elements of the first list in B, then the first element of K, then the second list in B, then the second element of K, ..., then the nth element of K, then the n+lth element of B, and return

