THE UNIVERSITY OF VERMONT
COLLEGE OF ENGINEERING &
MATHEMATICAL SCIENCES

Bucket Sort & Radix Sort

CS 124 / Department of Computer Science

Bucket Sort and Radix Sort

Bucket sort and radix sort work by distributing and collecting the elements to be sorted. This is a different approach from any we have seen so far.

All the algorithms we've seen so far use comparison to perform the sort. These algorithms can't do any better than $O(n \log n)$.

Bucket and radix sorting algorithms work best when we have more-or-less uniformly distributed data.

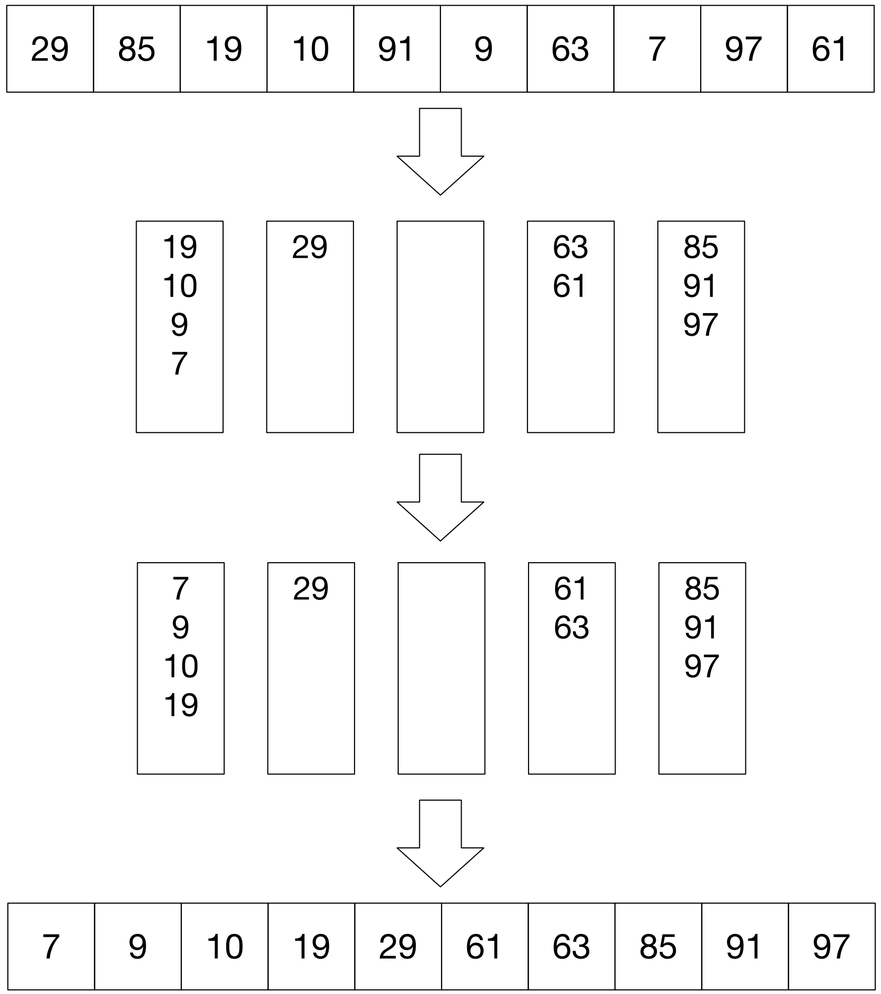
Bucket sort works best when we have a limited range of possible values.

Bucket Sort

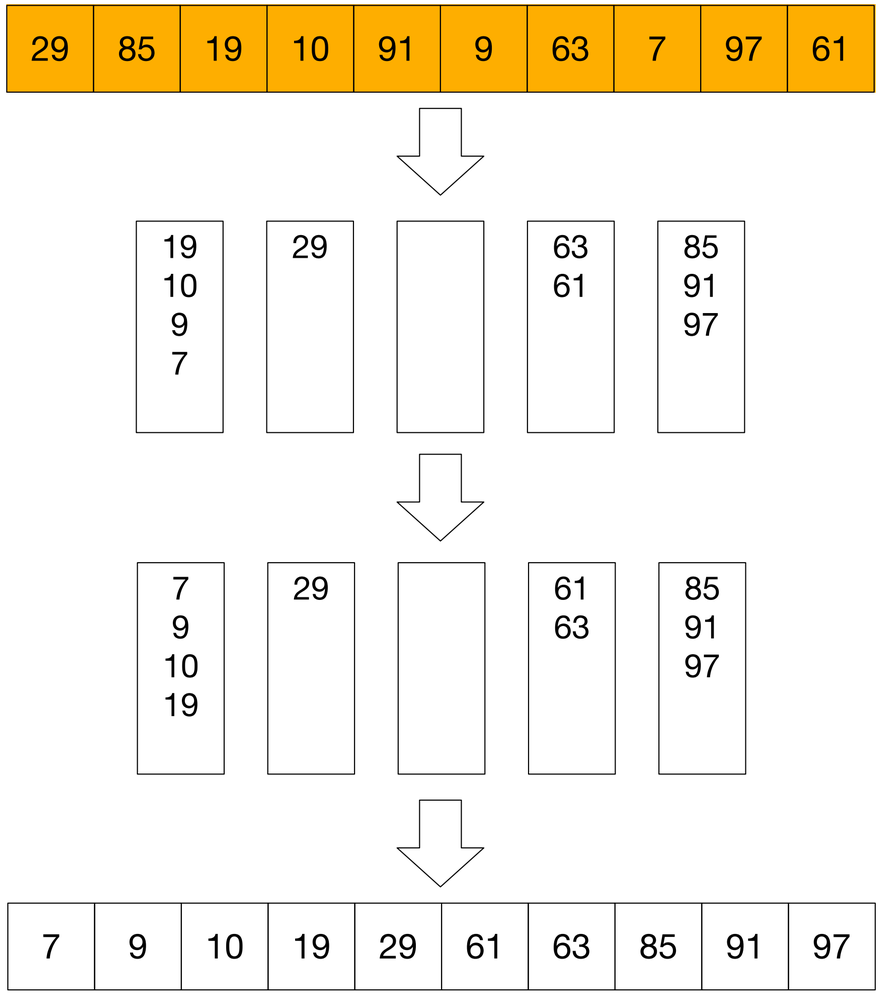
Bucketsort(vector)

- Make buckets
- Distribute the items into the buckets
- Sort the items within each bucket
- Gather the results back into the original vector

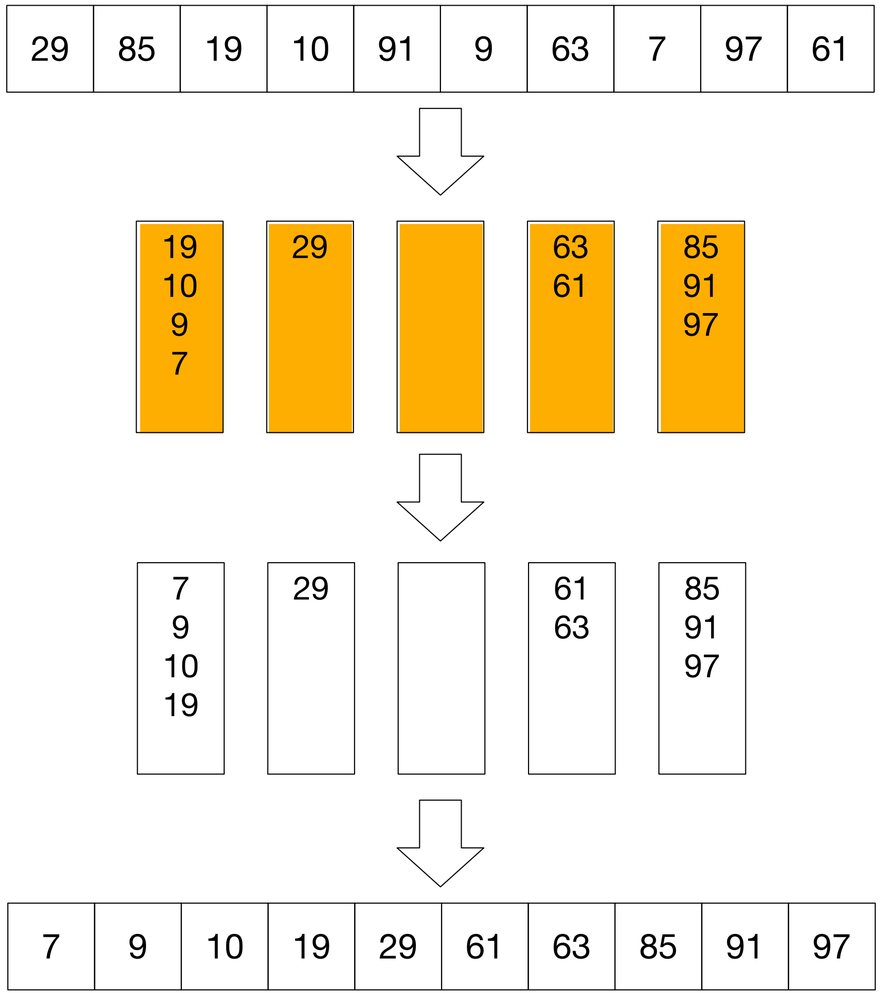
Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
7 9 10 19 29 61 63 85 91 97

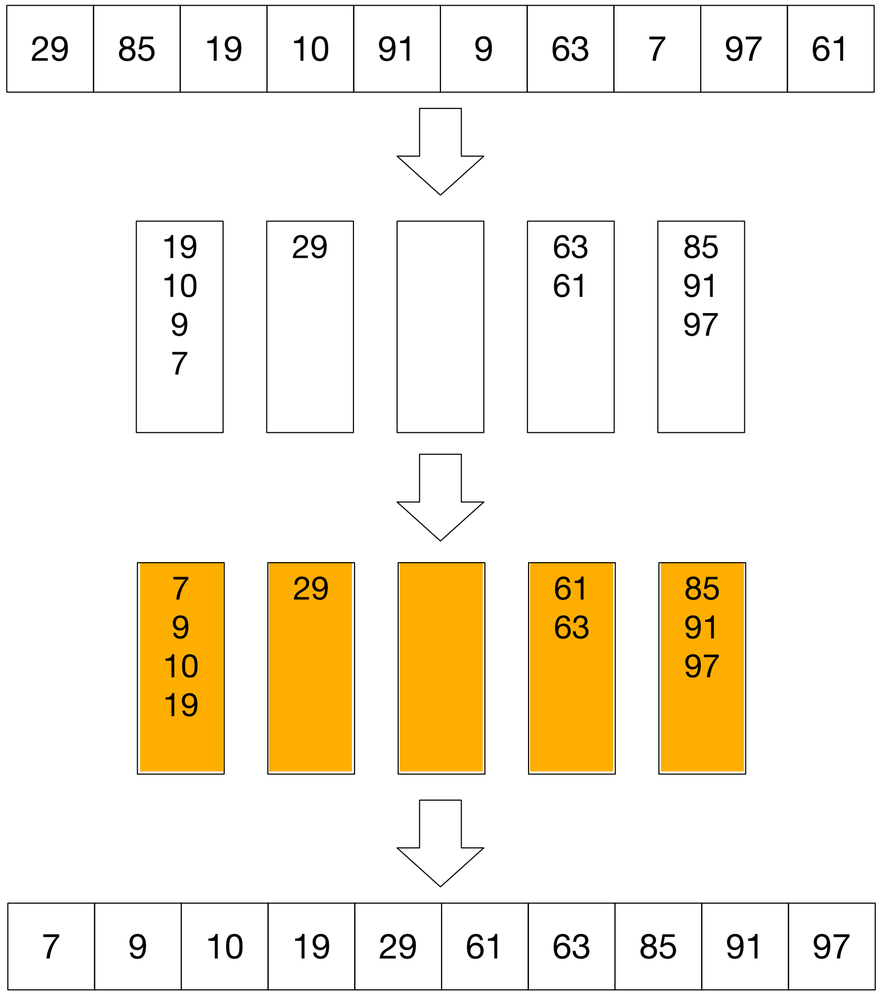
Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
7 9 10 19 29 61 63 85 91 97

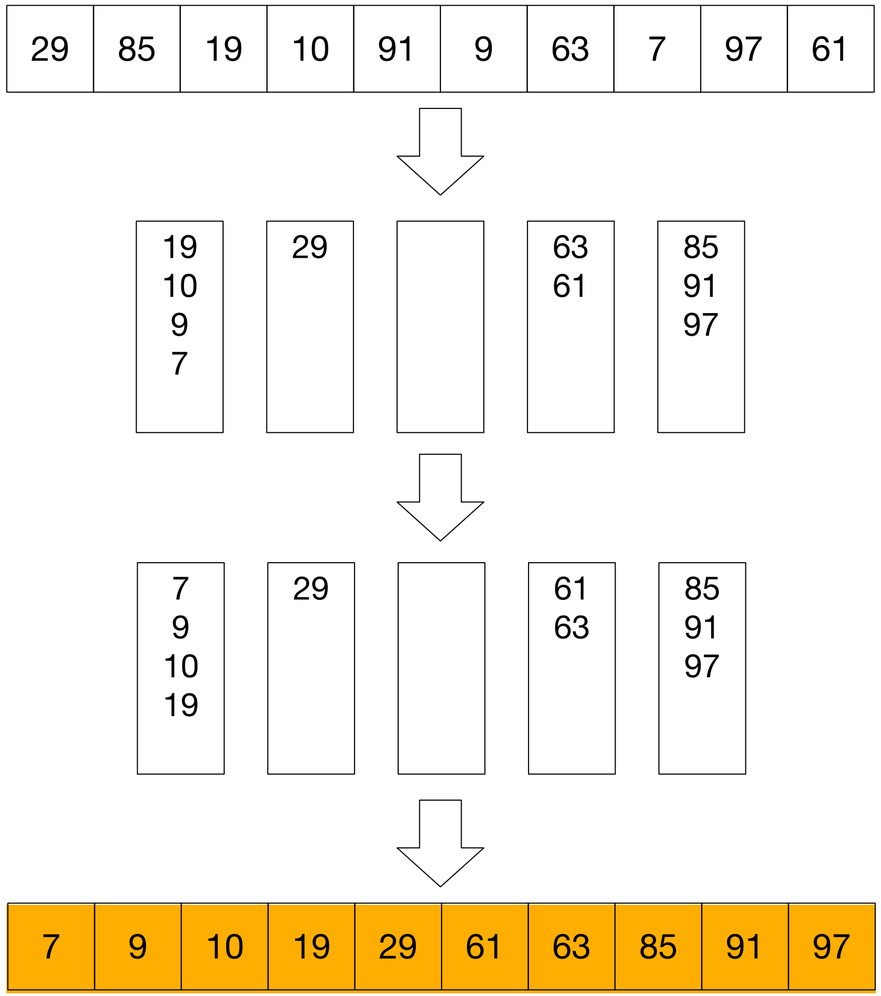
Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
7 9 10 19 29 61 63 85 91 97

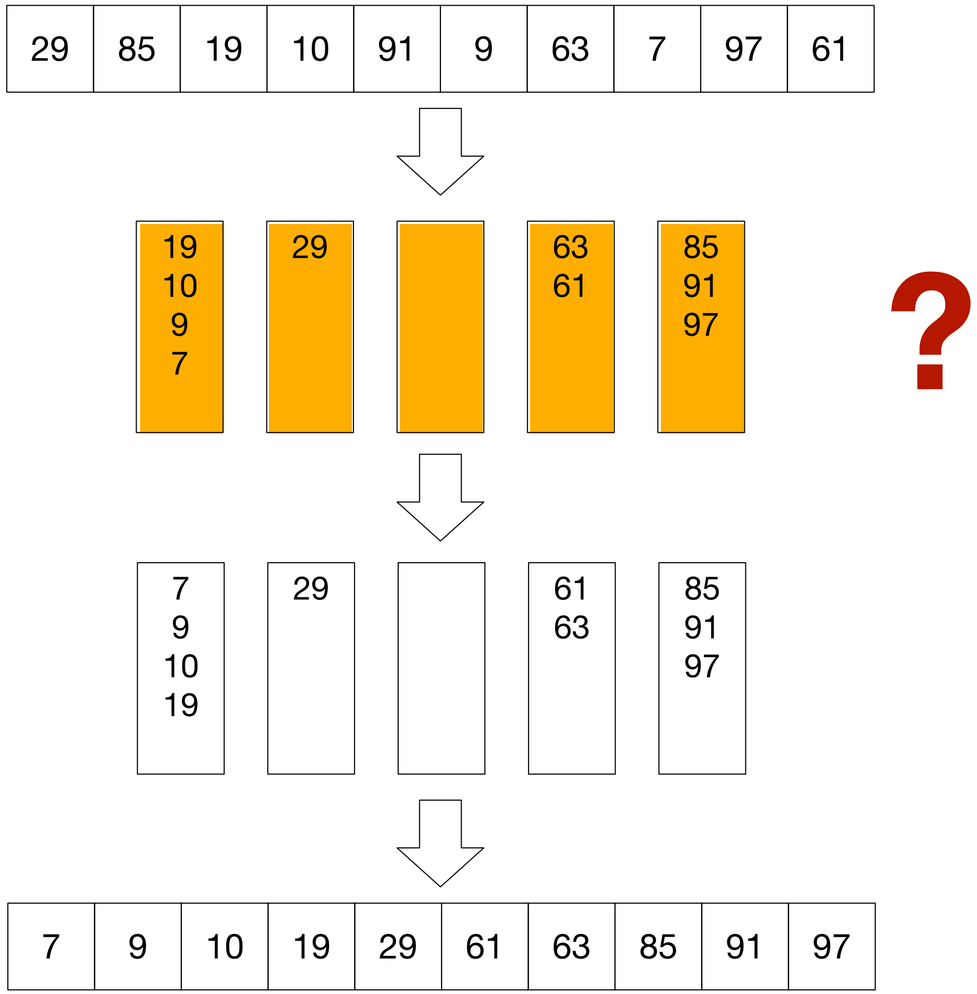
Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
7 9 10 19 29 61 63 85 91 97

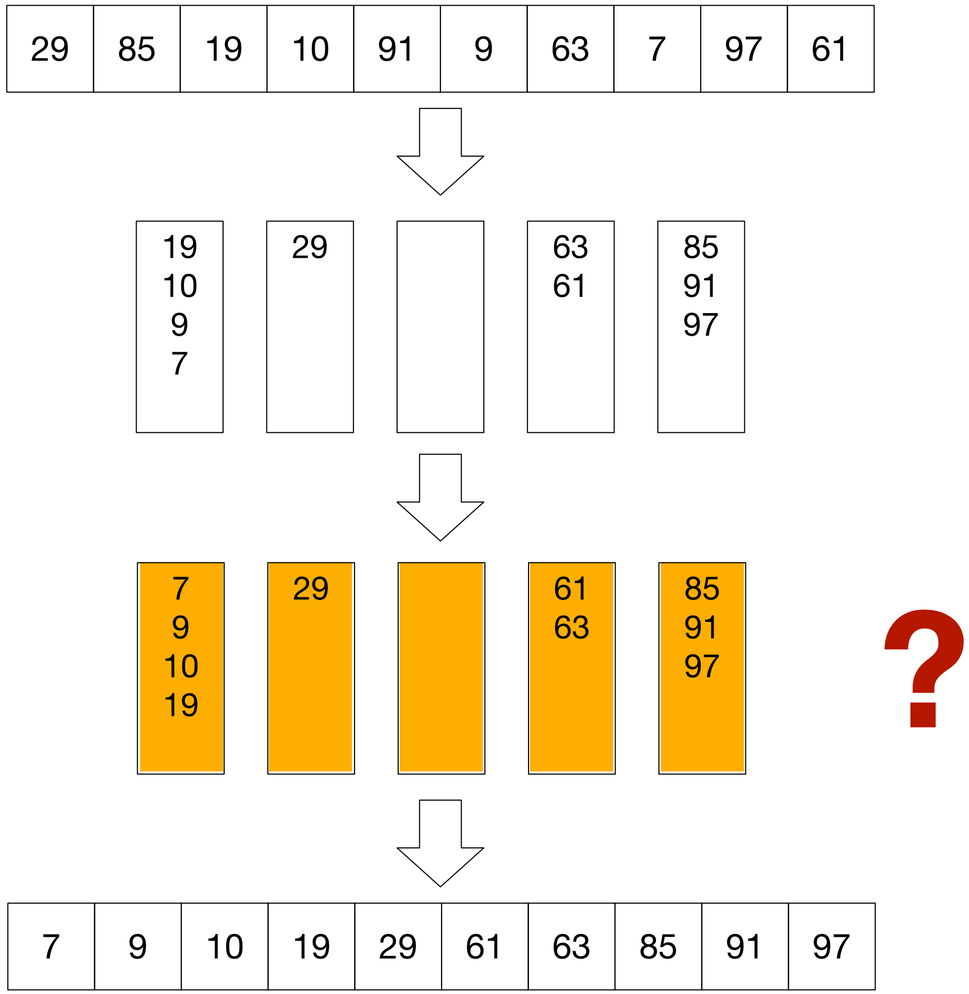
Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
7 9 10 19 29 61 63 85 91 97

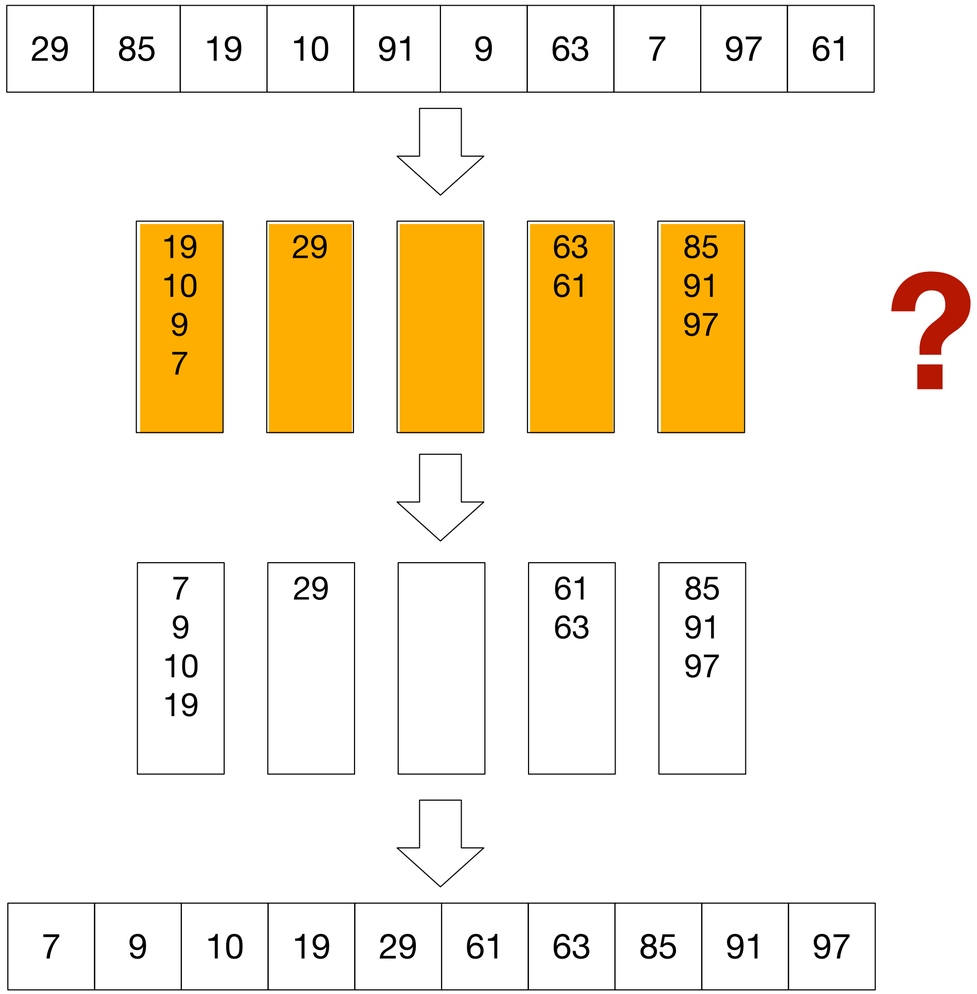
Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
7 9 10 19 29 61 63 85 91 97
?

Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
?
7 9 10 19 29 61 63 85 91 97

Bucket Sort


29 85 19 10 91 9 63 7 97 61
19 10 9 7
29
63 61
85 91 97
7 9 10 19
29
61 63
85 91 97
7 9 10 19 29 61 63 85 91 97
?

Bucket Sort

3000, 1000, 0, 2000

Bucket Sort

3000, 1000, 0, 2000

2001, 2010, 1999, 2002, 2000, 2008, 2009, 1998, 2004

Bucket Sort Complexity

Assuming data are reasonably uniformly distributed, and you have *n* elements and *m* buckets:

- Distributing takes *n* steps.
- Sorting each bucket takes *f*(*n* / *m*) steps where *f* is the run-time of the function used to sort the buckets. With *m* buckets this gives us *m* x *f*(*n* / *m*).
- Gathering takes n steps.

$$\mathcal{O}(n) + \mathcal{O}(m \times f(n/m)) + \mathcal{O}(n) = \mathcal{O}(n+m)$$

Comparison

Algorithm	Time complexity	Space complexity	Stable	Comment
Bubble sort	$O(n^2)$	$O(1)$	yes	can tell if list is already sorted
Selection sort	$O(n^2)$	$O(1)$	no	performs fewest swaps
Insertion sort	$O(n^2)$	$O(1)$	yes	ignores unsorted portion of vector / can process data on-line
Merge sort	$O(n \log n)$	$O(n)$	yes	recursive divide-and-conquer
Quicksort	$O(n \log n)$	$O(1)$	no	recursive divide-and-conquer
Bucket sort	$O(n + m)$	$O(n + m)$	depends	requires certain properties of the data to be effective

Radix Sort

Radix sort is a related algorithm that works with data that can be sorted lexicographically. Lexicographic sorting is just like sorting words in a dictionary.

Here we'll take a look at sorting numbers based on their digits -- working from the least significant digit to the most significant digit.

"Radix" is just another word for "base", as in "base two" for binary, or "base 10" in our everyday counting system.

Radix Sort

Radix sort is a related algorithm that works with data that can be sorted lexicographically. Lexicographic sorting is just like sorting words in a dictionary.

Here we'll take a look at sorting numbers based on their digits -- working from the least significant digit to the most significant digit.

"Radix" is just another word for "base", as in "base two" for binary, or "base 10" in our everyday counting system.

Radix sort's complexity is O (*n* x *d*). So radix sort can outperform O (*n* log *n*) algorithms when *d* < log *n*.

Radix Sort

Image source: National Museum of American History

Radix Sort

Radix sort (vector)
    Make buckets (10)
    For each digit:
        For each element:
            Distribute element into the appropriate bucket based on digit
        For each bucket:
            Gather the results back into the original vector

Radix Sort

192, 379, 012, 457, 004, 275, 014, 203

Radix Sort

192, 379, 012, 457, 004, 275, 014, 203

Radix Sort

192, 012, 203, 004, 014, 275, 457, 379

Radix Sort

192, 012, 203, 004, 014, 275, 457, 379

Radix Sort

203, 004, 012, 014, 457, 275, 379, 192

Radix Sort

203, 004, 012, 014, 457, 275, 379, 192

Radix Sort

004, 012, 014, 192, 203, 275, 379, 457

Radix Sort

004, 012, 014, 192, 203, 275, 379, 457

Radix

How do we get the value of a given digit without performing comparisons?

We take the number, divide by the appropriate power of our radix (or base) and then take that value modulo the radix. Example, base 10:

To get the third digit of 2708, we integer divide by 10^2 which gives us 27, and then 27 % 10 = 7.

insert video here

Comparison

Algorithm	Time complexity	Space complexity	Stable	Comment
Bubble sort	$O(n^2)$	$O(1)$	yes	can tell if list is already sorted
Selection sort	$O(n^2)$	$O(1)$	no	performs fewest swaps
Insertion sort	$O(n^2)$	$O(1)$	yes	ignores unsorted portion of vector / can process data on-line
Merge sort	$O(n \log n)$	$O(n)$	yes	recursive divide-and-conquer
Quicksort	$O(n \log n)$	$O(1)$	no	recursive divide-and-conquer
Bucket sort	$O(n + m)$	$O(n + m)$	depends	requires certain properties of the data to be effective
Radix sort	$O(nd)$	$O(n + d)$	yes	requires data that can be sorted lexicographically

Summary

- Bucket sort and radix sort perform sorting without making comparisons
- Distribute and collect
- They can outperform $O(n \log n)$ algorithms, but only under certain circumstances, i.e., data have to have certain properties -- both in terms of data type and distribution.