Bucket sort

From Wikipedia, the free encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

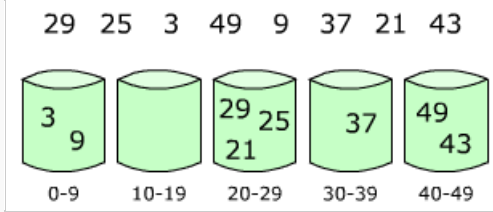
Languages
Български
Čeština
Deutsch
Español
فارسی
Français
한국어
Հայերեն
Italiano
עברית
日本語
Polski
Português
Русский
Српски / srpski
Türkçe
Українська
中文
Edit links

> This article **needs attention from an expert in Computer science**. Please add a *reason* or a *talk* parameter to this template to explain the issue with the article. WikiProject Computer science (or its Portal) may be able to help recruit an expert. *(November 2008)*

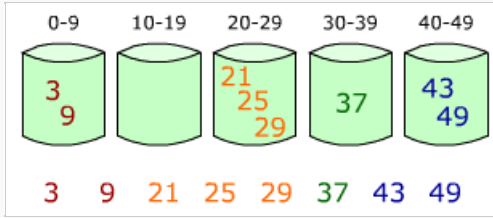
Bucket sort

Class	Sorting algorithm
Data structure	Array
Worst case performance	$O(n^2)$
Best case performance	$\Omega(n+k)$
Average case performance	$\Theta(n+k)$
Worst case space complexity	$O(n \cdot k)$

Bucket sort, or **bin sort**, is a sorting algorithm that works by distributing the elements of an array into a number of buckets. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. It is a distribution sort, and is a cousin of radix sort in the most to least significant digit flavour. Bucket sort is a generalization of pigeonhole sort. Bucket sort can be implemented with comparisons and therefore can also be considered a comparison sort algorithm. The computational complexity estimates involve the number of buckets.



Elements are distributed among bins



Then, elements are sorted within each bin

Bucket sort works as follows:

1. Set up an array of initially empty "buckets".
2. **Scatter**: Go over the original array, putting each object in its bucket.
3. Sort each non-empty bucket.
4. **Gather**: Visit the buckets in order and put all elements back into the original array.

Contents [hide]

Pseudocode [edit]

```
function bucketSort(array, n) is
  buckets ← new array of n empty lists
  for i = 0 to (length(array)-1) do
    insert array[i] into buckets[msbits(array[i], k)]
  for i = 0 to n - 1 do
    nextSort(buckets[i]);
  return the concatenation of buckets[0], ...., buckets[n-1]
```

Here *array* is the array to be sorted and *n* is the number of buckets to use. The function *msbits(x,k)* returns the

k most significant bits of *x* (*floor(x/2^(size(x)-k))*); different functions can be used to translate the range of elements in *array* to *n* buckets, such as translating the letters A–Z to 0–25 or returning the first character (0–255) for sorting strings. The function *nextSort* is a sorting function; using *bucketSort* itself as *nextSort* produces a relative of radix sort; in particular, the case *n* = *2* corresponds to quicksort (although potentially with poor pivot choices).

Optimizations [edit]

A common optimization is to put the unsorted elements of the buckets back in the original array *first*, then run insertion sort over the complete array; because insertion sort's runtime is based on how far each element is from its final position, the number of comparisons remains relatively small, and the memory hierarchy is better exploited by storing the list contiguously in memory.[1]

Variants [edit]

Generic bucket sort [edit]

The most common variant of bucket sort operates on a list of *n* numeric inputs between zero and some maximum value *M* and divides the value range into *n* buckets each of size *M*/*n*. If each bucket is sorted using insertion sort, the sort can be shown to run in expected linear time (where the average is taken over all possible inputs).[2] However, the performance of this sort degrades with clustering; if many values occur close together, they will all fall into a single bucket and be sorted slowly.

ProxmapSort [edit]

Main article: Proxmap sort

Similar to generic bucket sort as described above, **ProxmapSort** works by dividing an array of keys into subarrays via the use of a "map key" function that preserves a partial ordering on the keys; as each key is added to its subarray, insertion sort is used to keep that subarray sorted, resulting in the entire array being in sorted order when ProxmapSort completes. ProxmapSort differs from bucket sorts in its use of the map key to place the data approximately where it belongs in sorted order, producing a "proxmap" — a proximity mapping — of the keys.

Histogram sort [edit]

Another variant of bucket sort known as histogram sort or counting sort adds an initial pass that counts the number of elements that will fall into each bucket using a count array. Using this information, the array values can be arranged into a sequence of buckets in-place by a sequence of exchanges, leaving no space overhead for bucket storage.[3]

Postman's sort [edit]

The **Postman's sort** is a variant of bucket sort that takes advantage of a hierarchical structure of elements, typically described by a set of attributes. This is the algorithm used by letter-sorting machines in post offices: mail is sorted first between domestic and international; then by state, province or territory; then by destination post office; then by routes, etc. Since keys are not compared against each other, sorting time is O(*cn*), where *c* depends on the size of the key and number of buckets. This is similar to a radix sort that works "top down," or "most significant digit first."[4]

Shuffle sort [edit]

The **shuffle sort**[5] is a variant of bucket sort that begins by removing the first 1/8 of the *n* items to be sorted, sorts them recursively, and puts them in an array. This creates *n*/8 "buckets" to which the remaining 7/8 of the items are distributed. Each "bucket" is then sorted, and the "buckets" are concatenated into a sorted array. Shuffle sort is used as a step in a J sort.

Comparison with other sorting algorithms [edit]

Bucket sort can be seen as a generalization of counting sort; in fact, if each bucket has size 1 then bucket sort degenerates to counting sort. The variable bucket size of bucket sort allows it to use O(*n*) memory instead of O(*M*) memory, where *M* is the number of distinct values; in exchange, it gives up counting sort's O(*n* + *M*) worst-case behavior.

Bucket sort with two buckets is effectively a version of quicksort where the pivot value is always selected to be the middle value of the value range. While this choice is effective for uniformly distributed inputs, other means

of choosing the pivot in quicksort such as randomly selected pivots make it more resistant to clustering in the input distribution.

The *n*-way mergesort algorithm also begins by distributing the list into *n* sublists and sorting each one; however, the sublists created by mergesort have overlapping value ranges and so cannot be recombined by simple concatenation as in bucket sort. Instead, they must be interleaved by a merge algorithm. However, this added expense is counterbalanced by the simpler scatter phase and the ability to ensure that each sublist is the same size, providing a good worst-case time bound.

Top-down radix sort can be seen as a special case of bucket sort where both the range of values and the number of buckets is constrained to be a power of two. Consequently, each bucket's size is also a power of two, and the procedure can be applied recursively. This approach can accelerate the scatter phase, since we only need to examine a prefix of the bit representation of each element to determine its bucket.

References [edit]

1. ^ Corwin, E. and Logar, A. "Sorting in linear time — variations on the bucket sort". *Journal of Computing Sciences in Colleges*, 20, 1, pp.197–202. October 2004.
2. ^ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 8.4: Bucket sort, pp.174–177.
3. ^ NIST's Dictionary of Algorithms and Data Structures: histogram sort
4. ^ http://www.rrsd.com/psort/cuj/cuj.htm
5. ^ A revolutionary new sort from John Cohen Nov 26, 1997

- Paul E. Black "Postman's Sort" from Dictionary of Algorithms and Data Structures at NIST.
- Robert Ramey "'The Postman's Sort" *C Users Journal* Aug. 1992
- NIST's Dictionary of Algorithms and Data Structures: bucket sort

External links [edit]

- Bucket Sort Code for Ansi C
- Variant of Bucket Sort with Demo

v · t · e	**Sorting algorithms** [hide]
Theory	Computational complexity theory · Big O notation · Total order · Lists · Inplacement · Stability · Comparison sort · Adaptive sort · Sorting network · Integer sorting
Exchange sorts	Bubble sort · Cocktail sort · Odd–even sort · Comb sort · Gnome sort · Quicksort · Stooge sort · Bogosort
Selection sorts	Selection sort · Heapsort · Smoothsort · Cartesian tree sort · Tournament sort · Cycle sort
Insertion sorts	Insertion sort · Shellsort · Splaysort · Tree sort · Library sort · Patience sorting
Merge sorts	Merge sort · Cascade merge sort · Oscillating merge sort · Polyphase merge sort · Strand sort
Distribution sorts	American flag sort · Bead sort · **Bucket sort** · Burstsort · Counting sort · Pigeonhole sort · Proxmap sort · Radix sort · Flashsort
Concurrent sorts	Bitonic sorter · Batcher odd–even mergesort · Pairwise sorting network
Hybrid sorts	Block sort · Timsort · Introsort · Spreadsort · JSort
Other	Topological sorting · Pancake sorting · Spaghetti sort

Categories: Sorting algorithms | Stable sorts