= Integer sorting =

In computer science , integer sorting is the algorithmic problem of sorting a collection of data values by numeric keys , each of which is an integer . Algorithms designed for integer sorting may also often be applied to sorting problems in which the keys are floating point numbers or text strings . The ability to perform integer arithmetic on the keys allows integer sorting algorithms to be faster than comparison sorting algorithms in many cases , depending on the details of which operations are allowed in the model of computing and how large the integers to be sorted are .

Integer sorting algorithms including pigeonhole sort , counting sort , and radix sort are widely used and practical . Other integer sorting algorithms with smaller worst @-@ case time bounds are not believed to be practical for computer architectures with 64 or fewer bits per word . Many such algorithms are known , with performance depending on a combination of the number of items to be sorted , number of bits per key , and number of bits per word of the computer performing the sorting algorithm .

= = General considerations = =

= = = Models of computation = = =

Time bounds for integer sorting algorithms typically depend on three parameters : the number n of data values to be sorted , the magnitude K of the largest possible key to be sorted , and the number w of bits that can be represented in a single machine word of the computer on which the algorithm is to be performed . Typically , it is assumed that w ? log2 (max (n , K)) ; that is , that machine words are large enough to represent an index into the sequence of input data , and also large enough to represent a single key .

Integer sorting algorithms are usually designed to work in either the pointer machine or random access machine models of computing . The main difference between these two models is in how memory may be addressed . The random access machine allows any value that is stored in a register to be used as the address of memory read and write operations , with unit cost per operation . This ability allows certain complex operations on data to be implemented quickly using table lookups . In contrast , in the pointer machine model , read and write operations use addresses stored in pointers , and it is not allowed to perform arithmetic operations on these pointers . In both models , data values may be added , and bitwise Boolean operations and binary shift operations may typically also be performed on them , in unit time per operation . Different integer sorting algorithms make different assumptions , however , about whether integer multiplication is also allowed as a unit @-@ time operation . Other more specialized models of computation such as the parallel random access machine have also been considered .

Andersson , Miltersen & Thorup (1999) showed that in some cases the multiplications or table lookups required by some integer sorting algorithms could be replaced by customized operations that would be more easily implemented in hardware but that are not typically available on general @-@ purpose computers . Thorup (2003) improved on this by showing how to replace these special operations by the bit field manipulation instructions already available on Pentium processors .

= = = Sorting versus integer priority queues = = =

A priority queue is a data structure for maintaining a collection of items with numerical priorities , having operations for finding and removing the item with the minimum priority value . Comparison @-@ based priority queues such as the binary heap take logarithmic time per update , but other structures such as the van Emde Boas tree or bucket queue may be faster for inputs whose priorities are small integers . These data structures can be used in the selection sort algorithm , which sorts a collection of elements by repeatedly finding and removing the smallest element from

the collection , and returning the elements in the order they were found . A priority queue can be used to maintain the collection of elements in this algorithm , and the time for this algorithm on a collection of n elements can be bounded by the time to initialize the priority queue and then to perform n find and remove operations . For instance , using a binary heap as a priority queue in selection sort leads to the heap sort algorithm , a comparison sorting algorithm that takes O (n log n) time . Instead , using selection sort with a bucket queue gives a form of pigeonhole sort , and using van Emde Boas trees or other integer priority queues leads to other fast integer sorting algorithms .

Instead of using an integer priority queue in a sorting algorithm , it is possible to go the other direction , and use integer sorting algorithms as subroutines within an integer priority queue data structure . Thorup (2007) used this idea to show that , if it is possible to perform integer sorting in time T (n) per key , then the same time bound applies to the time per insertion or deletion operation in a priority queue data structure . Thorup 's reduction is complicated and assumes the availability of either fast multiplication operations or table lookups , but he also provides an alternative priority queue using only addition and Boolean operations with time T (n) + T (log n) + T (log log n) + ... per operation , at most multiplying the time by an iterated logarithm .

= = = Usability = = =

The classical integer sorting algorithms of pigeonhole sort , counting sort , and radix sort are widely used and practical . Much of the subsequent research on integer sorting algorithms has focused less on practicality and more on theoretical improvements in their worst case analysis , and the algorithms that come from this line of research are not believed to be practical for current 64 @-@ bit computer architectures , although experiments have shown that some of these methods may be an improvement on radix sorting for data with 128 or more bits per key . Additionally , for large data sets , the near @-@ random memory access patterns of many integer sorting algorithms can handicap them compared to comparison sorting algorithms that have been designed with the memory hierarchy in mind .

Integer sorting provides one of the six benchmarks in the DARPA High Productivity Computing Systems Discrete Mathematics benchmark suite , and one of eleven benchmarks in the NAS Parallel Benchmarks suite .

= = Practical algorithms = =

Pigeonhole sort or counting sort can both sort n data items having keys in the range from 0 to K ? 1 in time O (n + K) . In pigeonhole sort (often called bucket sort) , pointers to the data items are distributed to a table of buckets , represented as collection data types such as linked lists , using the keys as indices into the table . Then , all of the buckets are concatenated together to form the output list . Counting sort uses a table of counters in place of a table of buckets , to determine the number of items with each key . Then , a prefix sum computation is used to determine the range of positions in the sorted output at which the values with each key should be placed . Finally , in a second pass over the input , each item is moved to its key 's position in the output array . Both algorithms involve only simple loops over the input data (taking time O (n)) and over the set of possible keys (taking time O (K)) , giving their O (n + K) overall time bound .

Radix sort is a sorting algorithm that works for larger keys than pigeonhole sort or counting sort by performing multiple passes over the data . Each pass sorts the input using only part of the keys , by using a different sorting algorithm (such as pigeonhole sort or counting sort) that is suited only for small keys . To break the keys into parts , the radix sort algorithm computes the positional notation for each key , according to some chosen radix ; then , the part of the key used for the ith pass of the algorithm is the ith digit in the positional notation for the full key , starting from the least significant digit and progressing to the most significant . For this algorithm to work correctly , the sorting algorithm used in each pass over the data must be stable : items with equal digits should not change positions with each other . For greatest efficiency , the radix should be chosen to be near the number of data items , n . Additionally , using a power of two near n as the radix allows the keys

for each pass to be computed quickly using only fast binary shift and mask operations . With these choices , and with pigeonhole sort or counting sort as the base algorithm , the radix sorting algorithm can sort n data items having keys in the range from 0 to K ? 1 in time O (n logn K) .

= = Theoretical algorithms = =

Many integer sorting algorithms have been developed whose theoretical analysis shows them to behave better than comparison sorting , pigeonhole sorting , or radix sorting for large enough combinations of the parameters defining the number of items to be sorted , range of keys , and machine word size . Which algorithm has the best performance depends on the values of these parameters . However , despite their theoretical advantages , these algorithms are not an improvement for the typical ranges of these parameters that arise in practical sorting problems .

= = = Algorithms for small keys = = =

A Van Emde Boas tree may be used as a priority queue to sort a set of n keys , each in the range from 0 to K ? 1 , in time O (n log log K) . This is a theoretical improvement over radix sorting when K is sufficiently large . However , in order to use a Van Emde Boas tree , one either needs a directly @-@ addressable memory of K words , or one needs to simulate it using a hash table , reducing the space to linear but making the algorithm be randomized . Another priority queue with similar performance (including the need for randomization in the form of hash tables) is the Y @-@ fast trie of Willard (1983) .

A more sophisticated technique with a similar flavor and with better theoretical performance was developed by Kirkpatrick & Reisch (1984) . They observed that each pass of radix sort can be interpreted as a range reduction technique that , in linear time , reduces the maximum key size by a factor of n ; instead , their technique reduces the key size to the square root of its previous value (halving the number of bits needed to represent a key) , again in linear time . As in radix sort , they interpret the keys as two @-@ digit base @-@ b numbers for a base b that is approximately ? K. They then group the items to be sorted into buckets according to their high digits , in linear time , using either a large but uninitialized direct addressed memory or a hash table . Each bucket has a representative , the item in the bucket with the largest key ; they then sort the list of items using as keys the high digits for the representatives and the low digits for the non @-@ representatives . By grouping the items from this list into buckets again , each bucket may be placed into sorted order , and by extracting the representatives from the sorted list the buckets may be concatenated together into sorted order . Thus , in linear time , the sorting problem is reduced to another recursive sorting problem in which the keys are much smaller , the square root of their previous magnitude . Repeating this range reduction until the keys are small enough to bucket sort leads to an algorithm with running time O (n log logn K) .

A complicated randomized algorithm of Han & Thorup (2002) allows these time bounds to be reduced even farther , to O (n ? log log K) .

= = = Algorithms for large words = = =

An integer sorting algorithm is said to be non @-@ conservative if it requires a word size w that is significantly larger than log max (n , K) . As an extreme instance , if w ? K , and all keys are distinct , then the set of keys may be sorted in linear time by representing it as a bitvector , with a 1 bit in position i when i is one of the input keys , and then repeatedly removing the least significant bit .

The non @-@ conservative packed sorting algorithm of Albers & Hagerup (1997) uses a subroutine , based on Ken Batcher 's bitonic sorting network , for merging two sorted sequences of keys that are each short enough to be packed into a single machine word . The input to the packed sorting algorithm , a sequence of items stored one per word , is transformed into a packed form , a sequence of words each holding multiple items in sorted order , by using this subroutine repeatedly to double the number of items packed into each word . Once the sequence is in packed form ,

Albers and Hagerup use a form of merge sort to sort it ; when two sequences are being merged to form a single longer sequence , the same bitonic sorting subroutine can be used to repeatedly extract packed words consisting of the smallest remaining elements of the two sequences . This algorithm gains enough of a speedup from its packed representation to sort its input in linear time whenever it is possible for a single word to contain ? (log n log log n) keys ; that is , when log K log n log log n ? cw for some constant c > 0 .

= = = Algorithms for few items = = =

Pigeonhole sort , counting sort , radix sort , and Van Emde Boas tree sorting all work best when the key size is small ; for large enough keys , they become slower than comparison sorting algorithms . However , when the key size or the word size is very large relative to the number of items (or equivalently when the number of items is small) , it may again become possible to sort quickly , using different algorithms that take advantage of the parallelism inherent in the ability to perform arithmetic operations on large words .

An early result in this direction was provided by Ajtai , Fredman & Komlós (1984) using the cell probe model of computation (an artificial model in which the complexity of an algorithm is measured only by the number of memory accesses it performs) . Building on their work , Fredman & Willard (1994) described two data structures , the Q @-@ heap and the atomic heap , that are implementable on a random access machine . The Q @-@ heap is a bit @-@ parallel version of a binary trie , and allows both priority queue operations and successor and predecessor queries to be performed in constant time for sets of O ((log N) 1 / 4) items , where N ? 2w is the size of the precomputed tables needed to implement the data structure . The atomic heap is a B @-@ tree in which each tree node is represented as a Q @-@ heap ; it allows constant time priority queue operations (and therefore sorting) for sets of (log N) O (1) items .

Andersson et al . (1998) provide a randomized algorithm called signature sort that allows for linear time sorting of sets of up to 2O ((log w) 1 / 2 ? ?) items at a time , for any constant ? > 0 . As in the algorithm of Kirkpatrick and Reisch , they perform range reduction using a representation of the keys as numbers in base b for a careful choice of b . Their range reduction algorithm replaces each digit by a signature , which is a hashed value with O (log n) bits such that different digit values have different signatures . If n is sufficiently small , the numbers formed by this replacement process will be significantly smaller than the original keys , allowing the non @-@ conservative packed sorting algorithm of Albers & Hagerup (1997) to sort the replaced numbers in linear time . From the sorted list of replaced numbers , it is possible to form a compressed trie of the keys in linear time , and the children of each node in the trie may be sorted recursively using only keys of size b , after which a tree traversal produces the sorted order of the items .

= = = Trans @-@ dichotomous algorithms = = =

Fredman & Willard (1993) introduced the transdichotomous model of analysis for integer sorting algorithms , in which nothing is assumed about the range of the integer keys and one must bound the algorithm 's performance by a function of the number of data values alone . Alternatively , in this model , the running time for an algorithm on a set of n items is assumed to be the worst case running time for any possible combination of values of K and w . The first algorithm of this type was Fredman and Willard 's fusion tree sorting algorithm , which runs in time O (n log n / log log n) ; this is an improvement over comparison sorting for any choice of K and w . An alternative version of their algorithm that includes the use of random numbers and integer division operations improves this to O (n ? log n) .

Since their work , even better algorithms have been developed . For instance , by repeatedly applying the Kirkpatrick ? Reisch range reduction technique until the keys are small enough to apply the Albers ? Hagerup packed sorting algorithm , it is possible to sort in time O (n log log n) ; however , the range reduction part of this algorithm requires either a large memory (proportional to ? K) or randomization in the form of hash tables .

Han & Thorup (2002) showed how to sort in randomized time O (n ? log log n) . Their technique involves using ideas related to signature sorting to partition the data into many small sublists , of a size small enough that signature sorting can sort each of them efficiently . It is also possible to use similar ideas to sort integers deterministically in time O (n log log n) and linear space . Using only simple arithmetic operations (no multiplications or table lookups) it is possible to sort in randomized expected time O (n log log n) or deterministically in time O (n (log log n) 1 + ?) for any constant ? > 0 .