Bucket-Sort and Radix-Sort

Bucket-Sort (§10.5.1)

Let be a sequence of \(n\) (key, element) items with keys in the range \([0, N-1]\).

Bucket-sort uses the keys as indices into an auxiliary array \(B\) of sequences (buckets).

Phase 1: Empty sequence \(S\) by moving each item \((k, a)\) into its bucket \(B(k)\).

Phase 2: For \(i = 0, ..., N-1\), move the items of bucket \(B(i)\) to the end of sequence \(S\).

Analysis:
- Phase 1 takes \(O(n)\) time.
- Phase 2 takes \(O(n + N)\) time.
- Bucket-sort takes \(O(n + N)\) time.

Lexicographic Order

A \(d\)-tuple is a sequence of \(d\) keys \((k_1, k_2, ..., k_d)\), where key \(k_i\) is said to be the \(i\)-th dimension of the tuple.

Example:

The Cartesian coordinates of a point in space are a 3-tuple.

The lexicographic order of two \(d\)-tuples is recursively defined as follows:

\[(k_1, k_2, ..., k_d) < (y_1, y_2, ..., y_d)\]

\[\iff \exists i \leq d \text{ s.t. } k_i < y_i \land (k_j = y_j) \text{ for } j > i\]

I.e., the tuples are compared by the first dimension, then by the second dimension, etc.

Lexicographic-Sort

Let \(C\) be the comparator that compares two tuples by their \(i\)-th dimension.

Let stableSort\(S, C\) be a stable sorting algorithm that uses comparator \(C\).

Lexicographic-sort sorts a sequence of \(d\)-tuples in lexicographic order by executing \(d\) times algorithm stableSort, one per dimension.

Lexicographic-sort runs in \(O(dT(n))\) time, where \(T(n)\) is the running time of stableSort.

Example:

\[(7,4,6) (5,1,5) (2,4,6) (2,1,4) (3,2,4) (3,2,4) (7,4,6) (2,4,6) (2,1,4) (3,2,4) (3,2,4) (5,1,5) (7,4,6) (7,4,6)\]
Radix-Sort (§10.5.2)

- Radix-sort is a specialization of lexicographic-sort that uses bucket-sort as the stable sorting algorithm in each dimension.
- Radix-sort is applicable to tuples where the keys in each dimension are integers in the range \([0, N - 1]\).
- Radix-sort runs in time \(O(d(n + N))\).

Algorithm \(\text{radixSort}(S, N)\)

- Input: sequence \(S\) of \(d\)-tuples such that \((0, \ldots, 0) \leq (x_1, \ldots, x_d)\) and \((x_1, \ldots, x_d) \leq (N - 1, \ldots, N - 1)\) for each tuple \((x_1, \ldots, x_d)\) in \(S\).
- Output: sequence \(S\) sorted in lexicographic order.

for \(i \leftarrow d\) downto 1
    \(\text{bucketSort}(S, 2)\)

Radix-Sort for Binary Numbers

- Consider a sequence of \(n\) \(b\)-bit integers \(X = x_{b-1} \cdots x_0\).
- We represent each element as a \(b\)-tuple of integers in the range \([0, 1]\) and apply radix-sort with \(N = 2\).
- This application of the radix-sort algorithm runs in \(O(bn)\) time.
- For example, we can sort a sequence of 32-bit integers in linear time.

Algorithm \(\text{binaryRadixSort}(S)\)

- Input: sequence \(S\) of \(b\)-bit integers.
- Output: sequence \(S\) sorted.

replace each element \(x\) of \(S\) with the item \((0, x)\)
for \(i \leftarrow 0\) to \(b - 1\)
    replace the key \(k\) of each item \((k, x)\) of \(S\) with bit \(x_i\) of \(x\)
\(\text{bucketSort}(S, 2)\)

Example

- Sorting a sequence of 4-bit integers

\[
\begin{array}{cccccccc}
1001 & 0010 & 1001 & 0010 & 1001 & 0010 & 1001 & 0010 \\
0010 & 1110 & 1001 & 0010 & 1110 & 1001 & 0010 & 1110 \\
1101 & 0001 & 1001 & 0001 & 1001 & 0001 & 1001 & 0001 \\
0001 & 1110 & 1101 & 1110 & 1101 & 1110 & 1101 & 1110 \\
\end{array}
\]