Bucket-Sort

- Let be \( S \) be a sequence of \( n \) 
  (key, element) entries 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 entry \((k, o)\) into 
  its bucket \( B[k] \)

  **Phase 2**: For \( i = 0, \ldots, N - 1 \), move 
  the entries 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

**Algorithm** \( \text{bucketSort}(S, N) \)

**Input** sequence \( S \) of (key, element) 
items with keys in the range \([0, N - 1]\)

**Output** sequence \( S \) sorted by 
increasing keys

\[
\text{while } \neg S\text{.isEmpty}()
\begin{align*}
  f & \leftarrow S\text{.first()} \\
  (k, o) & \leftarrow S\text{.remove}(f) \\
  B[k]\text{.addLast}((k, o))
\end{align*}
\]

\[
\text{for } i \leftarrow 0 \text{ to } N - 1
\begin{align*}
  \text{while } \neg B[i]\text{.isEmpty}()
  \begin{align*}
    f & \leftarrow B[i]\text{.first()} \\
    (k, o) & \leftarrow B[i]\text{.remove}(f) \\
    S\text{.addLast}((k, o))
  \end{align*}
\end{align*}
\]

Example

- Key range \([0, 9]\)

\[
\begin{array}{cccccccc}
  & 1, c & & 3, a & & 7, d & & 7, g & & 7, e \\
B & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}
\]

**Phase 1**

**Phase 2**

Properties and Extensions

- **Key-type Property**
  - The keys are used as 
    indices into an array 
    and cannot be arbitrary objects
  - No external comparator

- **Stable Sort Property**
  - The relative order of 
    any two items with the 
    same key is preserved 
    after the execution of the algorithm

- **Extensions**
  - Integer keys in the range \([a, b]\)
    \- Put entry \((k, o)\) into bucket 
      \( B[k - a] \)
  - String keys from a set \( D \) of 
    possible strings, where \( D \) has 
    constant size (e.g., names of 
    the 50 U.S. states)
    \- Sort \( D \) and compute the rank 
      \( r(k) \) of each string \( k \) of \( D \) 
      in the sorted sequence
    \- Put entry \((k, o)\) into bucket 
      \( B[r(k)] \)
Lexicographic Order

- A $d$-tuple is a sequence of $d$ keys $(k_1, k_2, \ldots, k_d)$, 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:
    $$(x_1, x_2, \ldots, x_d) < (y_1, y_2, \ldots, y_d)$$
    $$x_i < y_i \lor x_i = y_i \land (x_{i+1}, \ldots, x_d) < (y_{i+1}, \ldots, y_d)$$
  - I.e., the tuples are compared by the first dimension, then by the second dimension, etc.

Lexicographic-Sort

- Let $C_i$ 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 $stableSort$.

Algorithm $lexicographicSort(S)$

Input sequence $S$ of $d$-tuples
Output sequence $S$ sorted in lexicographic order

for $i \leftarrow d$ downto 1
$stableSort(S, C_i)$

Example:

$(7,4,6)$ $(5,1,5)$ $(2,4,6)$ $(2,1,4)$ $(3,2,4)$
$(2,1,4)$ $(2,4,6)$ $(3,2,4)$ $(5,1,5)$ $(7,4,6)$

Radix-Sort

- 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 $i$ are integers in the range $[0, N-1]$.
- Radix-sort runs in time $O(d(nN))$.

Algorithm $radixSort(S, N)$

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
bucketSort(S, N)

Radix-Sort for Binary Numbers

- Consider a sequence of $n$ $b$-bit integers $x = x_{b-1} \cdots x_1 x_0$.
- 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 $binaryRadixSort(S)$

Input sequence $S$ of $b$-bit integers.
Output sequence $S$ sorted in lexicographic order.

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

Sorting a sequence of 4-bit integers:

1001 0010 1101 0001
0010 1110 1101 0010
1101 1001 0001 0010
0001 1101 0010 1101
1110 0001 1110 1110