Bucket-Sort and Radix-Sort

Bucket-Sort (§ 4.5.1)

- Let be $S$ 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 $(x, a)$ into its bucket $B(x)$
  - 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

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

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 $stableSort$

Example:

- Key range $[0, 9]$
- $A = \{ (7, a), (3, e), (3, d), (7, g), (3, b), (7, e) \}$
- $B = \{ \{ (1, e), (3, a), (3, b), (7, d), (7, g), (7, e) \} \}$

Phase 1

Phase 2

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
    
    $$ (x_1, x_2, ..., x_d) < (y_1, y_2, ..., y_d) \iff \begin{cases} x_1 < y_1 \\ \text{or} \\ x_1 = y_1 \land (x_2, ..., x_d) < (y_2, ..., y_d) \end{cases} $$

I.e., the tuples are compared by the first dimension, then by the second dimension, etc.
**Radix-Sort (§ 4.5.2)**

- 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 \(i\) 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 \((0, \ldots, 0) \leq (x_1, \ldots, x_d) \leq (N-1, \ldots, N-1)\) for each tuple \((x_1, \ldots, x_d)\) in \(S\).

**Output** sequence \(S\) sorted in lexicographic order.

for \(i \gets d\) downto 1

\[ \text{bucketSort}(S, N) \]

---

**Radix-Sort for Binary Numbers**

- Consider a sequence of \(n\) \(b\)-bit integers \(X = X_{b-1} \ldots X_0\).
- 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 \(O(bn)\) 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 \(i \gets 0\) 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.

<table>
<thead>
<tr>
<th>1001</th>
<th>0010</th>
<th>1001</th>
<th>1001</th>
<th>0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>1110</td>
<td>1101</td>
<td>0010</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>0001</td>
<td>1110</td>
<td>1101</td>
<td>1001</td>
</tr>
</tbody>
</table>

---

**Note:** The images depict the sorting process visually, with arrows indicating the movement of elements through the sorting algorithm.