Linear Sorting: An Example

Suppose that $A[1..n]$ is a permutation of the values $1, 2, ..., n$

To sort $A$:

1. for $i \leftarrow 1$ to $n$
2. $B[A[i]] \leftarrow A[i]$

Running Time: $O(n)$
Another Example

Again, assume that \( A[1..n] \) is a permutation of the values \( 1, 2, ..., n \)

To Sort:
1. for \( i \leftarrow 1 \) to \( n \)
2. while \( A[i] \neq i \)
3. swap \( A[i] \leftrightarrow A[A[i]] \)

Running Time: \( O(n) \)
Counting Sort

Assumes that all input elements are integers in the range 1...k

Basic idea:
- determine the number of elements less than x, for each input element x
- place each element directly into its place in the output array
Counting-Sort

Counting-Sort(A,B,k)
1   for i ← 1 to k
2       Count[i] ← 0
3   for j ← 1 to length[A]
4       Count[A[j]] ← Count[A[j]]+1
5   for i ← 2 to k
6       Count[i] ← Count[i] + Count[i-1]
7   for j ← length[A] downto 1
8       B[Count[A[j]]] ← A[j]

Running Time: \( O(n+k) \)

Due to Line 7 it is stable!
Bucket Sort

Assume:
input elements are “distributed uniformly”
over known range, e.g. fractions in \([0, 1)\)

\[
\begin{array}{cccc}
0 & \frac{1}{n} & \frac{2}{n} & \cdots & 1 \\
\end{array}
\]

\(x \rightarrow \left\lfloor nx \right\rfloor\)

or integers in \([1..m]\)

\[
\begin{array}{cccc}
1 & \frac{m}{n} & \frac{2m}{n} & \cdots & m \\
\end{array}
\]

\(x \rightarrow \left\lfloor \frac{nx}{m} \right\rfloor\)
Bucket Sort

**Idea:**
- Divide the interval to $n$ buckets
- Distribute the $n$ input keys into the buckets
- Sort the numbers in each bucket
- Scan sorted buckets and combine them to produce the output array
Bucket-Sort

Bucket-Sort (A)

1. \( n \leftarrow \text{length}[A] \)
2. for \( i \leftarrow 1 \) to \( n \)
3. insert \( A[i] \) into list \( B\left[\frac{nA[i]}{m}\right] \)
4. for \( i \leftarrow 0 \) to \( n-1 \)
5. sort list \( B[i] \) with Insertion-Sort
6. Concatenate the lists \( B[0], B[1], \ldots, B[n-1] \) together in order
Figure 8.4  The operation of BUCKET-SORT. (a) The input array $A[1..10]$. (b) The array $B[0..9]$ of sorted lists (buckets) after line 5 of the algorithm. Bucket $i$ holds values in the half-open interval $[i/10, (i + 1)/10)$. The sorted output consists of a concatenation in order of the lists $B[0], B[1], \ldots, B[9]$. 
Bucket Sort

Running Time:
$O(n)$ to distribute elements to buckets

Uniformly distributed keys
$\Rightarrow E[\text{number of elements per bucket}] = 1$
$\Rightarrow O(1) \text{ time to sort each bucket (on average)}$

$O(n)$ to concatenate the sorted buckets

$\Rightarrow \text{TOTAL: } O(n)$
Problematic Case

Range 1..m
All the keys are 1
=> After putting them all in buckets

=> worst-case running time: $O(n^2)$
Lexicographic Order

\[ x = (x_d x_{d-1} \ldots x_2 x_1) \quad y = (y_d \ldots y_1) \]

\[ x < y \iff \text{for some } t \text{ we have that } x_t < y_t \text{ and } x_k = y_k \text{ for all } k > t \]

Example:

safe < sail < salt < salty
Radix-Sort

Radix-Sort \((A, d, n)\)

1   for \(i \leftarrow 1\) to \(d\)
2   do a stable sort of \(A\),
    according to digit \(i\)

**Running Time:** \(O(d \cdot T_{ss}(n))\)
### 6 bit numbers, radix=2 (base=4)

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>101101</td>
<td>001100</td>
<td>100001</td>
<td>000101</td>
</tr>
<tr>
<td>001100</td>
<td>111000</td>
<td>010010</td>
<td>001100</td>
</tr>
<tr>
<td>111001</td>
<td>110100</td>
<td>110100</td>
<td>001101</td>
</tr>
<tr>
<td>111000</td>
<td>101101</td>
<td>000101</td>
<td>010010</td>
</tr>
<tr>
<td>000101</td>
<td>111001</td>
<td>100111</td>
<td>100001</td>
</tr>
<tr>
<td>010010</td>
<td>000101</td>
<td>111000</td>
<td>100111</td>
</tr>
<tr>
<td>101111</td>
<td>001101</td>
<td>111001</td>
<td>101011</td>
</tr>
<tr>
<td>110100</td>
<td>100001</td>
<td>101011</td>
<td>101101</td>
</tr>
<tr>
<td>100111</td>
<td>010010</td>
<td>001100</td>
<td>101111</td>
</tr>
<tr>
<td>001101</td>
<td>101111</td>
<td>101101</td>
<td>110100</td>
</tr>
<tr>
<td>101011</td>
<td>100111</td>
<td>001101</td>
<td>111000</td>
</tr>
<tr>
<td>100001</td>
<td>101011</td>
<td>101111</td>
<td>111001</td>
</tr>
</tbody>
</table>
General Radix Sort

• Key has fields of different types
  \[ f_1 \ f_2 \ \ldots \ f_{k-1} \ f_k \]
  – e.g. date = day / month / year

Idea:
Use a stable sort:
- first sort according to \( f_k \) = least significant “digit”
- then sort on \( f_{k-1} \) = next significant “digit”

Note: Each sort must be a stable
How fast can we Sort?

- **Merge Sort:** $\Theta(n \lg n)$ worst-case
- **Heap Sort:** $\Theta(n \lg n)$ worst-case
- **Quick Sort:** $\Theta(n \lg n)$ average-case
- **Counting Sort:** $\Theta(n + k)$
- **Bucket Sort:** $O(n)$ on average
- **Radix Sort:** $O(n \cdot d)$
Comparison Sorts

The only operations allowed on input elements are comparisons between two elements:
- Bubble, Insertion, Merge, Heap and Quick sort

Can we do better than $n \log n$?

Not with comparisons sorts!
   And we’ll prove it now...

Notice: Bucket, Counting and Radix sort are NOT comparison sorts
Decision Tree Model

- Only consider the comparisons in the algorithm
- Each internal node = one comparison of the algorithm

- Start at the root, make the first comparison:
  - if outcome is ≤ - take the left branch
  - if outcome is > - take the right branch
- Repeat this at each internal node

- Every sorting algorithm induces a single tree
- Each ordering is represented by a leaf of its own
Decision Tree Model

Let $S$ be the set of permutations.

- $S_1$ is a subset of $S$ such that $x_i \leq x_j$.
- $S_2$ is a subset of $S$ such that $x_i > x_j$.

Then,

$$\max\{ |S_1|, |S_2| \} \geq \frac{|S|}{2}$$
Example: Insertion Sort, $n=3$
Permutations

Claim:
The height of the decision tree = worst-case complexity of any comparison based sort

• Each leaf represents one correct ordering

⇒ There must be $n!$ leaves: one for each possible permutation
Theorem:
Any decision tree for sorting \( n \) elements has height \( \Omega(n \log n) \)

Proof:
Let \( h \) = the height of the tree
The tree is binary \( \Rightarrow \) it has \( \leq 2^h \) leaves
But, we know the tree has \( n! \) leaves
\[ \Rightarrow 2^h \geq n! \]
\[ \Rightarrow h \geq \log(n!) = \Omega(n \log n) \]
More Lower Bounds

**Theorem 1:**
Any comparison-based algorithm for finding the minimum of $n$ keys, must use at least $n/2$ comparisons

**Proof:**
Every key must participate in at least one comparison
Each comparison compares 2 elements
$\Rightarrow$ At least $n/2$ comparisons have to be made
More Lower Bounds

**Theorem 2:**
Any comparison-based algorithm for finding the minimum of $n$ keys, must use at least $n-1$ comparisons

**Proof:**
Every other key must have won at least one comparison.
Each comparison produces at most one winner.
⇒ At least $n-1$ comparisons have to be made
Lower Bound for Min and Max

**Theorem:** Every comparison-based algorithm for finding both the minimum and the maximum of $n$ elements requires at least $\left(\frac{3n}{2}\right) - 2$ comparisons.

**Idea:** Use an adversary argument. $\max$ is the maximum and $\min$ is the minimum only if:
- every element except $\min$ has won at least 1
- every element except $\max$ has lost at least 1