COMP2521 24T1
Sorting Algorithms (IV)
Non-Comparison-Based Sorting Algorithms

Kevin Luxa
cs2521@cse.unsw.edu.au

\( n \log n \) lower bound
radix sort
All of the sorting algorithms so far have been \textit{comparison-based} sorts.

It can be shown that these algorithms require $\Omega(n \log n)$ comparisons. That is, they require at least $kn \log n$ comparisons for some constant $k$.

Why?
Suppose we need to sort 3 items.

Obviously, one comparison is not sufficient to sort them.
Suppose we need to sort 3 items.

Even two comparisons are not sufficient to sort them. Why?
If we have 3 items, there are $3! = 6$ ways to order them:

Assuming items are unique, one of these permutations is in sorted order.
Suppose we performed the following comparisons:

\[
\begin{align*}
\text{•} &< \text{★} \\
\text{★} &< \triangle
\end{align*}
\]

Four combinations of results are possible:

(true, true), (true, false), (false, true), (false, false)
The two comparisons create four groups, and each permutation of items belongs to one of these groups.

<table>
<thead>
<tr>
<th></th>
<th>&lt;</th>
<th>true</th>
<th>true</th>
<th>false</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing four groups with different combinations of symbols representing the comparisons and their outcomes.](attachment:diagram.png)
Mathematically,

If we have 3 items, then there are $3! = 6$ ways to order them. In other words, 6 possible permutations.

But if we only perform 2 comparisons, then there are only $2^2 = 4$ groups, so at least one group will contain more than one permutation.

We need at least 3 comparisons, because this creates $2^3 = 8$ groups, so each permutation can belong in its own group.
The $n \log n$ Lower Bound

If we have $n$ items, then there are $n!$ permutations.

If we perform $k$ comparisons, that creates up to $2^k$ groups.

So given $n$ items, we must perform enough comparisons $k$ such that

$$2^k \geq n!$$
So given \( n \) items, we must perform enough comparisons \( k \) such that
\[
2^k \geq n!
\]

Taking the \( \log_2 \) of both sides gives
\[
\log_2 2^k \geq \log_2 n!
\]
Since \( \log_2 2^k = k \), we get
\[
k \geq \log_2 n!
\]

Using Stirling’s approximation, we get
\[
k \geq n \log_2 n - n \log_2 e + O(\log_2 n)
\]
Removing lower-order terms gives
\[
k = \Omega(n \log_2 n)
\]
Therefore:

The theoretical lower bound on worst-case execution time for comparison-based sorts is $\Omega(n \log n)$. 
If we aren’t limited to just comparing keys, we can achieve better than $O(n \log n)$ worst-case time.

Non-comparison-based sorting algorithms exploit specific properties of the data to sort it.
Radix sort is a non-comparison-based sorting algorithm. It requires us to be able to decompose our keys into individual symbols (digits, characters, bits, etc.), for example:

- The key 372 is decomposed into (3, 7, 2)
- The key “sydney” is decomposed into (‘s’, ‘y’, ‘d’, ‘n’, ‘e’, ‘y’) 

Formally, each key $k$ is decomposed into a tuple $(k_1, k_2, k_3, ..., k_m)$. 
Ideally, the range of possible symbols is reasonably small, for example:

- Numeric: 0-9
- Alphabetic: a-z

The number of possible symbols is known as the **radix**, and is denoted by \( R \).

- Numeric: \( R = 10 \) (for base 10)
- Alphabetic: \( R = 26 \)

If the keys have different lengths, pad them with a suitable symbol, for example:

- Numeric: 123, 015, 007
- Alphabetic: “abc”, “zz”, “t”
Radix Sort

Method:
- Perform stable sort on $k_m$
- Perform stable sort on $k_{m-1}$
- ...
- Perform stable sort on $k_1$

Example:
Radix Sort

Pseudocode

`radixSort(A):`

**Input:** array A of keys where
each key consists of m symbols from an "alphabet"

 initialise R buckets // one for each symbol

 for i from m down to 1:
    empty all buckets
    for each key in A:
       append key to bucket key[i]

 clear A
 for each bucket (in order):
    for each key in bucket:
       append key to A
Assume alphabet is \{'a', 'b', 'c'\}, so $R = 3$.

We want to sort the array:

```
[“abc”, “cab”, “baa”, “a”, “ca”]
```

First, pad keys with blank characters:

```
[“abc”, “cab”, “baa”, “a␣␣”, “ca␣”]
```

Each key contains three characters, so $m = 3$. 
Radix Sort
Example

Array:

| “abc” | “cab” | “baa” | “a”   | “ca” |

Buckets:

- a
- b
- c
Array:

```
“abc”  “cab”  “baa”  “a”  “ca”
```

Buckets:

```
□   □   □   □
```

Radix Sort Example
Array:

“abc” “cab” “baa” “a_” “ca_”

Buckets:

[Diagram of buckets with corresponding letters and characters]
Radix Sort

Example

Array:

“abc”  “cab”  “baa”  “a”  “ca”

Buckets:

“abc”
Radix Sort
Example

Array:

“abc” “cab” “baa” “a” “ca”

Buckets:

□ □ □ c

□ a □ b

□ □ □ □

“abc”
Radix Sort

Example

Array:

“abc” “cab” “baa” “a” “ca”

Buckets:

a

b

“cab”

c

“abc”
Radix Sort

Example

Array:

“abc” | “cab” | “baa” | “a” | “ca”

Buckets:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“cab”</td>
<td>“abc”</td>
<td></td>
</tr>
</tbody>
</table>
Radix Sort

Example

Array:

“abc”  “cab”  “baa”  “a”  “ca”

Buckets:

a  “baa”

b  “cab”

c  “abc”
Radix Sort

Example

Array:

```
“abc”  “cab”  “baa”  “a”  “ca”
```

Buckets:

```
□

□
  “baa”

□
  “cab”

□
  “abc”
```
Radix Sort Example

Array:

```
|   | abc | cab | baa | a  | ca |
```

Buckets:

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>baa</td>
<td>abc</td>
</tr>
</tbody>
</table>
```
Array:

```
“abc”  “cab”  “baa”  “a""""  “ca""
```

Buckets:

```
iali    a    b    c
“a""""    “baa” “cab” “abc"
```
Radix Sort Example

Array:

```
"abc"  "cab"  "baa"  "aℓ"  "caℓ"
```

Buckets:

```
\\
"aℓ"  "baa"
"ca\u2113"

a  "baa"

b  "cab"

c  "abc"
```
Radix Sort Example

Array:

Buckets:

\[
\begin{array}{c}
\text{"a"} \\
\text{"ca"}
\end{array}
, \quad
\begin{array}{c}
\text{"baa"}
\end{array}
, \quad
\begin{array}{c}
\text{"cab"}
\end{array}
, \quad
\begin{array}{c}
\text{"abc"}
\end{array}
\]
Radix Sort

Example

Array:

Buckets:

“a”
“baa”
“cab”
“abc”

“ca”
“cab”
“abc”

Properties

Analysis

Radix Sort

Pseudocode

Example

$n \log n$ Lower Bound

n log n Lower Bound
Radix Sort

Example

Array:

```
“a
“ca
```

Buckets:

```
a “baa”
b “cab”
c “abc”
```

“a
“ca

n log n Lower Bound
Radix Sort
Pseudocode
Example
Analysis
Properties
Radix Sort

Example

Array:

```
| "a   " | "ca  " |   |   |
```

Buckets:

```
|   | "a   " | "ca  " |   |   |
```
```
| a | "baa " | "cab " |   |   |
```
```
| b | "abc " |   |   |   |
```

Radix Sort
Example

Array:

```
"a_" | "ca_" |   |   |
```

Buckets:

```
__           __           __           __
"a_" | "baa"   | "cab"   | "abc"
"ca_" |
```
Radix Sort
Example

Array:

```
“a”  “ca”  “baa”
```

Buckets:

```
  □
  “a”  “ca”

  a
  “baa”

  b
  “cab”

  c
  “abc”
```
Radix Sort

Example

Array:

```
"aabc" "ca" "baa"
```

Buckets:

```

  "aabc"
  "ca"

  a
  "baa"

  b
  "cab"

  c
  "abc"
```
Radix Sort

Example

Array:

| "a" | "ca" | "baa" | "cab" |

Buckets:

1. "a" "ca"

2. a "baa"

3. b "cab"

4. c "abc"
Radix Sort

Example

Array:

"a–a"  "ca"  "baa"  "cab"

Buckets:

"a"  "baa"  "cab"

"abc"
Radix Sort Example

Array:

```
"a\_\_\_" "ca\_\_" "baa" "cab" "abc"
```

Buckets:

```
\_ \_ "a\_\_\_" "ca\_\_"
\_ a "baa"
\_ b "cab"
\_ c "abc"
```
Radix Sort

Example

Array:

“a”  “ca”  “baa”  “cab”  “abc”

Buckets:

∪  ∪  a  b  c
Radix Sort

Example

Array:

```
"a_"    "ca_"    "baa"    "cab"    "abc"
```

Buckets:

```
∅    a    b    c
```
Radix Sort

Example

Array:

"a
ca
"baa
"cab
"abc"

Buckets:

\[ \square \quad \square \quad \square \quad \square \]
Radix Sort

Example

Array:

"a"  "ca"  "baa"  "cab"  "abc"

Buckets:

"a"  "a"  "b"  "c"
**Radix Sort**

**Example**

Array:

```
"a_
"ca_
"baa
"cab
"abc
```

Buckets:

```
  "a_
  a
  b
  c
```
Radix Sort

Example

Array:

“a”  “ca”  “baa”  “cab”  “abc”

Buckets:

- “a”  “ca”  “b”  “c”
Array:

```
"aゥゥ"  "caゥ"  "baa"  "cab"  "abc"
```

Buckets:

```
  "aゥゥ"
  "caゥ"

  a
  "caゥ"

  b

  c
```
Array:

```
"a" "ca" "baa" "cab" "abc"
```

Buckets:

```
"a"
"ca"
"baa"

"a"
"ca"

"b"

"c"
```
Radix Sort

Example

Array:

“a_{a}”  “ca_{a}”  “baa”  “cab”  “abc”

Buckets:

“a_{a}”

“ca_{a}”  “baa”

b

c
Radix Sort

Example

Array:

"a", "ca", "baa", "cab", "abc"

Buckets:

- "a": "a, ca, baa, cab"
- "b": "abc"
- "c": empty
Radix Sort

Example

Array:

“a→”  “ca”  “baa”  “cab”  “abc”

Buckets:

“a→”  “ca”  “baa”  “cab”  “b”  “c”
Radix Sort

Example

Array:

“a” “ca” “baa” “cab” “abc”

Buckets:

“a” “ca” “baa” “cab”

“abc”

“c”
Radix Sort

Example

Array:

```
[ ] [ ] [ ]
```

Buckets:

```
[ ]
  "a_
  "ca_
   "baa"
   "cab"
  "abc"

[ ]
  "b_
  "abc"

[ ]
  "c_
```

Radix Sort
Pseudocode
Example
Analysis
Properties

$n \log n$ Lower Bound
Radix Sort Example

Array:

Buckets:

"a, ca, baa, cab"

"a, ca, baa, cab"

"b, abc"

"c"
Radix Sort

Example

Array:

"a\text{a}\text{a}"

Buckets:

\begin{itemize}
  \item \text{a}
    \begin{itemize}
      \item \text{ca}\text{a}\text{a}
      \item \text{baa}
      \item \text{cab}
    \end{itemize}
  \item \text{b}
    \begin{itemize}
      \item \text{abc}
    \end{itemize}
  \item \text{c}
\end{itemize}
Radix Sort

Example

Array:

\[
\begin{array}{c}
\text{“a”} \\
\text{“a”} \\
\text{“ca”} \\
\text{“baa”} \\
\text{“cab”} \\
\end{array}
\]

Buckets:

\[
\begin{array}{c}
\text{a} \\
\text{“ca”} \\
\text{“baa”} \\
\text{“cab”} \\
\text{b} \\
\text{“abc”} \\
\text{c} \\
\end{array}
\]
Radix Sort
Example

Array:

```
<table>
<thead>
<tr>
<th>“a“</th>
<th>“ca“</th>
</tr>
</thead>
</table>
```

Buckets:

```
<table>
<thead>
<tr>
<th></th>
<th>“a“</th>
<th>&quot;ca“</th>
<th>“b“</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“baa”</td>
<td>“cab”</td>
<td>“abc”</td>
</tr>
</tbody>
</table>
```

Analysis
Radix Sort
Example

Array:

“a” “ca” “baa”

Buckets:

“a” “ca” “baa” “cab”

“b” “abc”

“c”
Radix Sort Example

Array:

```
  "a
  "ca
  "baa
  "cab
```

Buckets:

```
  a
  "ca"
  "baa"
  "cab"
  b
  "abc"
  c
```
Radix Sort

Example

Array:

```
"a_" | "ca_" | "baa" | "cab"
```

Buckets:

```
|  "a_"          |
| "ca_"          |
| "baa"          |
| "cab"          |
|                |
| "abc"          |
|                |
| "c"            |
```
Radix Sort

Example

Array:

```
"a", "ca", "baa", "cab", "abc"
```

Buckets:

```
A
  "a"
  "ca"
  "baa"
  "cab"

B
  "abc"

C
```

Properties

Radix Sort

Pseudocode

Example

Analysis

$n \log n$ Lower Bound
Radix Sort

Example

Array:

```
“a”, “ca”, “baa”, “cab”, “abc”
```

Buckets:

```
“a”, “ca”, “baa”, “cab”,
“ abc”,
“ b”,
“ c”,
```
Radix Sort

Example

Array:

```
“a”
“ca”
“baa”
“cab”
“abc”
```

Buckets:

```
   □
   □
   □
   □
   □
   □
   □
   □
```

Properties
Radix Sort

Example

Array:
```
"a
ca
"baa
"cab
"abc"
```

Buckets:
```
□
a
b
c
```
Radix Sort Example

Array:

```
“a
“ca
“baa
“cab
“abc
```

Buckets:

```
[ ]
[ a ]
[ b ]
[ c ]
```
Radix Sort

**Example**

**Array:**

- "a_\_

- "ca_\_

- "baa"

- "cab"

- "abc"

**Buckets:**

- a

  - "a_\_

- b

- c
### Radix Sort Example

#### Pseudocode

#### Example

**Array:**

```
"aabc"  "caab"  "baac"  "cab"  "abc"
```

**Buckets:**

```

   □
   □
   □
   □
   □
```

"aabc"
Radix Sort

Example

Array:

```
"aaaa"  "caaa"  "baa"  "cab"  "abc"
```

Buckets:

```
□   □   □   □   □
□    □   □    □  □
□  "aaaa"  □    □  □
□   □   □    □  □
□   □   □    □  □
```
Radix Sort

Example

Array:

“a₉₉”  “ca₉”  “baₐ”  “cab”  “abc”

Buckets:

аУ  a  b  c

“a₉₉”  "ca₉"
Radix Sort

Example

Array:

```
"a", "ca", "baa", "cab", "abc"
```

Buckets:

```
\[
\begin{array}{c}
\text{a} \\
\text{"a"} \\
\text{baa} \\
\text{ca} \\
\text{c}
\end{array}
\]```
Radix Sort

Example

Array:

"a_{\text{"a"}}" "ca_{\text{"a"}}" "baa" "cab" "abc"

Buckets:

\text{"a"}_{\text{"a"}} \text{a}_{\text{"baa"}} \text{b}_{\text{"cab"}} \text{c}_{\text{"abc"}}
Radix Sort

Example

Array:

“a”  “ca”  “baa”  “cab”  “abc”

Buckets:

a  “a”  “baa”

b  “cab”

c  “ca”  “cab”
Array:

```
"a
ca
baa
cab
abc"
```

Buckets:

```

  |
  |
  "a

  a
  "a

  b
  "baa"

  c
  "ca"
  "cab"
```
Radix Sort

Example

Array:

```
"a
ca
baa
cab
abc"
```

Buckets:

```
□
a
  "a
  "abc"
b
  "baa"
c
  "ca
  "cab"
```
Radix Sort Example

Array:

Buckets:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“a”</td>
<td>“baa”</td>
<td>“ca”</td>
</tr>
<tr>
<td></td>
<td>“abc”</td>
<td></td>
<td>“cab”</td>
</tr>
</tbody>
</table>
Radix Sort

Example

Array:

Buckets:

\[
\begin{array}{l}
\text{a}
\begin{array}{l}
\text{“a”}
\text{“abc”}
\end{array}
\end{array}
\begin{array}{l}
\text{b}
\begin{array}{l}
\text{“baa”}
\end{array}
\end{array}
\begin{array}{l}
\text{c}
\begin{array}{l}
\text{“ca”}
\text{“cab”}
\end{array}
\end{array}
\end{array}
\]
Radix Sort

Example

Array:

Buckets:

a
“a♡”
“abc”

b
“baa”

c
“ca♡”
“cab”
Radix Sort

array:

```
"aabc"
"baa"
"ca"
"cab"
```

buckets:

- `a`: "abc"
- `b`: "baa"
- `c`: "caab"

Radix Sort

Example
Radix Sort

Example

Array:

```
"a" "abc"
```

Buckets:

```
\[
\begin{array}{c}
\text{Unit} \\
\text{a}
\end{array}
\begin{array}{c}
\text{a}
\end{array}
\begin{array}{c}
b
\end{array}
\begin{array}{c}
c
\end{array}
\]
\]
\end{array}
```

- Unit
- a
  - "a"
  - "abc"
- b
  - "baa"
- c
  - "ca"
  - "cab"
Radix Sort

Example

Array:

```
"a\_\_\_"  "abc"
```

Buckets:

```
\_

\_
a
"a\_\_\_"
"abc"

\_
b
"baa"

\_
c
"ca\_\_"
"cab"
```
Radix Sort Example

Array:

```
"a

"abc"

"baa"

```

Buckets:

```

a

"a

"abc"

b

"baa"

c

"ca

"cab"

```
Array:

```
“a
“abc
“baa
```

Buckets:

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>
| “a
“abc
| “baa
| “ca
“cab
```
Radix Sort

Example

Array:

```
“a
abc”
“baa”
“ca
”
```

Buckets:

```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|   | “a
abc”| “baa”| “ca
”|
|   | “a
abc”| “baa”| “ca
”|
```

Radix Sort

Pseudocode

Example

Analysis

Properties

$n \log n$ Lower Bound
Radix Sort

Pseudocode

Example

Array:

```
"a●"  "abc"  "baa"  "ca●"  "cab"
```

Buckets:

```
\n
a
  "a●"
  "abc"

b
  "baa"

c
  "ca●"
  "cab"
```

Analysis

Properties

$n \log n$ Lower Bound

Radix Sort

Pseudocode

Example

Array:
Radix Sort

Example

Array:

```
“a”  “abc”  “baa”  “ca”  “cab”
```

Buckets:

```
   a
   b
   c
```
Analysis:

- Array contains $n$ keys
- Each key contains $m$ symbols
- Radix sort uses $R$ buckets
- A single stable sort runs in time $O(n + R)$
- Radix sort uses stable sort $m$ times

Hence, time complexity for radix sort is $O(m(n + R))$.

- $\approx O(mn)$, assuming $R$ is small

Therefore, radix sort performs better than comparison-based sorting algorithms:

- When keys are short (i.e., $m$ is small) and arrays are large (i.e., $n$ is large)
Stable
All sub Sorts performed are stable

Non-adaptive
Same steps performed, regardless of sortedness

Not in-place
Uses $O(R + n)$ additional space for buckets and storing keys in buckets
Other Non-Comparison-Based Sorts

- Bucket sort
- MSD Radix Sort
  - The version shown was LSD
- Key-indexed counting sort
- ...and others
Lower Bound
Radix Sort
Pseudocode
Example
Analysis
Properties

https://forms.office.com/r/5c0fb4tvMb