COMP2521 23T3
Sorting Algorithms (IV)

Kevin Luxa
cs2521@cse.unsw.edu.au

non-comparison-based sorts
All of the sorting algorithms so far have been comparison-based sorts.

That is, they work by comparing whole keys. Knowing how to compare whole keys is all they need to be able to sort.

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:

\[
\begin{align*}
\text{○} & < \text{□} \\
\text{□} & < \text{▲}
\end{align*}
\]

Four combinations of results are possible:

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

<table>
<thead>
<tr>
<th></th>
<th>&lt;</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>true</td>
<td></td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 $2^2 = 4$ buckets, so at least one bucket will contain more than one permutation.

We need at least 3 comparisons, because this creates $2^3 = 8$ buckets, so each permutation can sit in its own bucket.
If we have $n$ items, then there are $n!$ permutations.

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

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 $(k_1, k_2, k_3, \ldots, k_m)$. 
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 \textit{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 character, for example:

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

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

Example:

```
  cat
  ace
  dog
  cog
  key
  buy
```

```
  cat
  ace
  dog
  cog
  key
  buy
```

```
  cat
  ace
  dog
  cog
  key
  buy
```

```
  cat
  ace
  dog
  cog
  key
  buy
```

```
  cat
  ace
  dog
  cog
  key
  buy
```
radixSort(A):
    Input: array A of keys where
    each key consists of m symbols from an "alphabet"

    initialise m buckets // one for each symbol

    for i = m down to 1 do
        empty all buckets
        for key in A do
            append key to bucket key[i]
        end for

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

We want to sort the array:

\[
\text{[“abc”, “cab”, “baa”, “a”, “ca”]}
\]

First, pad keys with blank characters:

\[
\text{[“abc”, “cab”, “baa”, “a␣”, “ca␣”]}
\]

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

Example

Array:

```
“abc”  “cab”  “baa”  “a_”  “ca_”
```

Buckets:

```
□    □    □    □    □
```

"abc"” “cab”’ “baa”” “a_”” “ca_”"
Radix Sort

Example

Array:

“abc” | “cab” | “baa” | “a” | “ca”

Buckets:

[a] | [b] | [c]
Radix Sort

Example

Array:

```
“ab c”  “cab”  “baa”  “a   ”  “ca   ”
```

Buckets:

```
□
□
□
□
```

Properties

Lower Bound

$n \log n$
Radix Sort

Example

Array:

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

Buckets:

```
ad
b
“abc”
```
Radix Sort

Example

Array:

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

Buckets:

```
□
□
□
□
```

“abc”
Radix Sort

Example

Array:

“abc”  “cab”  “baa”  “a”  “c”

Buckets:

□  a  b  c

“abc”  “cab”
Radix Sort

Example

Array:

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

Buckets:

```
□
□
□
□
```

```
□
□
□
□
```

```
□
□
“cab”
□
```

```
□
□
“abc”
□
```
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:

```
| □      | a      | b      | c      |
| "baa"  | "cab"  | "abc"  |
```
Radix Sort

Example

Array:

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

Buckets:

```
†  "a"  "baa"  "b"  "abc"
```
Array:

“abc”  “cab”  “baa”  “a”  “ca”

Buckets:

“a”  “baa”  “cab”  “abc”
Radix Sort
Example

Array:

“abc”  “cab”  “baa”  “a”  “ca”

Buckets:

“a”  “baa”  “cab”  “abc”
Radix Sort

Example

Array:

Buckets:

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

Example

Array:

Buckets:

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

"ca"
Radix Sort

Example

Array:

```
“a
“ca
```

Buckets:

```
“a”
“baa”
“cab”
“abc”
```
Radix Sort

Example

Array:

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

Buckets:

- “a”
  - “baa”
- “b”
  - “cab”
- “c”
  - “abc”
Radix Sort

Example

Array:

```
<table>
<thead>
<tr>
<th>&quot;a\underline{a}&quot;</th>
<th>&quot;ca\underline{a}&quot;</th>
</tr>
</thead>
</table>
```

Buckets:

```
| "a\underline{a}" | "baa" |
| "ca\underline{a}" | "cab" |
| "abc" |
```
Radix Sort

Example

Array:

```
"a
ca
baa"
```

Buckets:

```
\[\begin{array}{c}
\text{a} \\
\text{baa} \\
\text{cab} \\
\text{abc}
\end{array}\]
```
Radix Sort

Example

Array:

```
“a”  “ca”  “baa”  
```

Buckets:

```
□  □  □  □
“a”  “baa”  “cab”  “abc”

“ca”
```
Radix Sort

Example

Array:

```
| "a_" | "ca" | "baa" | "cab" |
```

Buckets:

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

Array:

```
"a_\_\_"  "ca_\_"  "baa"  "cab"
```

Buckets:

```
\_
"a_\_\_"  "ca_\_"
\_
"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" "ca" "baa" "cab" "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___"  "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"
```
Radix Sort

Example

Array:

“a” “ca” “baa” “cab” “abc”

Buckets:

“a” “ca”

b
c
Array:

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

Buckets:

```
[]
[]
[]
["ca\_\_"
 "bba"]
["
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
#
```
Array:

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>baa</td>
<td>cab</td>
<td>abc</td>
</tr>
</tbody>
</table>
```

Buckets:

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c</td>
</tr>
</tbody>
</table>
```
Radix Sort

Example

Array:

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

Buckets:

"a_" "ca_" "baa" "cab"

b
c
Radix Sort

Example

Array:

```
“a”, “ca”, “ba”, “cab”, “abc”
```

Buckets:

```
“a”, “ca”, “ba”, “cab”
```

```
b
```

```
c
```
Radix Sort

Example

Array:

```
“a” “ca” “baa” “cab” “abc”
```

Buckets:

```
“a”
“ca”
“baa”
“cab”
```

```
“abc”
```

```
```

```
```
Radix Sort Example

Array:

Buckets:

```
Array:

Buckets:

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

Example

Array:

Buckets:

“a”
“ca”
“baa”
“cab”

“abc”

“b”

“c”
Radix Sort

Example

Array:

```
“a

“ca

“baa

“cab
```

Buckets:

```
“a

“ca

“baa

“cab

“abc
```

```

b

“abc
```

```

c
```
Radix Sort Example

Array:

```
"a_ _ _"
```

Buckets:

```
_ _ _
"a_ _ _"
"ca_ _"
"baa"
"cab"
```

```
_ _ _
"abc"
```

```
_ _ _
```

```
c
```

Radix Sort

Example

Array:

| "a_\_
\_" | "ca_\_
\_" |       |       |
|------------------|------------------|-------|-------|

Buckets:

| "a_\_
\_" | a | "ca_\_
\_" | b | "abc" | c |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;baa&quot;</td>
<td></td>
<td>&quot;cab&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Radix Sort

Pseudocode

Example

Analysis

Properties

\( n \log n \) Lower Bound
Array:

```
<table>
<thead>
<tr>
<th></th>
<th>&quot;aabc&quot;</th>
<th>&quot;ca&quot;</th>
<th>&quot;baa&quot;</th>
</tr>
</thead>
</table>
```

Buckets:

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

Radix Sort

Example
Array:

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

Buckets:

```
    "a
    "ca
    "baa
    "cab

    a
    "ca"
    "baa"
    "cab"

    b
    "abc"

    c
```
Radix Sort

**Example**

**Array:**

```
“a–a” “ca” “baa” “cab”
```

**Buckets:**

```
“a–a”

a
“ca”
“baa”
“cab”

b
“abc”

C
```
Radix Sort

Example

Array:

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

Buckets:

```

  "a\_a\_"

  "ca\_"
    "baa"
    "cab"

  b
    "abc"

  c
```
Radix Sort

Example

Array:

```
"a_"  "ca"  "baa"  "cab"  "abc"
```

Buckets:

```
  "a_"  "ca"  "baa"  "abc"
  "a_"  "ca"  "cab"
    "b"  "abc"
        "c"
```
Radix Sort Example

Array:

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

Buckets:

```
□ □ □ □
□ □ □ □
□ a □ □
□ b □ □
□ □ □ □
```

Properties
Radix Sort

Example

Array:

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

Buckets:

```
□
□
□
□
```

Properties
Radix Sort

Example

Array:

```
“a”
“ca”
“baa”
“cab”
“abc”
```

Buckets:
Radix Sort

Example

Array:

```
“a
ca
baa
cab
abc”
```

Buckets:

```
、“a
a
“a
b
b
abc”
```
Radix Sort

Example

Array:

```
“a––” “ca” “baa” “cab” “abc”
```

Buckets:

```
– 

a

“a––”

b

c
```
Radix Sort

Example

Array:

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

Buckets:

```
null
a
  “a”

b

null

  “ca”
```
Radix Sort

Example

Array:

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

Buckets:

```
[], a: "a", b, c: "ca"
```
Radix Sort

Example

Array:

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

Buckets:

```
  □
  a  “a※”
  b  “baa”
  c  “ca”
```
Radix Sort

Example

Array:

```
"aabc"  "cba"  "baa"  "cab"  "abc"
```

Buckets:

```
[]  ["aabc"]  ["baa"]  ["cab"]  ["abc"]
```
Radix Sort

Example

Array:

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

Buckets:

```
[ ]
[a]  "a"
[ ]  "baa"
[b]  "cab"
[c]  "abc"
```
Array:

```
| "a\\n\n\n" | "ca\\n\n\n" | "baa" | "cab" | "abc"
```

Buckets:

```
| a |
\n| "a\\n\n\n" |
| b |
\n| "baa" |
| c |
\n| "ca\\n\n\n" |
\n| "cab" |
```
Radix Sort

Example

Array:

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

Buckets:

```
\_

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

b
  "baa"

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

Example

Array:

```

```

Buckets:

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```
Radix Sort

Example

Array:

Buckets:

- a
  - “a”
  - “abc”
- b
  - “baa”
- c
  - “ca”
  - “cab”
Array:

Buckets:

- a:
  - "a_|_|"
  - "abc"
- b:
  - "baa"
- c:
  - "ca_|_"
  - "cab"
Radix Sort
Example

Array:
“aabc”

Buckets:

a
“aabc”
“abc”

b
“baa”

c
“caabc”
“cab”
Radix Sort
Example

Array:

```
“a””  “abc”
```

Buckets:

```
[ ]
```
```
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:

```
“aabc”  “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>
<tr>
<td>“aabc”</td>
<td>“baa”</td>
<td>“cab”</td>
<td></td>
</tr>
</tbody>
</table>
```
Radix Sort

Example

Array:

```
"a“"  "abc"  "baa"
```

Buckets:

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

Example

Array:

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

Buckets:

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

Properties

Pseudocode

Analysis

Example

$n \log n$ Lower Bound

Radix Sort

COMP2521

23T3
Radix Sort

Example

Array:

```
“a
“abc”
“baa”
“ca
“cab”
```

Buckets:

```

a
“a
“abc”

b
“baa”

c
“ca
“cab”
```
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)
Radix Sort
Properties

**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
https://forms.office.com/r/aPF09YHZ3X