Problem: \( U = \text{Universe of keys} \), so what?

\( W = \text{Universe of keys small!} \), so that

\( \{0, 1, 2, \ldots, k\} \)

\( A = (a_1, a_2, \ldots, a_N) \) sequence of keys (to sort) BIG

| 2 | 5 | 3 | 0 | 2 | 3 | 0 | 3 |

In most real world applications, is 3 an integer?

\( 3^{(i)} \) is George, \( 3^{(i)} \) is Andy, \( 3^{(i)} \) is Tom.

Other data is sometimes called a **handle**

It's easy \( O(n) \) time to compute

\[ C[i] = \text{number of records in } A \text{ with key } = i \]

\[ C \]

\[ 2 0 2 3 0 1 \]

0 1 2 3 4 5

Dream!

| 2 | 5 | 3 | 0 | 2 | 3 | 0 | 3 |

| 0 | 0 | 2 | 2 | 3 | 3 | 3 | 5 |
Given key like \[426, 295\] the computer can quickly address memory to access an array element whose index is the key.
Idea 1: Ask where should the 5 go?

Bad idea...
Radix Sort

329 657
457 436 355

First idea: organize by highest order digit

329 355 457 436 657 720 839

have to sort the sublists separately
Clever idea: First organize by lowest order digit!
Bucket sort

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>.78</td>
<td></td>
<td>.17</td>
<td>.26</td>
<td>.72</td>
<td>.94</td>
<td>.21</td>
<td>.12</td>
<td>.23</td>
<td>.68</td>
<td>.94</td>
</tr>
<tr>
<td>.17</td>
<td></td>
<td>.17</td>
<td>.26</td>
<td>.39</td>
<td></td>
<td></td>
<td>.68</td>
<td>.78</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.39</td>
<td></td>
<td></td>
<td>.26</td>
<td>.21</td>
<td>.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.26</td>
<td></td>
<td></td>
<td></td>
<td>.39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.94</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.68</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One Bucket
What data structure should you choose?
And how will it be used so the bucket is eventually sorted?

Requirements:
(1) Insert an unpredictable number of elements one by one.
(2) End up with the elements sorted.

Option 1

Option 2
How much should be the replacement of each array be when it fills up?

\[ +16 \quad +1 \quad \text{Squared} \quad ! \]

\[ \times 2 \quad \Rightarrow \text{we will analyze} \]
Total =

\[ N \left( \frac{1}{2^k} + \frac{1}{2^{k-1}} + \cdots + \frac{1}{2} \right) < 2N \]
Quick Sort

Input Set N keys

Pick one randomly
partition the rest

all ≤ .
all > .

Recursively until done!

A little clever programming makes the computer do the partitioning in \( \mathcal{O}(N \log N) \) really small and no extra space (except for controlling the recursions).

Running time? \( \mathcal{O}(N \log N) \) if you are lucky
\( \mathcal{O}(N^2) \) if NOT 😞
Bad Luck \( \Theta \left( \sum_{k=1}^{N} (N-k) \right) = \Theta(N^2) \)
"Intuitive" or "Heuristic" analysis

Very rough: When each split is about half and half, the same analysis as merge sort tells us the recursion depth is \( \lceil \log_2 N \rceil \) and \( O(N) \) work each level, total: \( CN \log_2 N \)

Less rough: If the split is 9:1 at worst,

\[
\begin{align*}
\text{Depth} & = \# \text{times} \left( \frac{9}{10} \right) \left( \frac{9}{10} \right) \left( \frac{9}{10} \right) \cdots \left( \frac{9}{10} \right) \cdot N \text{ to get } \leq 1 \\
\left( \frac{9}{10} \right)^L N & \approx 1 \quad N = \left( \frac{10}{9} \right)^L
\end{align*}
\]

\[
\log_{\left( \frac{10}{9} \right)} N = L = \frac{\log_2 B}{\log_2 A} = \frac{\log_2 N}{\log_2 \left( \frac{10}{9} \right)}
\]

\[
\log_A B = L \iff A^L = B \quad \text{What's } L? \]

Street Clobber fight: \( A \) with a \( \log_2 A \)

\[
\left( 2^{\log_2 A} \right)^L = B \\
2^{(\log_2 A) \cdot L} = B
\]

Now clobber \( B \): \( \left( \log_2 A \right) \cdot L = \log_2 B \)

\( \equiv \text{Solve for } L \)
\[
\log_{(10)} N = L = \frac{\log_2 B}{\log_2 A} \quad \text{or} \quad \frac{\log_2 N}{\log_2 (10/q)}
\]

\[
\frac{10}{q} - \frac{q}{q} + \frac{1}{q} = \frac{1}{q} = 1.1111111 \quad 0.15...
\]

\[
2^0 = 1.00000... \quad C \log_2 N
\]

\[
C = \frac{1}{0.15} = 6.57
\]
Towards a precise analysis:

Dream: Imagine A is sorted (it really isn't!)

The first random choice

• Gets compared to all these
  • Gets compared to all these

But

• are NEVER compared.
\[ a'_i \quad a'_j \quad i \neq j \]

Will \( a'_i \) and \( a'_j \) ever get compared to each other?

\[ a'_i \quad a'_j \]

IF and ONLY IF in that interval, either \( a'_i \) or \( a'_j \) was the first dog made red.

\[ P(\text{\( a'_i \) or \( a'_j \) one carrier}) = \frac{2}{(i-i+1) \mod (i,j)} \]
\[ f(N) \rightarrow \]
\[
\begin{array}{c}
1234 \\
\times 4321 \\
\hline
1234 \\
2468 \\
3702 \\
4938
\end{array}
\]

\[ 12 + 34 = 46 \]
\[ 43 + 21 = 64 \]

\[ \theta(N) \]
\[ 2944 \times 10,000 \]

\[ \frac{576}{714} \times 100 + 34.21 \]