Choosing Pivot:
- Smallest/largest element bad, why?
- Only one element (i.e. the pivot) will get sorted in each partition (RunTime in this case?)
- Random expensive to generate
- Median of 3: choose the median of leftmost, middle, rightmost elements.

Ex: 26 5 37 61 15 11 59 12 48 1

Median 3 (26, 15, 1) = 15

Partition (A, first, last, pivot):
1. Swap pivot with last element
2. I points to first element
3. J is element before pivot, plus 1
4. While I and J have not crossed:
   * Move I right, move J left until
     \[ A[i] > \text{pivot} \] and \[ A[j] < \text{pivot} \]
     * Swap \[ A[i], A[j] \]
5. Swap pivot with \[ A[i] \]
6. Return i
Example:

\[
26 \ 5 \ 37 \ 61 \ 15 \ 11 \ 59 \ 12 \ 48 \ 1
\]

\[
\text{move pivot out of the way}
\]

\[
26 \ 5 \ 37 \ 61 \ 11 \ 59 \ 12 \ 48 \ 15
\]

\[
i 
\]

\[
j < j
\]

\[
12 \ 5 \ 37 \ 61 \ 11 \ 59 \ 26 \ 48 \ 15
\]

\[
i \rightarrow i
\]

\[
j < j
\]

\[
12 \ 5 \ 11 \ 61 \ 137 \ 59 \ 26 \ 48 \ 15
\]

\[
i
\]

\[
j \rightarrow j
\]

\[
12 \ 5 \ 11 \ 61 \ 37 \ 59 \ 26 \ 48 \ 15
\]

\[
i
\]

\[
j
\]

Done since \( j < i \)

Swap pivot w/ A[i]

\[
12 \ 5 \ 11 \ 1 \ 15 \ 37 \ 59 \ 26 \ 48 \ 61
\]

\[
\text{Qsort} \ 1
\]

\[
\text{Qsort} \ 1
\]
Partition \((A, \text{first}, \text{last}, \text{pivot})\)  
//Partitions \(A\) around \(\text{pivot}\)

\[
\begin{align*}
\text{swap } & \text{pivot and } A[\text{last}] \\
i & = \text{first} \\
j & = \text{last} - 1 \\
\text{loop} & = \text{true} \\
\text{while(\text{loop})} \{ \\
\quad \text{while } (A[i] \leq \text{pivot}) \{ i++ \} \\
\quad \text{while } (A[j] \geq \text{pivot}) \{ j-- \} \\
\quad \text{if } (i < j) \quad // i \text{ and } j \text{ have not crossed} \\
\quad \text{swap } A[i], A[j] \\
\quad \text{else} \quad // i \text{ and } j \text{ have crossed} \\
\quad \text{loop} = \text{false} \\
\quad \text{swap } \text{pivot with } A[i] \\
\quad \text{return } i
\}
\end{align*}
\]

**Time:** \(O(n)\)

cutoff - For small list \((\leq 3)\) use insertionSort.

QuickSort\((A, \text{first}, \text{last})\)  
//Sorts \(A\) with cutoff of 3

\[
\begin{align*}
\text{if } (\text{last-first} < 3) \quad // |A| \leq 3 & \quad 3.95 \\
\text{InsertionSort}(A) \\
\text{else} \quad // |A| > 3 \\
\quad \text{pivot} = \text{med3 } (A, \text{first}, \text{last}) \\
\quad \text{split-point} = \text{Partition} (A, \text{first}, \text{last}, \text{pivot}) \\
\quad \text{QuickSort}(A, \text{first}, \text{split-point}-1) \\
\quad \text{QuickSort}(A, \text{split-point}+1, \text{last})
\end{align*}
\]
Runtime QuickSort?

Average-case: pivot always roughly halves the list.
\[ O(\log n) \] recursive calls, each takes \( O(n) \Rightarrow O(n \log n) \)

Worst-case: if pivot is always smallest or largest value, \( O(n) \) recursive calls \( \Rightarrow O(n^2) \)

Best-case: also \( O(n \log n) \) (sorted list, perfect pivot doesn't help).

cut-off: For small lists \( (n < 3) \), use insertion sort.

Insertion, merge, quick sorts are examples of general general sorting algorithms. Can we for specific

ex: \( A \) is list of \( n \) integers between 0 and \( m \):
\[ 0 \leq A[0], A[1], \ldots, A[n-1] < m \]

ex: \( n = 10 \)

3 1 4 1 5 9 2 5

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Sorted list: 1 1 2 3 4 5 5 9
BucketSort (A, m) // Sorts list A where each value in A

  count[0..m-1] = 0 // is in range [0,m-1].

  for i = 0 to n-1
      count[A[i]] ++;

  for i = 0 to m-1
      print i * count[i] times.

Time: O(m+n)

Bucket sort seems very restrictive (only integers).

How to sort:

0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68

0

1 -> [0.17, 0.12]

2 -> [0.26, 0.21, 0.23]

3 -> [0.39]

4

5

6 -> [0.68]

7 -> [0.78, 0.72]

8

9 -> [0.94]

Now sort each bucket

with insertion/quick sort

# items in bucket will be O(1)

(Since elements will be evenly distributed)

\[
\frac{\text{Total # elements}}{\text{# buckets}} \approx \frac{n}{m}
\]
Generalized Bucket Sort - distribute elements into an array of "buckets". Then, sort each bucket aka hash-sort.

(Can skip this example)

Radix Sort: A list of fixed-length integers
Start with least significant digit
Continuously sort by next least significant digit

ex: 170 45 75 90 2 24 802 66

170 045 075 090 002 024 802 066

170 090 002 802 024 045 075 066

002 802 024 045 066 170 075 090
Radix Sort

Suppose sorting fixed-length strings alphabetically:

cats
cost
cent
cups
coot

How?
Sort letter by letter starting from first letter

cats  cats  coot

cost  cent  cent  ...  (not working...)
cent  cost  cups
cups  coat  cost
coot  cups  cats

Problem: When we sort by next letter to the right, we "unsort" previous letters. Instead, start with last letter:

cats  coat  cats

cups  cent  cent

cost  cups  coat
cent  cost  cost
coot  cats  cups
Why does this work?

Now, if we "unsort" letters to the right, it's OK since letter to the left is more important in the sort.

**Radix Sort - A** is a list of fixed-length strings or integers.
Start with least significant character or digit and continuously sort by next significant character, digit.

What if not fixed-length?

cats Suppose cent should be sorted before cents.
costly cents ⇒ Add a character that is alphabetically before 'a' to the end of shorter
cent... cups strings.

cats- cats- cats- cats- cats-
costly cents- cent- cups- cent- cents-
cent- cups- costly costly costly
cups- costly cents- cents- cups-