Quick Sort: Runtime
Goals

Understand the best-case and worst-case runtime analysis of quicksort.

Know the average-case runtime of quicksort.
Merge vs Quick

"real work" done here

"real work" done here
Quicksort: Runtime

```python
/** quicksort A[st..end]*/
quickSort(A, st, end):
    if (small):
        return
    mid = partition(A, st, end)
quickSort(A, st, mid)
quickSort(A, mid+1, end)

# (nothing to do!)
```

$O(1)$

$O(??)$
Partition: Runtime

```
partition(A, start, end)
    initialize i, j
    choose pivot
    swap pivot to A[0]
    while [[] section != []
        # process A[i]:
        if <= p:
            move to <= p section
        else:
            move to > p section
```

\[ O(1) \]
\[ n \times O(1) \]

Total: \( O(n) \), where \( n = \text{end} - \text{start} \).
**Runtime: Best case**

**Best case:**
- pivot is the median of the array
- partition splits the array exactly in half
- same analysis as merge sort

Best-case runtime: $O(n \log n)$
**Runtime: Worst case**

**Worst case:**
- pivot is the minimum or maximum of the array
- partition splits the array into 1 and n-1.

```
+-------+-------+-------+-------+-------+
|       |       |       |       |       |
| p     | p     | p     | p     | p     |
| n work| n-1 work| n-2 work| n-3 work| n-4 work|
|       |       |       |       |       |
|       |       |       |       |       |
| n levels| n-5 work|       |       |
```

Worst-case runtime: $O(n^2)$
Runtime: Average case

**Average** case:
- more like best case than worst case (this is rare!)
- full analysis is out of scope, but you should know this result

<table>
<thead>
<tr>
<th>O(log n) levels</th>
<th>n work</th>
<th>n work</th>
<th>n work</th>
<th>n work</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O(n/2)</td>
<td>O(n/2)</td>
<td>O(n/4)</td>
<td>O(n/4)</td>
</tr>
<tr>
<td></td>
<td>O(n/4)</td>
<td>O(n/4)</td>
<td>O(n/4)</td>
<td>O(n/4)</td>
</tr>
</tbody>
</table>

Average-case runtime: $O(n \log n)$