What does it do?

A[r] is called the **pivot**

Partitions the elements

A[p…r-1] into two sets, those

≤ pivot and those > pivot

Operates in place

Final result:

```
A          P  pivot  R
  ≤ pivot  > pivot
```

```
PARTITION(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4          i ← i + 1
5      swap A[i] and A[r]
6    swap A[i + 1] and A[r]
7    return i + 1
```
Partition($A, p, r$)
1 $i \leftarrow p - 1$
2 for $j = p$ to $r - 1$
3 if $A[j] \leq A[r]$
4 $i \leftarrow i + 1$
5 swap $A[i]$ and $A[r]$
6 swap $A[i + 1]$ and $A[r]$
7 return $i + 1$
Partition($A$, $p$, $r$)
1 $i \leftarrow p - 1$
2 for $j = p$ to $r - 1$
3 \[\text{if } A[j] \leq A[r] \]
4 \[i \leftarrow i + 1\]
5 swap $A[i]$ and $A[r]$
6 swap $A[i + 1]$ and $A[r]$
7 return $i + 1$

Partition($A$, $p$, $r$)
1 $i \leftarrow p - 1$
2 for $j = p$ to $r - 1$
3 \[\text{if } A[j] \leq A[r] \]
4 \[i \leftarrow i + 1\]
5 swap $A[i]$ and $A[r]$
6 swap $A[i + 1]$ and $A[r]$
7 return $i + 1$
2/1/24

13

\[
\begin{array}{cccccccc}
& & & & & & & \\
i & j & \downarrow & & & & & \\
\ldots & 5 & 7 & 1 & 2 & 8 & 4 & 3 & 6 \ldots \\
\downarrow & & & & & & & \\
p & & & & & & & \\
\end{array}
\]

**Partition** \((A, p, r)\)
1. \(i \leftarrow p - 1\)
2. for \(j \leftarrow p\) to \(r - 1\)
3. if \(A[j] \leq A[r]\)
4. \(i \leftarrow i + 1\)
5. swap \(A[i]\) and \(A[j]\)
6. swap \(A[i + 1]\) and \(A[r]\)
7. return \(i + 1\)

14

\[
\begin{array}{cccccccc}
& & & & & & & \\
i & j & \downarrow & & & & & \\
\ldots & 5 & 1 & 7 & 2 & 8 & 4 & 3 & 6 \ldots \\
\downarrow & & & & & & & \\
p & & & & & & & \\
\end{array}
\]

**Partition** \((A, p, r)\)
1. \(i \leftarrow p - 1\)
2. for \(j \leftarrow p\) to \(r - 1\)
3. if \(A[j] \leq A[r]\)
4. \(i \leftarrow i + 1\)
5. swap \(A[i]\) and \(A[j]\)
6. swap \(A[i + 1]\) and \(A[r]\)
7. return \(i + 1\)

15

\[
\begin{array}{cccccccc}
& & & & & & & \\
i & j & \downarrow & & & & & \\
\ldots & 5 & 1 & 7 & 2 & 8 & 4 & 3 & 6 \ldots \\
\downarrow & & & & & & & \\
p & & & & & & & \\
\end{array}
\]

**Partition** \((A, p, r)\)
1. \(i \leftarrow p - 1\)
2. for \(j \leftarrow p\) to \(r - 1\)
3. if \(A[j] \leq A[r]\)
4. \(i \leftarrow i + 1\)
5. swap \(A[i]\) and \(A[j]\)
6. swap \(A[i + 1]\) and \(A[r]\)
7. return \(i + 1\)

16

\[
\begin{array}{cccccccc}
& & & & & & & \\
i & j & \downarrow & & & & & \\
\ldots & 5 & 1 & 7 & 2 & 8 & 4 & 3 & 6 \ldots \\
\downarrow & & & & & & & \\
p & & & & & & & \\
\end{array}
\]

**Partition** \((A, p, r)\)
1. \(i \leftarrow p - 1\)
2. for \(j \leftarrow p\) to \(r - 1\)
3. if \(A[j] \leq A[r]\)
4. \(i \leftarrow i + 1\)
5. swap \(A[i]\) and \(A[j]\)
6. swap \(A[i + 1]\) and \(A[r]\)
7. return \(i + 1\)
What's happening?
21

```
Partition(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4 i ← i + 1
5 swap A[i] and A[r]
6 swap A[i + 1] and A[r]
7 return i + 1
```

22

```
Partition(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4 i ← i + 1
5 swap A[i] and A[r]
6 swap A[i + 1] and A[r]
7 return i + 1
```

23

```
Partition(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4 i ← i + 1
5 swap A[i] and A[r]
6 swap A[i + 1] and A[r]
7 return i + 1
```

24

```
Partition(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
4 i ← i + 1
5 swap A[i] and A[r]
6 swap A[i + 1] and A[r]
7 return i + 1
```
Partition running time?

$\Theta(n)$

**Partition**($A, p, r$)
1. $i \leftarrow p - 1$
2. for $j \leftarrow p$ to $r - 1$
   3. if $A[j] \leq A[r]$
   4. $i \leftarrow i + 1$
   5. swap $A[i]$ and $A[j]$
6. swap $A[i + 1]$ and $A[r]$
7. return $i + 1$

**QuickSort**($A, p, r$)
1. if $p < r$
2. $q \leftarrow \text{Partition}(A, p, r)$
3. QuickSort($A, p, q - 1$)
4. QuickSort($A, q + 1, r$)

8 5 1 3 6 2 7 4
QuickSort($A, p, r$)
1. if $p < r$
2. $q \leftarrow \text{Partition}(A, p, r)$
3. QuickSort($A, p, q - 1$)
4. QuickSort($A, q + 1, r$)
QuickSort(A, p, r)
1 if p < r
2 q ← Partition(A, p, r)
3 QuickSort(A, p, q - 1)
4 QuickSort(A, q + 1, r)
What happens here?
Some observations

Divide and conquer: different than MergeSort – do the work before recursing

How many times is/can an element be selected as a pivot?

What happens after an element is selected as a pivot?
Is Quicksort correct?
Assuming Partition is correct

Proof by induction
- Base case: Quicksort works on a list of 1 element
- Inductive case:
  - Assume Quicksort sorts arrays for arrays of smaller \(< n\) elements,
    show that it works to sort \(n\) elements
  - If partition works correctly then we have:
    and, by our inductive assumption, we have:

\[A\]

\[\text{pivot}\]

\[\text{sorted} \quad \text{sorted}\]

\[\leq \text{pivot} \quad > \text{pivot}\]

Running time of Quicksort?
Worst case?
Each call to Partition splits the array into an empty array and \(n\) array

Quicksort: Worse case running time

\[T(n) = T(n-1) + \Theta(n)\]

Which is? \(\Theta(n^2)\)

When does this happen?
- sorted
- reverse sorted
- near sorted/reverse sorted

Quicksort best case?
Each call to Partition splits the array into two equal parts

\[T(n) = 2T(n/2) + \Theta(n)\]

\(\Theta(n \log n)\)

When does this happen?
- random data?
Quicksort Average case?

How close to “even” splits do they need to be to maintain an $\Theta(n \log n)$ running time?

Say the Partition procedure always splits the array into some constant ratio $b$-to-$a$, e.g. 9-to-1

What is the recurrence?

$$T(n) \leq T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn$$
What is the depth of the tree?

Leaves will have different heights
Want to pick the deepest leaf
Assume $a < b$

Cost of the tree

Cost of each level $\leq cn$?
Cost of the tree

Cost of each level ≤ cn
Times the maximum depth
\[ O(n \log_{\frac{a+b}{b}} n) \]

Why not?
\[ \Theta(n \log_{\frac{a+b}{b}} n) \]

QuickSort average case: take 2

What would happen if half the time Partition produced a “bad” split and the other half “good”?

\[ T(n) = 2T\left(\frac{n-1}{2}\right) + \Theta(n) \]

Quicksort average case: take 2

```
T(i)
  | cn
  v
  "bad" split
  \v
T(n-1)
```

```
T(i)
  | cn
  v
  "good" 50/50 split
  \v
T\left(\frac{n-1}{2}\right)
T\left(\frac{n-1}{2}\right)
```

\[ T(n) = T(i) + T\left(\frac{n-1}{2}\right) + T\left(\frac{n-1}{2}\right) + \Theta(n) + \Theta(n-1) \]
**Quicksort average case: take 2**

![Diagram of the quicksort algorithm with average case analysis]

- We absorb the “bad” partition. In general, we can absorb any constant number of “bad” partitions.

**How can we avoid the worst case?**

Inject randomness into the data

```
RANDOMIZED-PARTITION(A, p, r)
1   i ← RANDOM(p, r)
2   swap A[r] and A[i]
3   return PARTITION(A, p, r)
```

---

**What is the running time of randomized Quicksort?**

Worst case?

\[ \Theta(n^2) \]

Still could get very unlucky and pick “bad” partitions at every step.

---

**Sorting bounds**

Mergsort is \( \Theta(n \log n) \)

Quicksort is \( O(n \log n) \) on average

Can we do better?
**Comparison-based sorting**

Sorted order is determined based **only** on a comparison between input elements:
- $A[i] \leq A[j]$
- $A[i] \geq A[j]$

Do any of the sorting algorithms we've looked at use additional information?
- No
- All the algorithms we've seen are comparison-based sorting algorithms

In Java (and many languages) for a class of objects to be sorted we define a comparator

What does it do?
- Just compares any two elements
- Useful for comparison-based sorting algorithms

Can we do better than $O(n \log n)$ for comparison based sorting approaches?
**Decision-tree model**

Full binary tree representing the comparisons between elements by a sorting algorithm

Internal nodes contain indices to be compared

Leaves contain a complete permutation of the input

Tracing a path from root to leave gives the correct reordering/permutation of the input for an input.

Is $12 \leq 7$ or is $12 > 7$?
Is 12 \leq 7 or is 12 > 7?

Is 12 \leq 3 or is 12 > 3?
A decision tree model

[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

[12, 7, 3] → 3, 2, 1
A decision tree model

[7, 12, 3]

A decision tree model

[7, 12, 3]

A decision tree model

[7, 12, 3]

A decision tree model

[7, 12, 3]
How many leaves are in a decision tree?

Leaves must have all possible permutations of the input.

What if decision tree model didn’t?

Some input would exist that didn’t have a correct reordering.

Input of size $n$, $n!$ leaves.

A lower bound

What is the worst-case number of comparisons for a tree?
A lower bound

The longest path in the tree, i.e. the height

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

What is the maximum number of leaves a binary tree of height $h$ can have?

A complete binary tree has $2^h$ leaves

$2^h \geq n!$

$h \geq \log n!$

$h = \Omega(n \log n)$ from group work! 😊

Can we do better?