What does it do?

Algorithm: Partition

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 $A[r]$
6. swap $A[i + 1]$ and $A[r]$
7. return $i + 1$

A[r] is called the *pivot*

Partitions the elements $A[p..r-1]$ into two sets, those $\leq$ pivot and those $> pivot$

Operates in place

Final result:

- $A[p..r]$
\textbf{Partition}(A, p, r)
1 \(i \leftarrow p - 1\)
2 \(\text{for } j \leftarrow p \text{ to } r - 1\)
3 \(\text{if } A[j] \leq A[r]\)
4 \(i \leftarrow i + 1\)
5 \(\text{swap } A[i] \text{ and } A[r]\)
6 \(\text{swap } A[i + 1] \text{ and } A[r]\)
7 \(\text{return } i + 1\)
Partition($A, p, r$)
1  $i \leftarrow p - 1$
2  for $j \leftarrow p$ to $r - 1$
3     if $A[j] \leq A[p]$
4         $i \leftarrow i + 1$
5     end
6  swap $A[i]$ and $A[p]$
7  return $i + 1$
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 $A[r]$
6  swap $A[i + 1]$ and $A[r]$
7  return $i + 1$
What’s happening?
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 \(i + 1\)
Is Partition correct?
Partitions the elements $A[p...r-1]$ in to two sets, those $\leq$ pivot and those $> pivot$?

Loop Invariant:

$$A[p...i] \leq A[r] \text{ and } A[i+1...j-1] > A[r]$$

Proof by induction

Loop Invariant: $A[p...i] \leq A[r] \text{ and } A[i+1...j-1] > A[r]$

Base case: $A[p...i]$ and $A[i+1...j-1]$ are empty

Assume it holds for $j-1$, two cases:
  - $A[p...i]$ remains unchanged
  - $A[i+1...j-1]$ contains one additional element, $A[j]$ which is $> A[r]$

```plaintext
PARTITION(A, p, r)
1 i ← p - 1
2 for j ← p to r - 1
3 if $A[j] \leq A[r]$
4 i ← i + 1
5 swap $A[i]$ and $A[j]$
6 swap $A[i+1]$ and $A[r]$
7 return $i + 1$
```
Proof by induction

Loop Invariant: \(A[p...i] \leq A[r]\) and \(A[i+1...j-1] > A[r]\)

2nd case:
- \(A[j] \leq A[r]\)
- \(i\) is incremented
- \(A[i]\) swapped with \(A[j] = A[p...j]\) contains one additional element which is \(\leq A[i]\)
- \(A[i+1...j-1]\) will contain the same elements, except the last element will be the old first element

\[
\text{PARTITION}(A, p, r)
\]

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

Partition running time?

\(\Theta(n)\)

\[
\text{PARTITION}(A, p, r)
\]

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

Quicksort

\[
\text{QUICKSORT}(A, p, r)
\]

1. \(\text{if } p < r\)
2. \(q \leftarrow \text{PARTITION}(A, p, r)\)
3. \(\text{QUICKSORT}(A, p, q - 1)\)
4. \(\text{QUICKSORT}(A, q + 1, r)\)

\[
\text{PARTITION}(A, p, r)
\]

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

\[
\text{QUICKSORT}(A, p, r)
\]

1. \(\text{if } p < r\)
2. \(q \leftarrow \text{PARTITION}(A, p, r)\)
3. \(\text{QUICKSORT}(A, p, q - 1)\)
4. \(\text{QUICKSORT}(A, q + 1, r)\)
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?

```
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)
```
Some observations

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

How many times is/can an element selected for 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
  - If partition works correctly then we have:
  - and, by our inductive assumption, we have:

```
A ≤ pivot ≤ pivot > pivot
```

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 to maintain an $O(n \log n)$ 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?

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

Assume $a < b$

\[
\left( \frac{b}{a+b} \right)^d n = 1
\]

\[
\vdots
\]

\[
d = \log_{a+b} n
\]
Cost of the tree

Cost of each level \( \leq cn \)

Times the maximum depth

\[ O(n \log_{a+b} n) \]

Why not?

\[ \Theta(n \log_{a+b} n) \]

Quicksort average case: take 2

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

“good” 50/50 split

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

Quicksort average case: take 2

Recursion cost

Partition cost

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

- **“bad” split**
  - \( T(n) \)
  - \( c \)

- **“good” 50/50 split**
  - \( c(n-1) \)
  - \( T(\frac{n-1}{2}) \)

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

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. \( i \leftarrow \text{Random}(p, r) \)
2. \( \text{swap } A[r] \text{ and } A[i] \)
3. \( \text{returnPartition}(A, p, r) \)

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

- **Worst case?** \( O(n^2) \)
- 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.

- $[3, 12, 7] \rightarrow [1,3,2] \rightarrow [3,7,12]$  
- $[7, 3, 12] \rightarrow [2,1,3] \rightarrow [3,7,12]$  

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

A decision tree model

Is 12 ≤ 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, 7, 12]

3, 2, 1
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

Can we do better?