Sorting Concluded

David Kauchak
CS140
Fall 2024

Administrative

Assignment 1 grading

Assignment 2

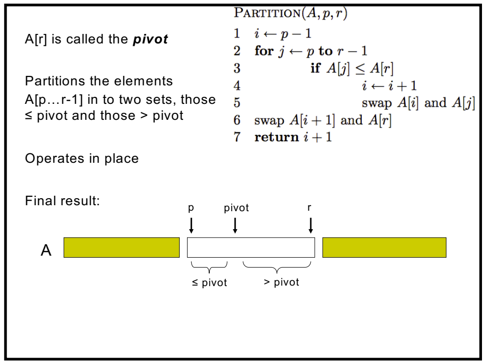
- Available now
- Must work with new partner (last time)
- Start early!

Finding a partner

Why does the master method work?

```
PARTITION($A, p, r$)
$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$
    if $A[j] \leq A[r]$
        $i \leftarrow i + 1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i + 1$
```

What does it do?



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

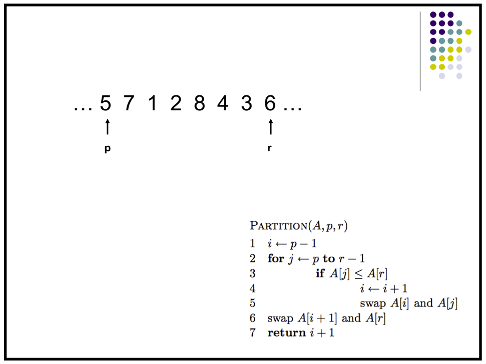
Partitions the elements A[p…r-1] in to two sets, those ≤ pivot and those > pivot

Operates in place

Final result:

```
PARTITION($A, p, r$)
    $i \leftarrow p - 1$
    for $j \leftarrow p$ to $r - 1$
        if $A[j] \leq A[r]$
            $i \leftarrow i + 1$
            swap $A[i]$ and $A[j]$
    swap $A[i+1]$ and $A[r]$
    return $i + 1$
```

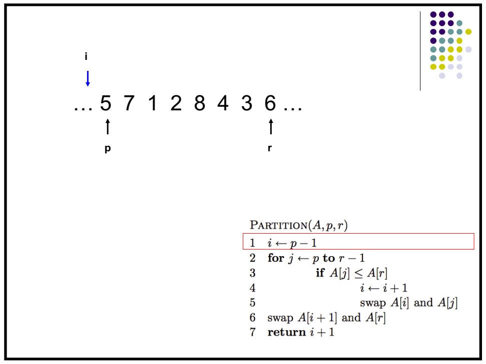
5



… 5 7 1 2 8 4 3 6 …

```
PARTITION($A, p, r$)
    $i \leftarrow p - 1$
    for $j \leftarrow p$ to $r - 1$
        if $A[j] \leq A[r]$
            $i \leftarrow i + 1$
            swap $A[i]$ and $A[j]$
    swap $A[i+1]$ and $A[r]$
    return $i + 1$
```

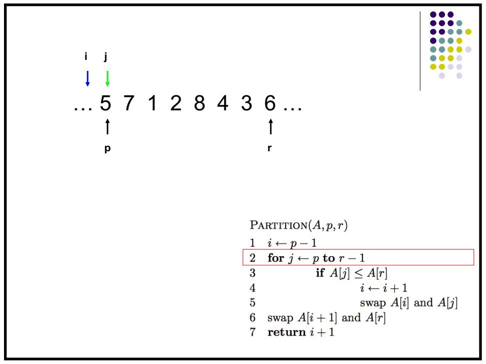
6



… 5 7 1 2 8 4 3 6 …

```
PARTITION($A, p, r$)
    $i \leftarrow p - 1$
    for $j \leftarrow p$ to $r - 1$
        if $A[j] \leq A[r]$
            $i \leftarrow i + 1$
            swap $A[i]$ and $A[j]$
    swap $A[i+1]$ and $A[r]$
    return $i + 1$
```

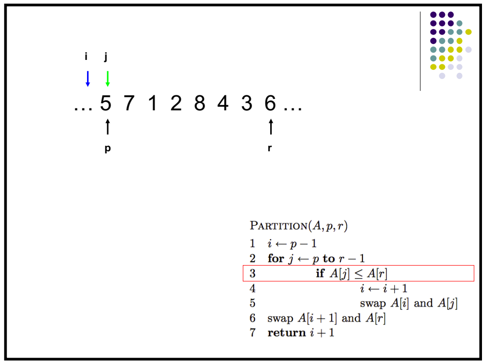
7



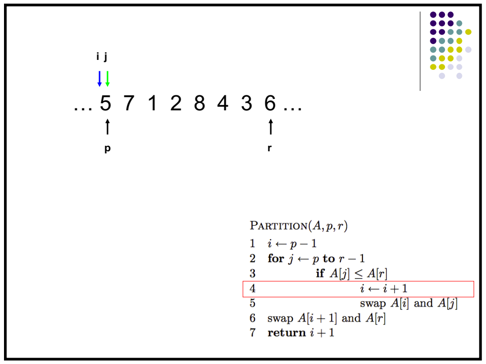
… 5 7 1 2 8 4 3 6 …

```
PARTITION($A, p, r$)
    $i \leftarrow p - 1$
    for $j \leftarrow p$ to $r - 1$
        if $A[j] \leq A[r]$
            $i \leftarrow i + 1$
            swap $A[i]$ and $A[j]$
    swap $A[i+1]$ and $A[r]$
    return $i + 1$
```

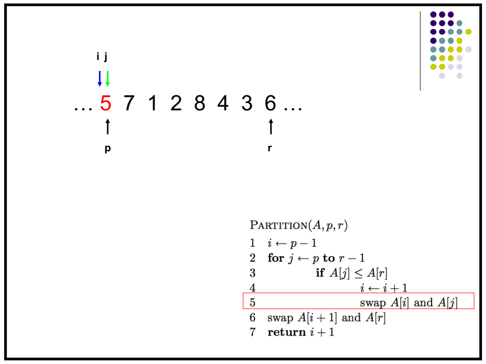
8



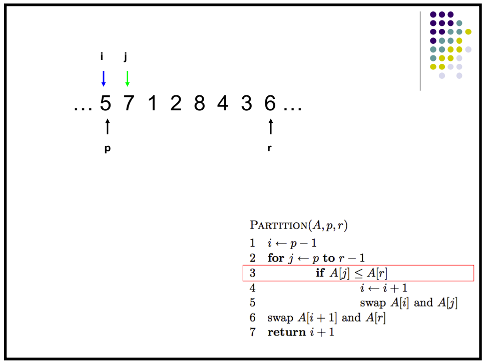
```
PARTITION$(A,p,r)$
  $i \leftarrow p-1$
  for $j \leftarrow p$ to $r-1$
        if $A[j] \le A[r]$
              $i \leftarrow i+1$
              swap $A[i]$ and $A[j]$
  swap $A[i+1]$ and $A[r]$
  return $i+1$
```



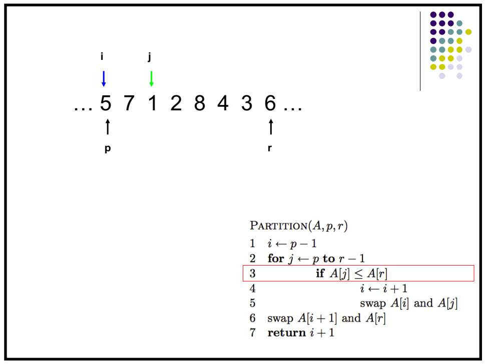
```
PARTITION$(A,p,r)$
  $i \leftarrow p-1$
  for $j \leftarrow p$ to $r-1$
        if $A[j] \le A[r]$
              $i \leftarrow i+1$
              swap $A[i]$ and $A[j]$
  swap $A[i+1]$ and $A[r]$
  return $i+1$
```



```
PARTITION$(A,p,r)$
  $i \leftarrow p-1$
  for $j \leftarrow p$ to $r-1$
        if $A[j] \le A[r]$
              $i \leftarrow i+1$
              swap $A[i]$ and $A[j]$
  swap $A[i+1]$ and $A[r]$
  return $i+1$
```

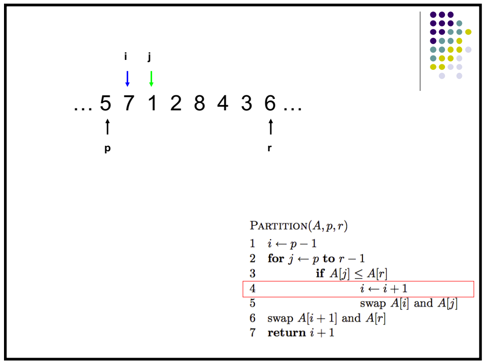


```
PARTITION$(A,p,r)$
  $i \leftarrow p-1$
  for $j \leftarrow p$ to $r-1$
        if $A[j] \le A[r]$
              $i \leftarrow i+1$
              swap $A[i]$ and $A[j]$
  swap $A[i+1]$ and $A[r]$
  return $i+1$
```



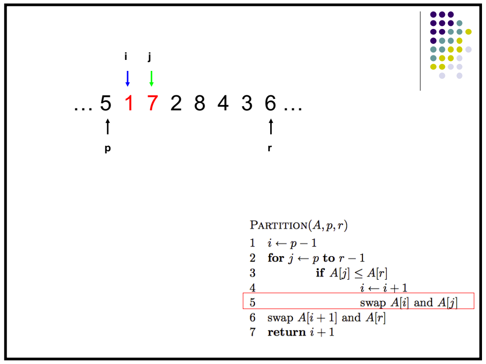
```
PARTITION$(A, p, r)$
  $i \leftarrow p - 1$
  for $j \leftarrow p$ to $r - 1$
      if $A[j] \le A[r]$
          $i \leftarrow i + 1$
          swap $A[i]$ and $A[j]$
  swap $A[i + 1]$ and $A[r]$
  return $i + 1$
```

13



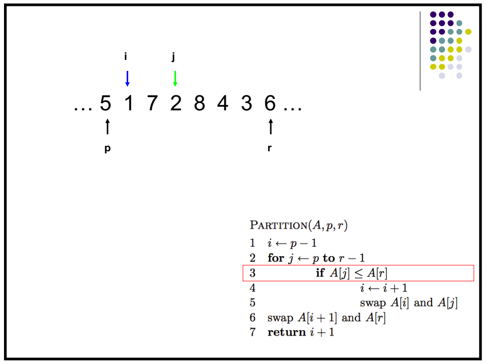
```
PARTITION$(A, p, r)$
  $i \leftarrow p - 1$
  for $j \leftarrow p$ to $r - 1$
      if $A[j] \le A[r]$
          $i \leftarrow i + 1$
          swap $A[i]$ and $A[j]$
  swap $A[i + 1]$ and $A[r]$
  return $i + 1$
```

14



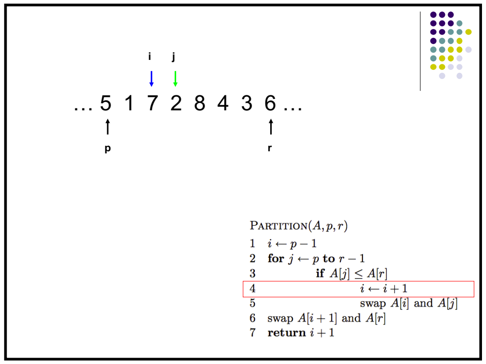
```
PARTITION$(A, p, r)$
  $i \leftarrow p - 1$
  for $j \leftarrow p$ to $r - 1$
      if $A[j] \le A[r]$
          $i \leftarrow i + 1$
          swap $A[i]$ and $A[j]$
  swap $A[i + 1]$ and $A[r]$
  return $i + 1$
```

15



```
PARTITION$(A, p, r)$
  $i \leftarrow p - 1$
  for $j \leftarrow p$ to $r - 1$
      if $A[j] \le A[r]$
          $i \leftarrow i + 1$
          swap $A[i]$ and $A[j]$
  swap $A[i + 1]$ and $A[r]$
  return $i + 1$
```

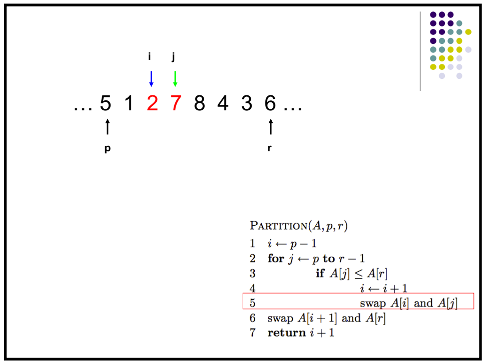
16



… 5 1 7 2 8 4 3 6 …

```
PARTITION($A, p, r$)
$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$
        if $A[j] \le A[r]$
                $i \leftarrow i + 1$
                swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i + 1$
```

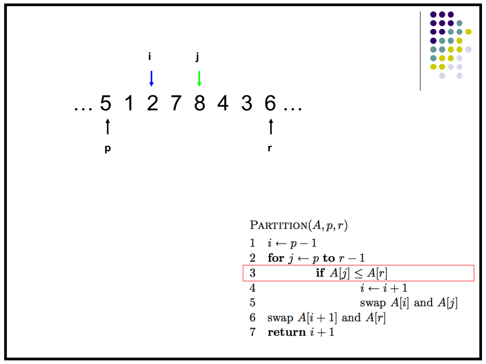
17



… 5 1 2 7 8 4 3 6 …

```
PARTITION($A, p, r$)
$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$
        if $A[j] \le A[r]$
                $i \leftarrow i + 1$
                swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i + 1$
```

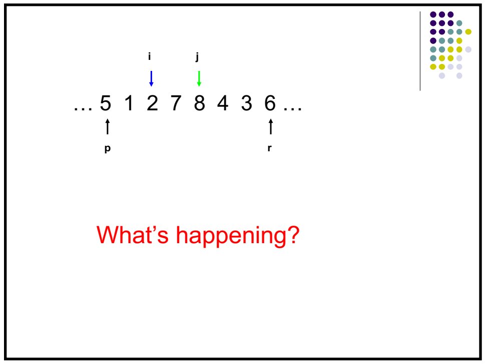
18



… 5 1 2 7 8 4 3 6 …

```
PARTITION($A, p, r$)
$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$
        if $A[j] \le A[r]$
                $i \leftarrow i + 1$
                swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i + 1$
```

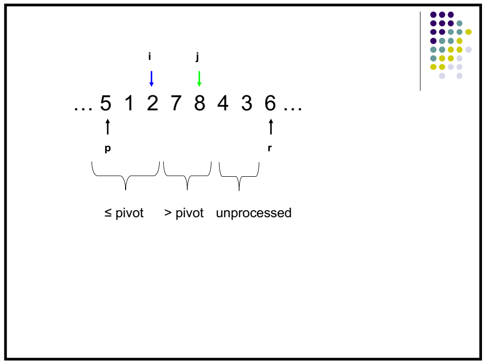
19



… 5 1 2 7 8 4 3 6 …

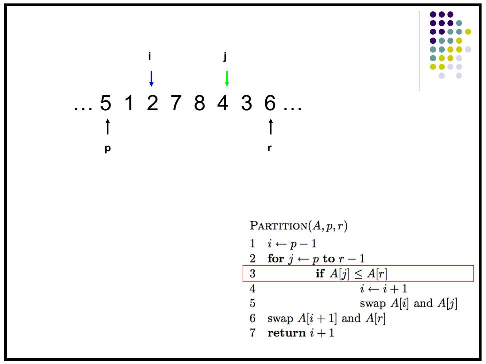
What's happening?

20



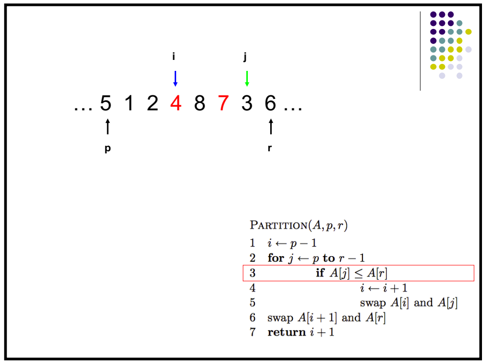
… 5 1 2 7 8 4 3 6 …

≤ pivot　> pivot　unprocessed



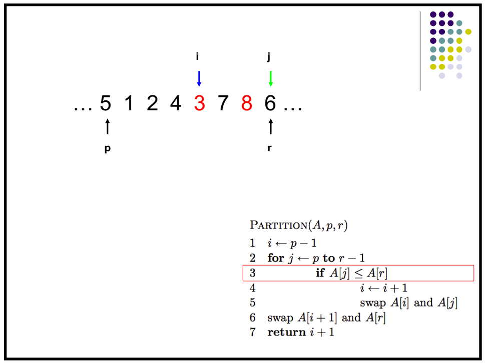
… 5 1 2 7 8 4 3 6 …

```
PARTITION($A, p, r$)
$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$
    if $A[j] \leq A[r]$
        $i \leftarrow i + 1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i + 1$
```



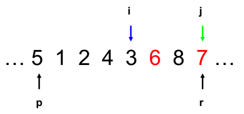
… 5 1 2 4 8 7 3 6 …

```
PARTITION($A, p, r$)
$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$
    if $A[j] \leq A[r]$
        $i \leftarrow i + 1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i + 1$
```

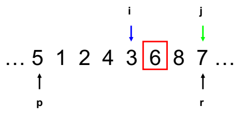


… 5 1 2 4 3 7 8 6 …

```
PARTITION($A, p, r$)
$i \leftarrow p - 1$
for $j \leftarrow p$ to $r - 1$
    if $A[j] \leq A[r]$
        $i \leftarrow i + 1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i + 1$
```



```
PARTITION($A, p, r$)
$i \leftarrow p-1$
for $j \leftarrow p$ to $r-1$
    if $A[j] \leq A[r]$
        $i \leftarrow i+1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i+1$
```



```
PARTITION($A, p, r$)
$i \leftarrow p-1$
for $j \leftarrow p$ to $r-1$
    if $A[j] \leq A[r]$
        $i \leftarrow i+1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i+1$
```

Partition running time?

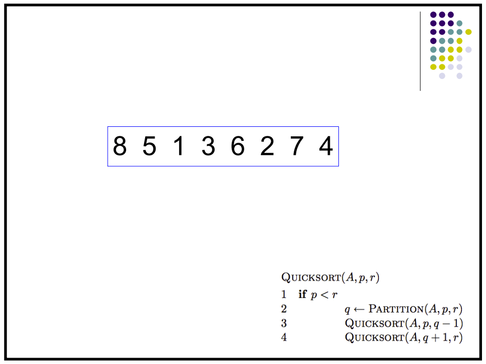
Θ(n)

```
PARTITION($A, p, r$)
$i \leftarrow p-1$
for $j \leftarrow p$ to $r-1$
    if $A[j] \leq A[r]$
        $i \leftarrow i+1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i+1$
```

Quicksort

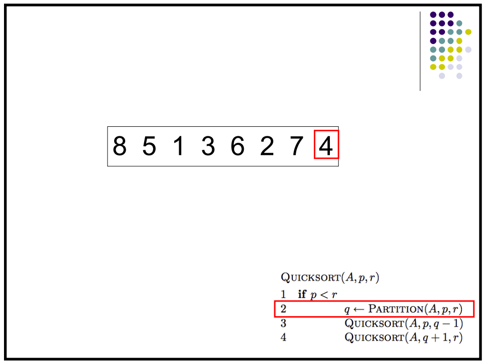
```
QUICKSORT($A, p, r$)
if $p < r$
    $q \leftarrow$ PARTITION($A, p, r$)
    QUICKSORT($A, p, q-1$)
    QUICKSORT($A, q+1, r$)
```

```
PARTITION($A, p, r$)
$i \leftarrow p-1$
for $j \leftarrow p$ to $r-1$
    if $A[j] \leq A[r]$
        $i \leftarrow i+1$
        swap $A[i]$ and $A[j]$
swap $A[i+1]$ and $A[r]$
return $i+1$
```



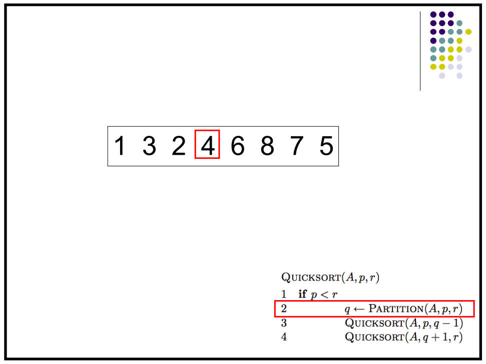
8 5 1 3 6 2 7 4

```
QUICKSORT($A, p, r$)
if $p < r$
    $q \leftarrow$ PARTITION($A, p, r$)
    QUICKSORT($A, p, q - 1$)
    QUICKSORT($A, q + 1, r$)
```



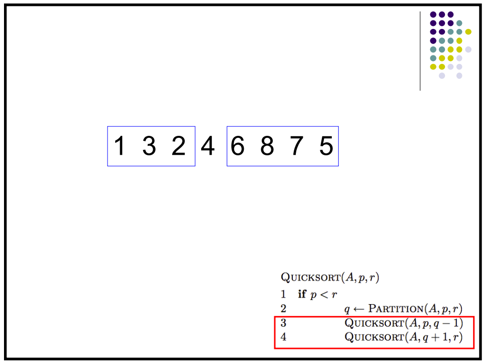
8 5 1 3 6 2 7 4

```
QUICKSORT($A, p, r$)
if $p < r$
    $q \leftarrow$ PARTITION($A, p, r$)
    QUICKSORT($A, p, q - 1$)
    QUICKSORT($A, q + 1, r$)
```



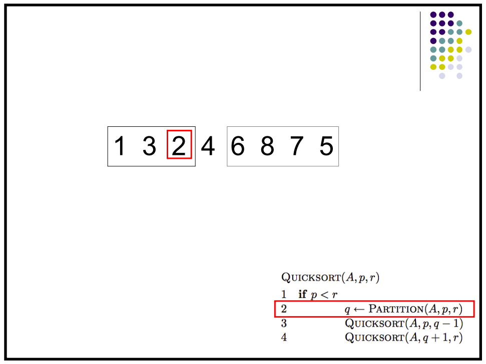
1 3 2 4 6 8 7 5

```
QUICKSORT($A, p, r$)
if $p < r$
    $q \leftarrow$ PARTITION($A, p, r$)
    QUICKSORT($A, p, q - 1$)
    QUICKSORT($A, q + 1, r$)
```



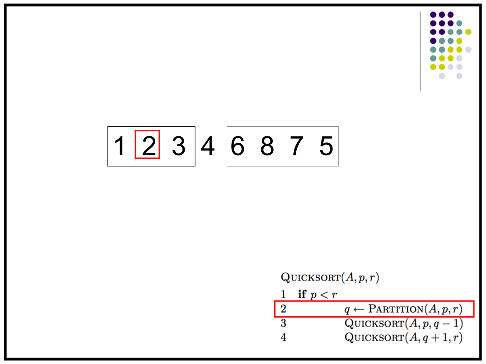
1 3 2 4 6 8 7 5

```
QUICKSORT($A, p, r$)
if $p < r$
    $q \leftarrow$ PARTITION($A, p, r$)
    QUICKSORT($A, p, q - 1$)
    QUICKSORT($A, q + 1, r$)
```



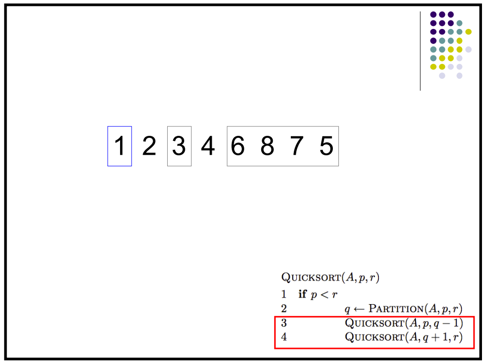
1 3 2 4 6 8 7 5

```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



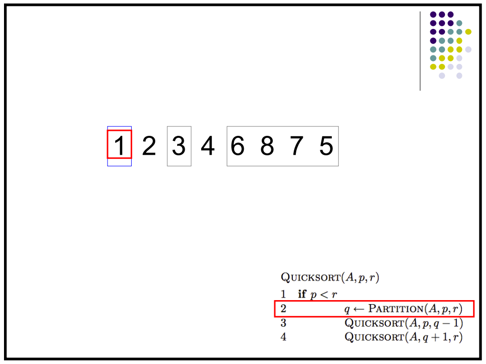
1 2 3 4 6 8 7 5

```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



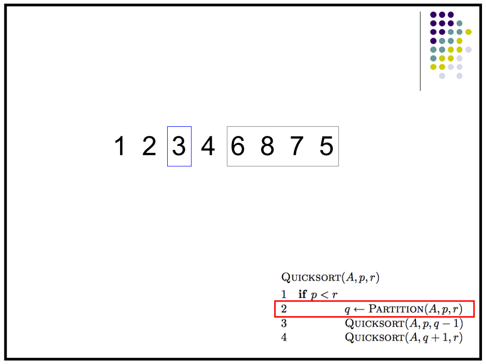
1 2 3 4 6 8 7 5

```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



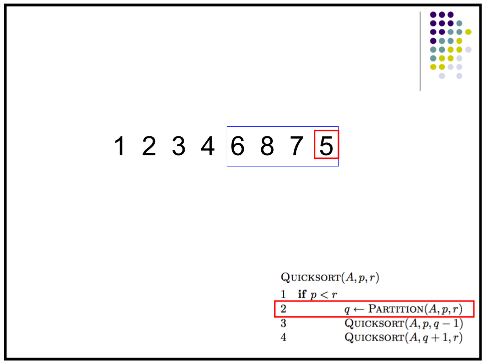
1 2 3 4 6 8 7 5

```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



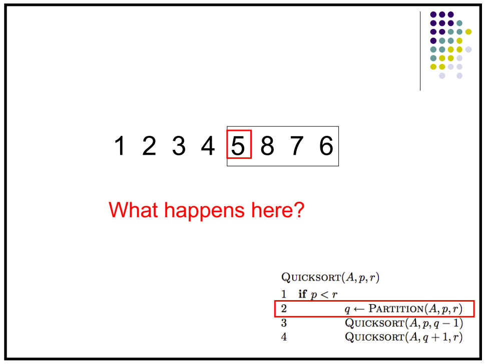
1 2 3 4 6 8 7 5

```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



1 2 3 4 6 8 7 5

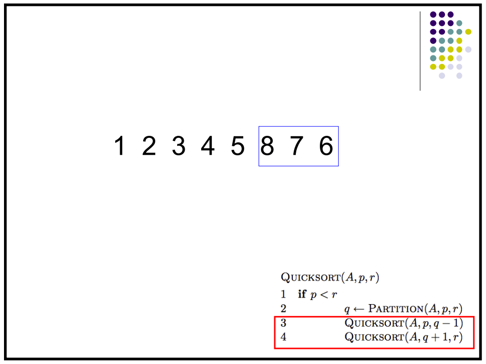
```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



1 2 3 4 5 8 7 6

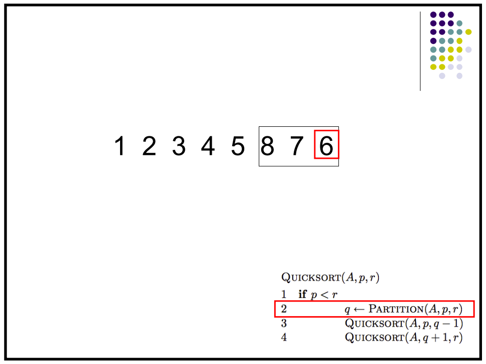
What happens here?

```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



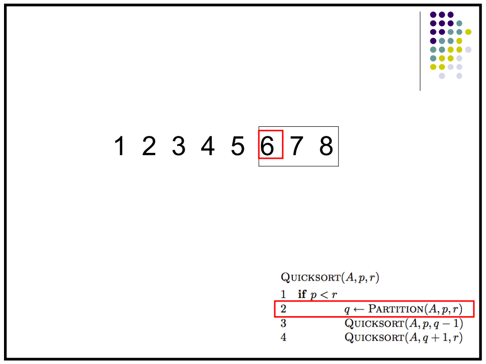
1 2 3 4 5 8 7 6

```
QUICKSORT(A, p, r)
if $p < r$
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```



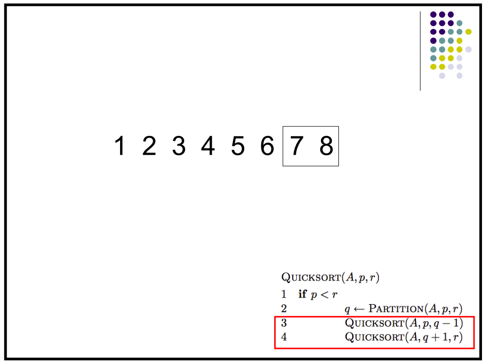
1 2 3 4 5 8 7 6

```
QUICKSORT$(A, p, r)$
    if $p < r$
        $q \leftarrow$ PARTITION$(A, p, r)$
        QUICKSORT$(A, p, q-1)$
        QUICKSORT$(A, q+1, r)$
```



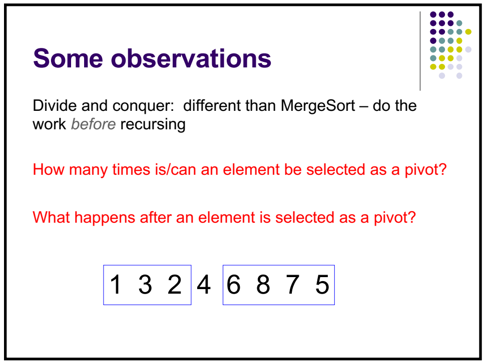
1 2 3 4 5 6 7 8

```
QUICKSORT$(A, p, r)$
    if $p < r$
        $q \leftarrow$ PARTITION$(A, p, r)$
        QUICKSORT$(A, p, q-1)$
        QUICKSORT$(A, q+1, r)$
```



1 2 3 4 5 6 7 8

```
QUICKSORT$(A, p, r)$
    if $p < r$
        $q \leftarrow$ PARTITION$(A, p, r)$
        QUICKSORT$(A, p, q-1)$
        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 be selected as a pivot?

What happens after an element is selected as a pivot?

1 3 2 4 6 8 7 5

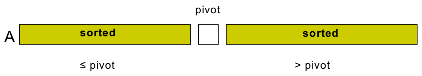
Is Quicksort correct?

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:



Running time of Quicksort?

Worst case?

Each call to Partition splits the array into an empty array and n-1 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

$$T(n) = 2T(n/2) + \Theta(n)$$

Θ(n log n)

When does this happen?

- random data?

Quicksort Average case?

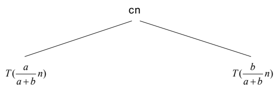
How close to “even” splits do they need to be to maintain an Θ(n log n) running time?

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

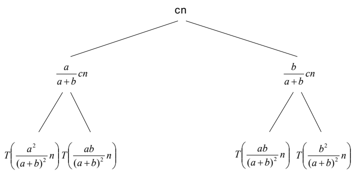
What is the recurrence?

$$T(n) \le T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn$$

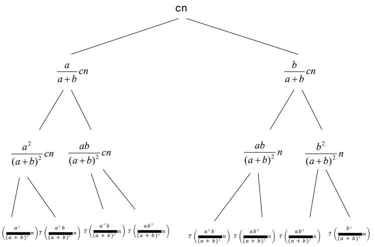
$$T(n) \le T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn$$



$$T(n) \le T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn$$



$$T(n) \le T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn$$



$$T(n) \le T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn$$

Level 0: cn

Level 1: $= cn\left(\frac{a}{a+b}\right) + cn\left(\frac{b}{a+b}\right) = cn$

Level 2: $= cn\left(\frac{a^2}{(a+b)^2}\right) + cn\left(\frac{ab}{(a+b)^2}\right) + cn\left(\frac{ab}{(a+b)^2}\right) + cn\left(\frac{b^2}{(a+b)^2}\right)$

$= cn\left(\frac{a^2+2ab+b^2}{(a+b)^2}\right) = cn\left(\frac{(a+b)^2}{(a+b)^2}\right) = cn$

Level 3: $= cn\left(\frac{(a+b)^2 a + (a+b)^2 b}{(a+b)^3}\right)$

$= cn\left(\frac{(a+b)(a+b)^2}{(a+b)^3}\right) = cn$

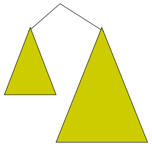
Level d: $= cn\left(\frac{(a+b)^d}{(a+b)^d}\right) = cn$

What is the depth of the tree?

Leaves will have different heights

Want to pick the deepest leaf

Assume a < b



What is the depth of the tree?

Assume a < b

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

$$\ldots$$

$$d = \log_{\frac{a+b}{b}} n$$

Cost of the tree

Cost of each level ≤ cn

?

Cost of the tree

Cost of each level ≤ cn

Times the maximum depth

$$O(n\log_{\frac{a+b}{b}} n)$$

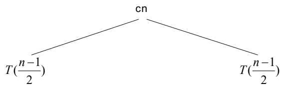
Why not?

$$\Theta(n\log_{\frac{a+b}{b}} n)$$

Quicksort average case: take 2

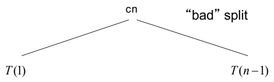
What would happen if half the time Partition produced a "bad" split and the other half "good"?

"good" 50/50 split



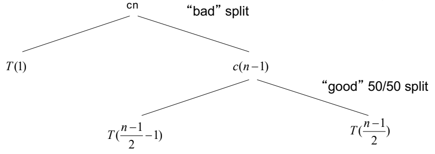
$$T(n) = 2T(\frac{n-1}{2}) + \Theta(n)$$

Quicksort average case: take 2



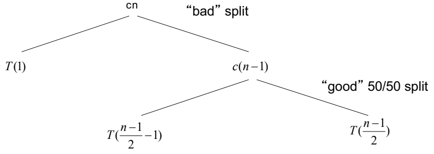
"bad" split

Quicksort average case: take 2



$$T(n) = T(1) + \overbrace{T(\frac{n-1}{2}-1) + T(\frac{n-1}{2})}^{\text{recursion cost}} + \overbrace{\Theta(n) + \Theta(n-1)}^{\text{partition cost}}$$

Quicksort average case: take 2



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

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($A, p, r$)
  $i \leftarrow$ RANDOM($p, r$)
  swap $A[r]$ and $A[i]$
  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 θ(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] < A[j]
- A[i] > A[j]
- A[i] = A[j]
- A[i] ≤ A[j]
- A[i] ≥ 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

Comparison-based sorting

Sorted order is determined based **only** on a comparison between input elements

- A[i] < A[j]
- A[i] > A[j]
- A[i] = A[j]
- A[i] ≤ A[j]
- A[i] ≥ A[j]

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

What does it do?

Comparison-based sorting

Sorted order is determined based **only** on a comparison between input elements

- A[i] < A[j]
- A[i] > A[j]
- A[i] = A[j]
- A[i] ≤ A[j]
- A[i] ≥ A[j]

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

Comparison-based sorting

Sorted order is determined based **only** on a comparison between input elements

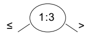
- A[i] < A[j]
- A[i] > A[j]
- A[i] = A[j]
- A[i] ≤ A[j]
- A[i] ≥ A[j]

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

| 1,3,2 |

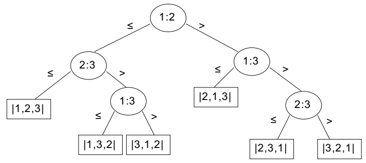
[3, 12, 7] ⇨ | 1,3,2 | ⇨ [3, 7, 12]

[7, 3, 12] ⇨ | 2,1,3 | ⇨ [3, 7, 12]

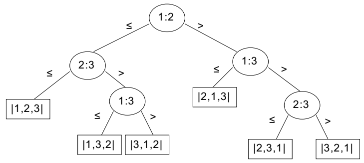
Tracing a path from root to leave gives the correct reordering/permutation of the input for an input

---

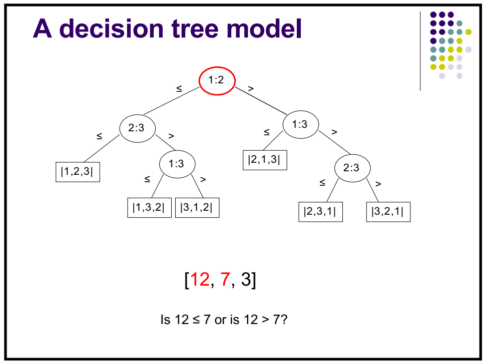
A decision tree model



A decision tree model



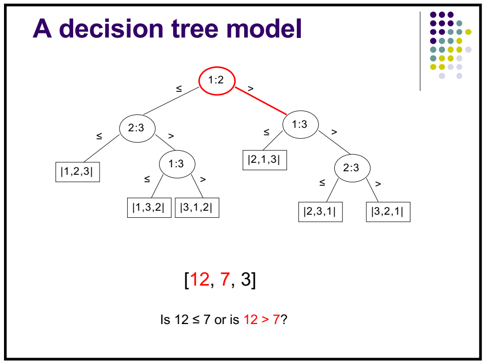
[12, 7, 3]



A decision tree model

[12, 7, 3]

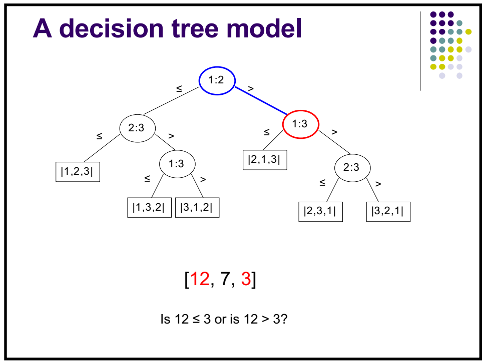
Is 12 ≤ 7 or is 12 > 7?



A decision tree model

[12, 7, 3]

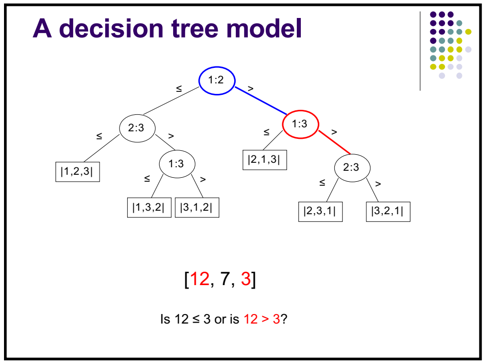
Is 12 ≤ 7 or is 12 > 7?



A decision tree model

[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

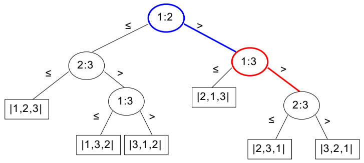


A decision tree model

[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

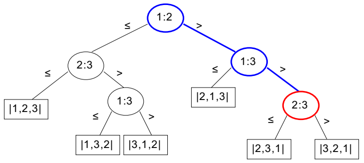
A decision tree model



[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

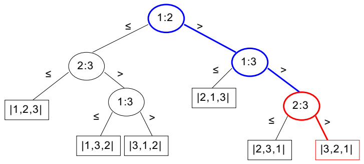
A decision tree model



[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

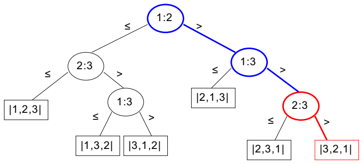
A decision tree model



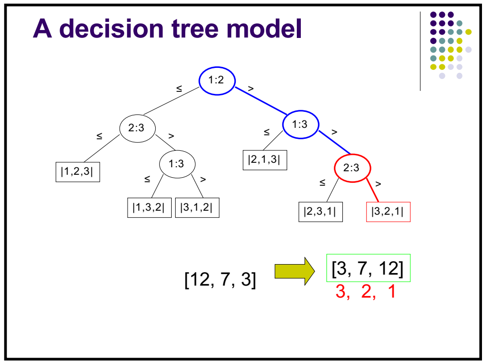
[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

A decision tree model



[12, 7, 3] ⇨ 3, 2, 1

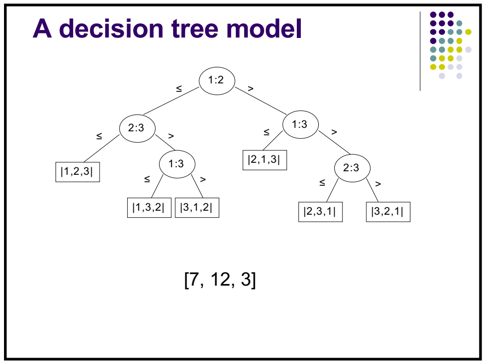


A decision tree model

[12, 7, 3]

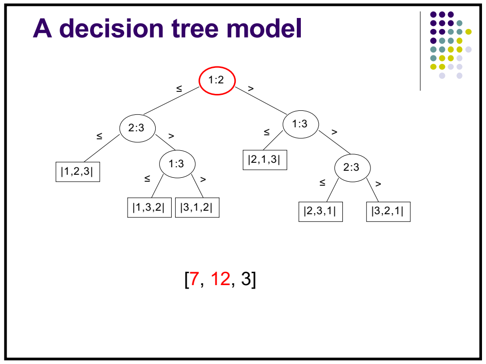
[3, 7, 12]

3, 2, 1



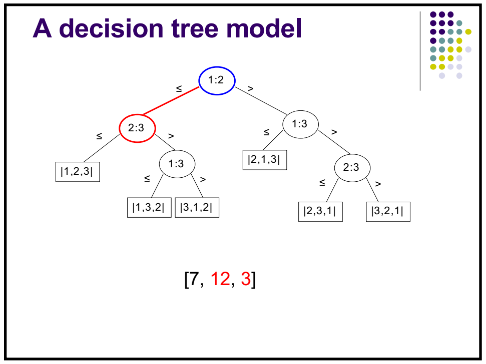
A decision tree model

[7, 12, 3]



A decision tree model

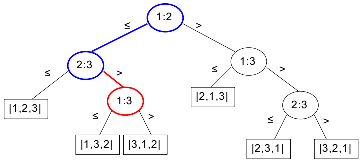
[7, 12, 3]



A decision tree model

[7, 12, 3]

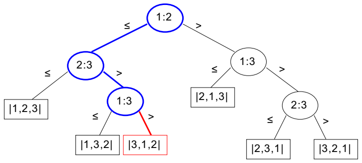
A decision tree model



[7, 12, 3]

97

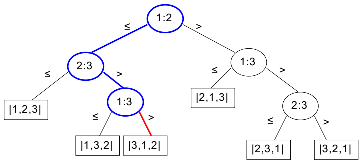
A decision tree model



[7, 12, 3]

98

A decision tree model



[7, 12, 3] ⇨ [3, 7, 12]

3, 1, 2

99

How many leaves are in a decision tree?

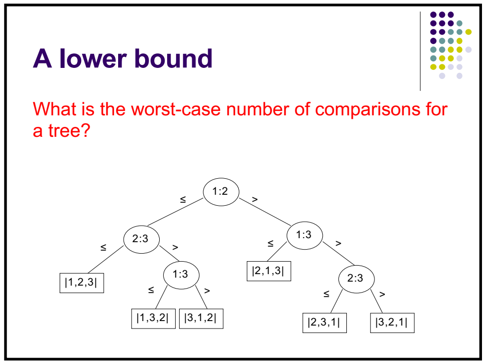
Leaves **must** have all possible permutations of the input

What if decision tree model didn't?

Some input would exist that didn't have a correct reordering

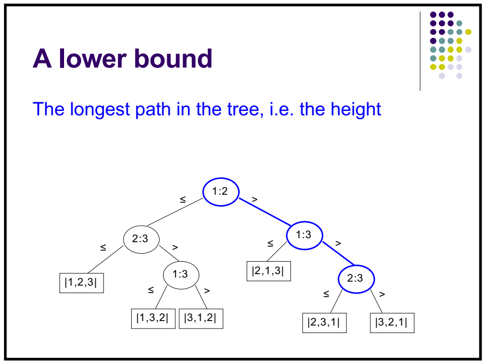
Input of size *n*, *n!* leaves

100



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

A lower bound

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?