Sorting Concluded

David Kauchak
CS140
Fall 2022

1

Administrative

- Homework 3 out

2

```
PARTITION(A, p, r)
  i ← p − 1
  for j ← p to r − 1
        if A[j] ≤ A[r]
              i ← i + 1
              swap A[i] and A[j]
  swap A[i + 1] and A[r]
  return i + 1
```

What does it do?

3

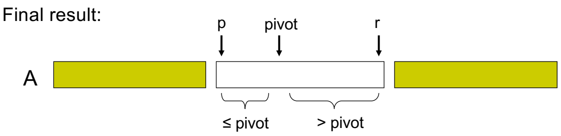
A[r] is called the ***pivot***

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

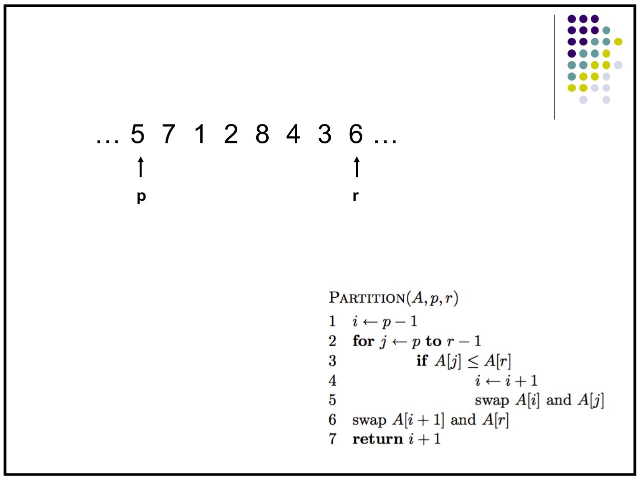
Operates in place

Final result:

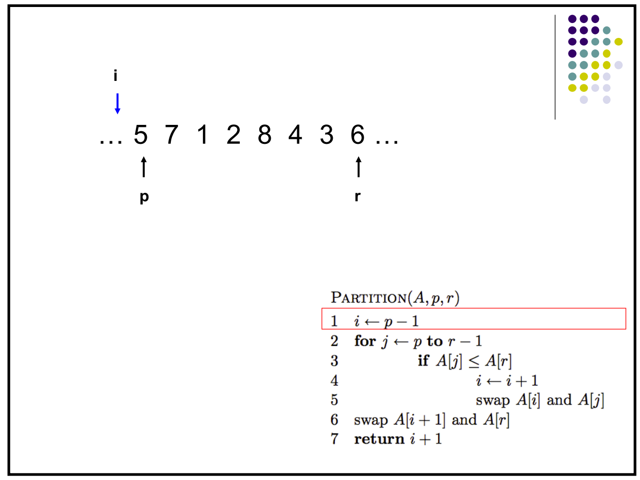
```
PARTITION(A, p, r)
  i ← p − 1
  for j ← p to r − 1
        if A[j] ≤ A[r]
              i ← i + 1
              swap A[i] and A[j]
  swap A[i + 1] and A[r]
  return i + 1
```



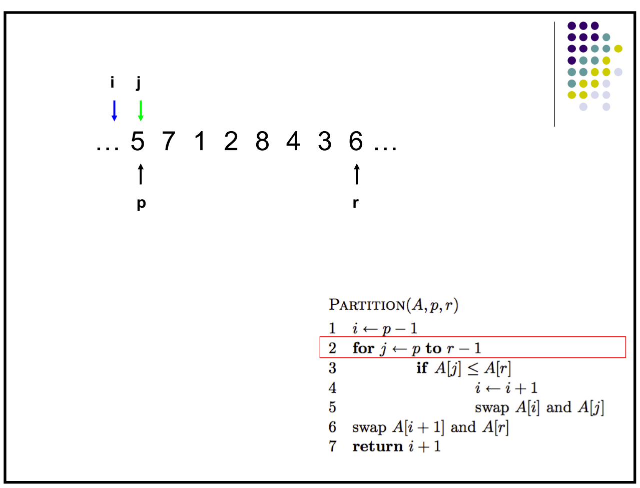
4


… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

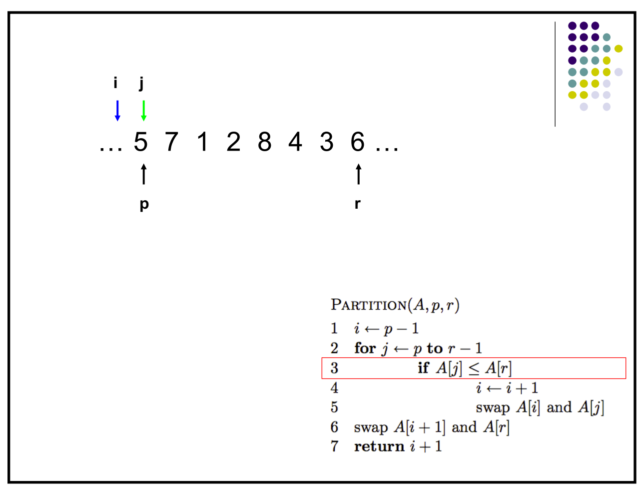
5


i
… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

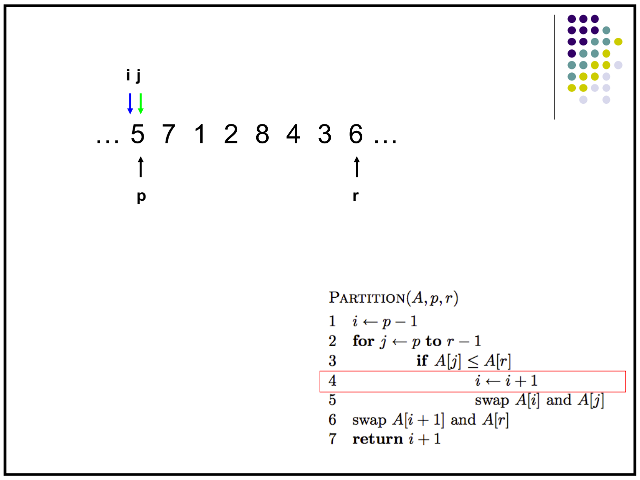
6


i
j
… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

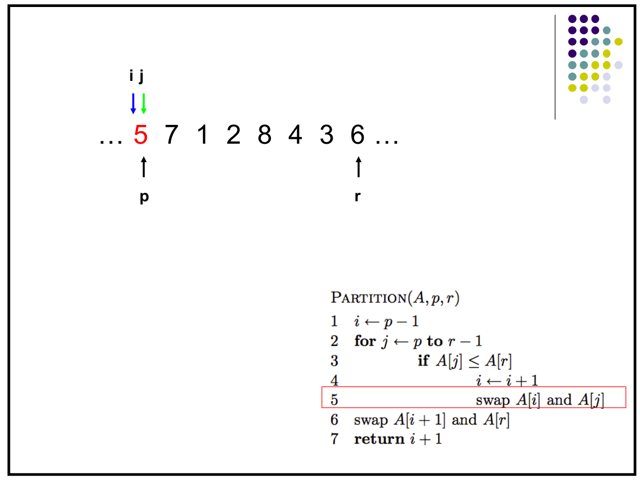
7


i
j
… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

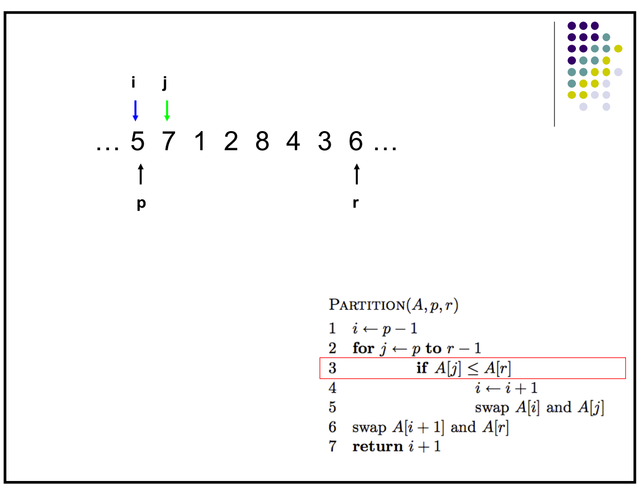
8


i j
… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

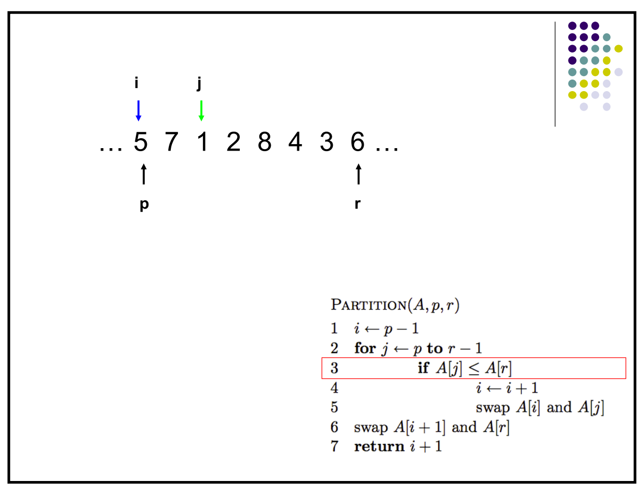
9


i j
… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

10


i
j
… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

11


i
j
… 5 7 1 2 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

12

i j

… 5 7 1 2 8 4 3 6 …

p r

```
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

13

i j

… 5 1 7 2 8 4 3 6 …

p r

```
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

14

i j

… 5 1 7 2 8 4 3 6 …

p r

```
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

15

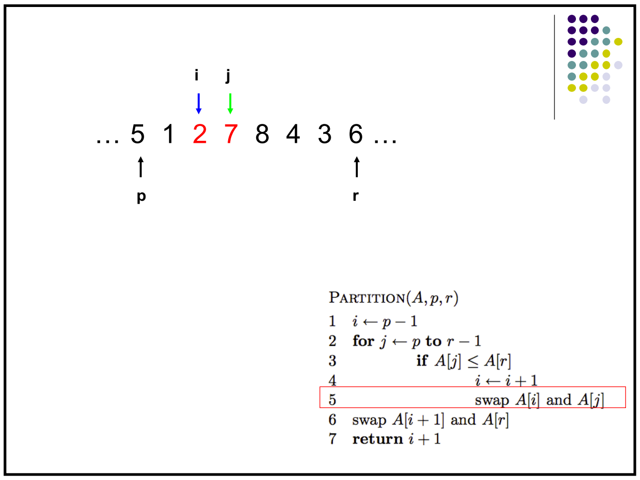
i j

… 5 1 7 2 8 4 3 6 …

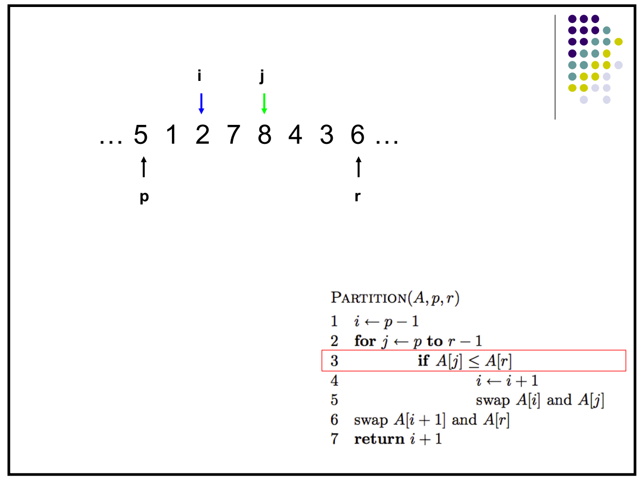
p r

```
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

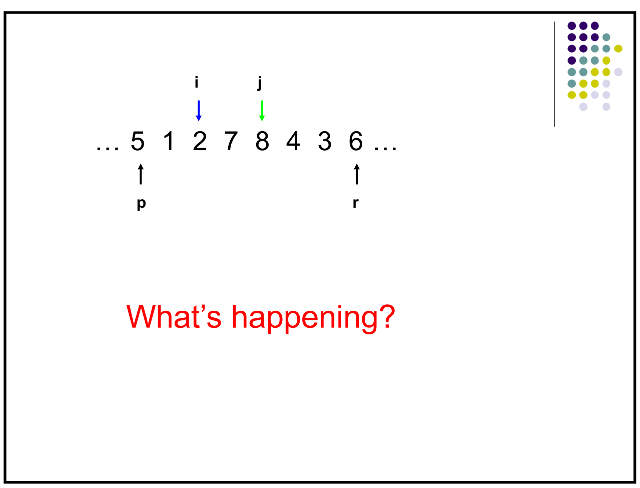
16


i
j
… 5 1 2 7 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

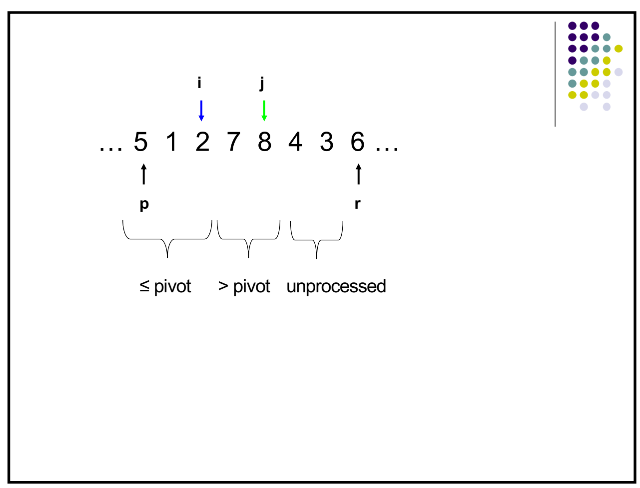
17


i
j
… 5 1 2 7 8 4 3 6 …
p
r
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
if A[j] ≤ A[r]
i ← i + 1
swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1

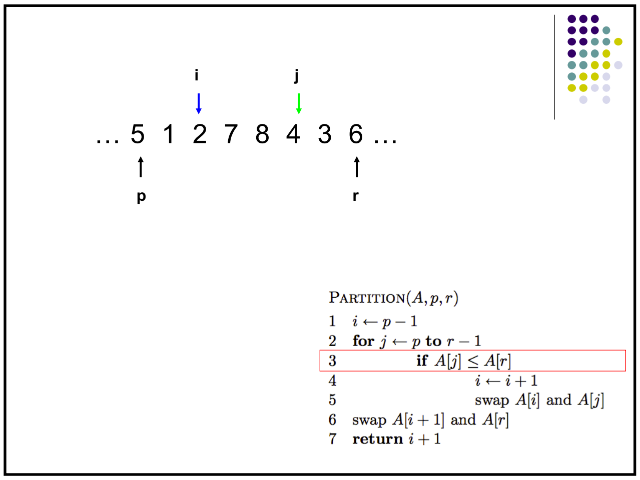
18


i
j
… 5 1 2 7 8 4 3 6 …
p
r
What's happening?

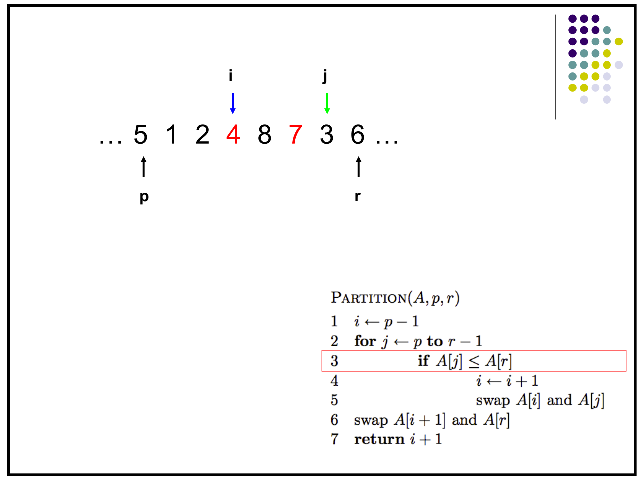
19


i
j
… 5 1 2 7 8 4 3 6 …
p
r
≤ pivot
> pivot
unprocessed

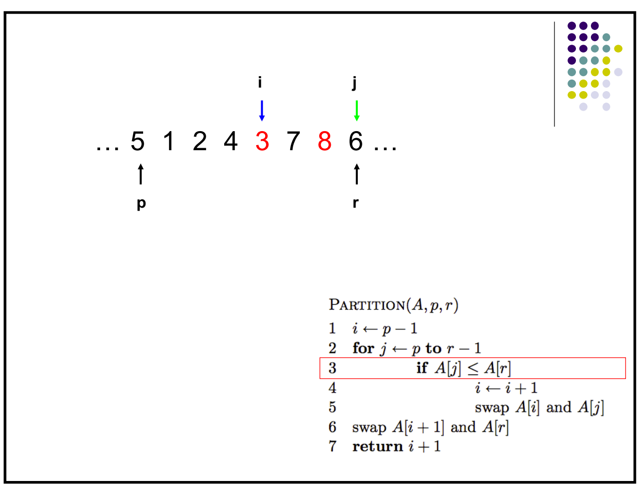
20


i
j
… 5 1 2 7 8 4 3 6 …
p
r
PARTITION(A, p, r)
1 i ← p − 1
2 for j ← p to r − 1
3 if A[j] ≤ 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

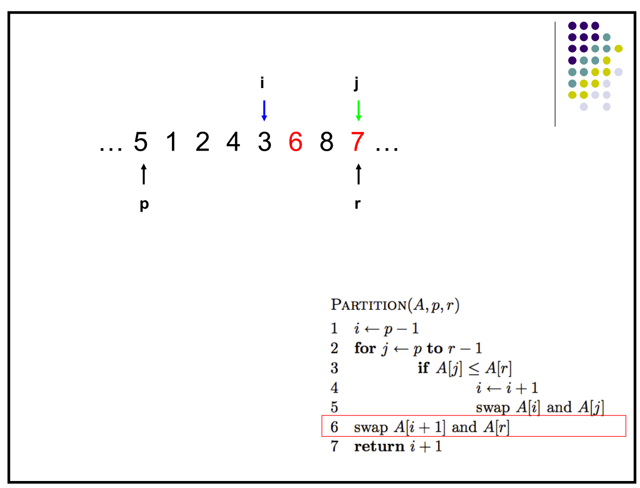
21


i
j
… 5 1 2 4 8 7 3 6 …
p
r
PARTITION(A, p, r)
1 i ← p − 1
2 for j ← p to r − 1
3 if A[j] ≤ 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

22


i
j
… 5 1 2 4 3 7 8 6 …
p
r
PARTITION(A, p, r)
1 i ← p − 1
2 for j ← p to r − 1
3 if A[j] ≤ 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

23


i
j
… 5 1 2 4 3 6 8 7 …
p
r
PARTITION(A, p, r)
1 i ← p − 1
2 for j ← p to r − 1
3 if A[j] ≤ 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

24

i j

… 5 1 2 4 3 6 8 7 …

p r

$\text{PARTITION}(A, p, r)$

```
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

25

Is Partition correct?

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

Loop Invariant:

$\text{PARTITION}(A, p, r)$

```
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

26

Is Partition correct?

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

Loop Invariant:

A[p…i] ≤ A[r] and A[i+1…j-1] > A[r]

proof?

$\text{PARTITION}(A, p, r)$

```
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

27

Proof by induction

Loop Invariant: A[p…i] ≤ A[r] 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[j] > A[r]
  - A[p…i] remains unchanged
  - A[i+1…j] contains one additional element, A[j] which is > A[r]

$\text{PARTITION}(A, p, r)$

```
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

28

Proof by induction

Loop Invariant: A[p…i] ≤ A[r] and A[i+1…j-1] > A[r]

2nd case:

- A[j] ≤ A[r]
  - *i* is incremented
  - A[i] swapped with A[j] – A[p…i] constains one additional element which is ≤ A[r]
  - A[i+1…j-1] will contain the same elements, except the last element will be the old first element

```
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

29

Partition running time?

Θ(n)

```
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

30

Quicksort

```
QUICKSORT(A, p, r)
if p < r
        q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

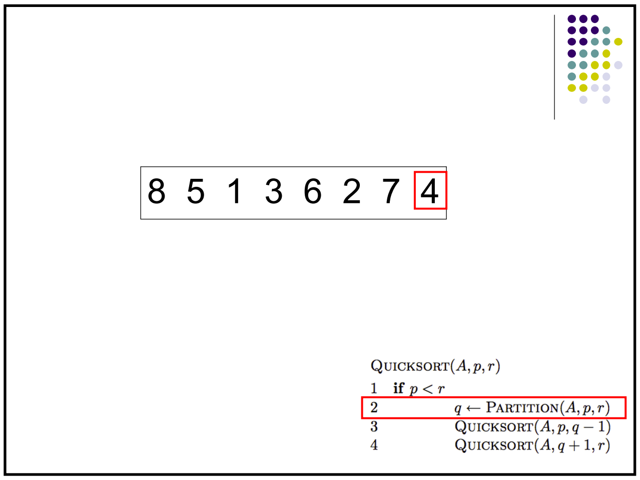
```
PARTITION(A, p, r)
i ← p − 1
for j ← p to r − 1
        if A[j] ≤ A[r]
                i ← i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

31

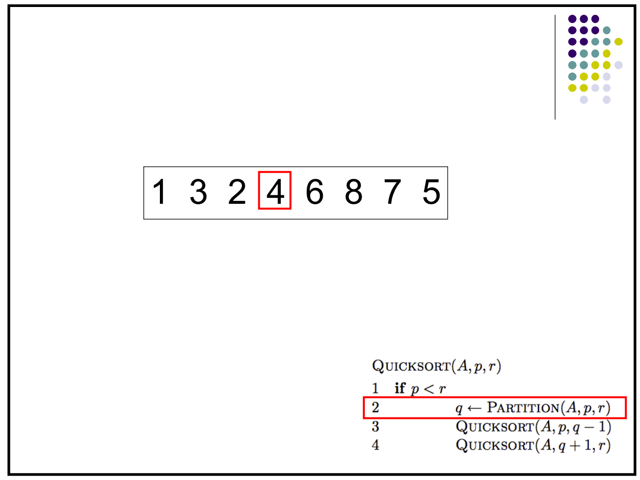
8	5	1	3	6	2	7	4

```
QUICKSORT(A, p, r)
if p < r
        q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

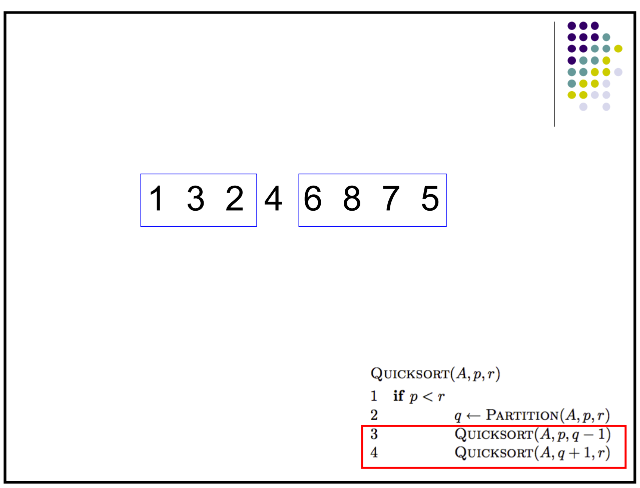
32


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$(A, q+1, r)$

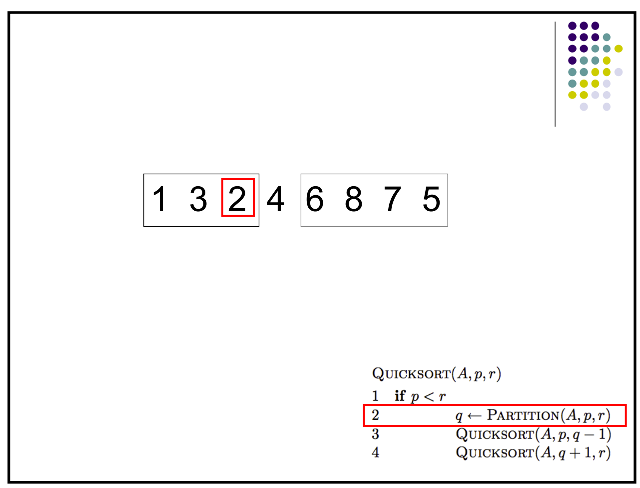
33


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$(A, q+1, r)$

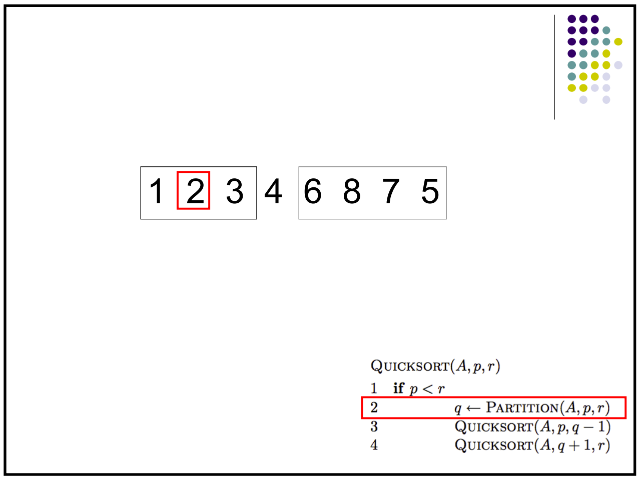
34


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$(A, q+1, r)$

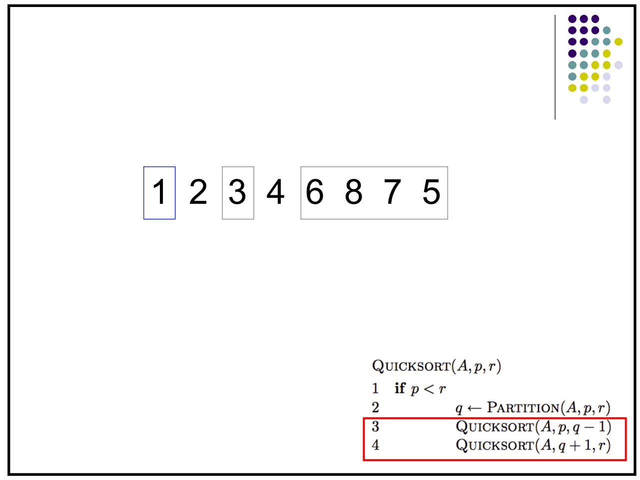
35


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$(A, q+1, r)$

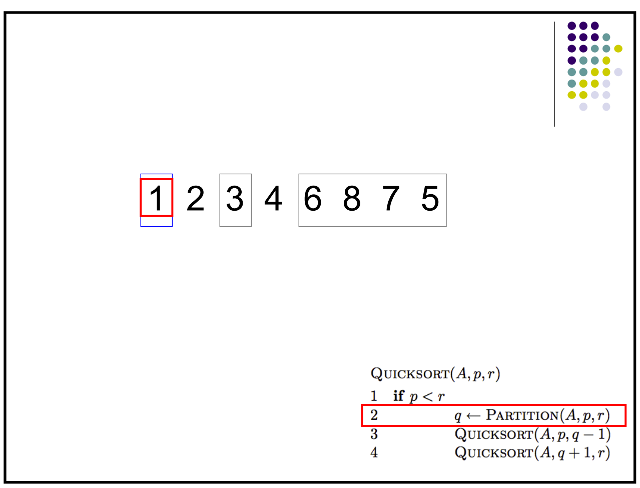
36


1 2 3 4 6 8 7 5
QUICKSORT(A, p, r)
if p < r
q ← PARTITION(A, p, r)
QUICKSORT(A, p, q − 1)
QUICKSORT(A, q + 1, r)

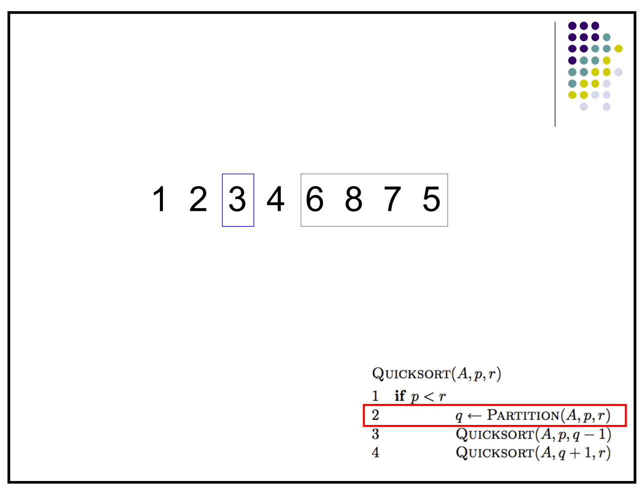
37


1 2 3 4 6 8 7 5
QUICKSORT(A, p, r)
if p < r
q ← PARTITION(A, p, r)
QUICKSORT(A, p, q − 1)
QUICKSORT(A, q + 1, r)

38


1 2 3 4 6 8 7 5
QUICKSORT(A, p, r)
if p < r
q ← PARTITION(A, p, r)
QUICKSORT(A, p, q − 1)
QUICKSORT(A, q + 1, r)

39


1 2 3 4 6 8 7 5
QUICKSORT(A, p, r)
if p < r
q ← PARTITION(A, p, r)
QUICKSORT(A, p, q − 1)
QUICKSORT(A, q + 1, r)

40

1 2 3 4 6 8 7 5

```
QUICKSORT(A, p, r)
if p < r
    q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

41

1 2 3 4 5 8 7 6

What happens here?

```
QUICKSORT(A, p, r)
if p < r
    q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

42

1 2 3 4 5 8 7 6

```
QUICKSORT(A, p, r)
if p < r
    q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

43

1 2 3 4 5 8 7 6

```
QUICKSORT(A, p, r)
if p < r
    q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

44

1 2 3 4 5 6 7 8

```
QUICKSORT(A, p, r)
if p < r
        q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

45

1 2 3 4 5 6 7 8

```
QUICKSORT(A, p, r)
if p < r
        q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

46

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?

1 3 2 4 6 8 7 5

47

Is Quicksort correct?

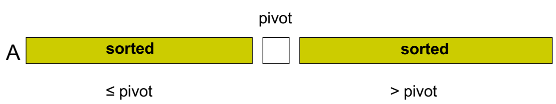
48

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:

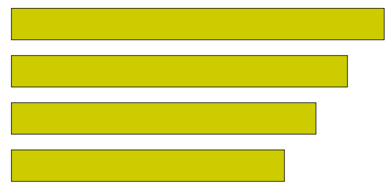


49

Running time of Quicksort?

Worst case?

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



50

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

51

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?

52

Quicksort Average case?

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

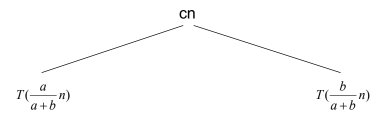
Say the Partition procedure always splits the array into some constant ratio b-to-a, e.g. 9-to-1

What is the recurrence?

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

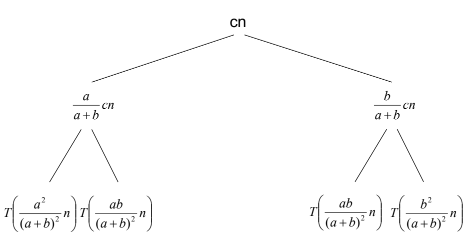
53

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



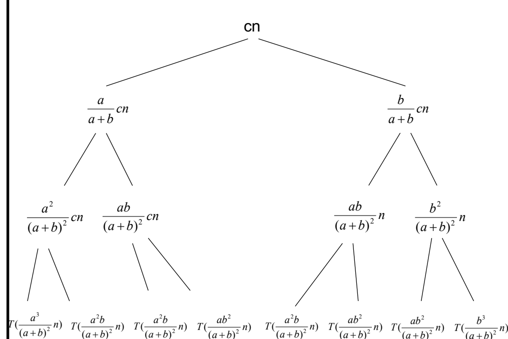
54

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



55

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



56

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

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

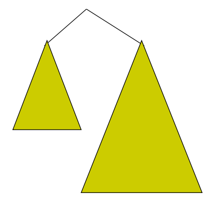
57

What is the depth of the tree?

Leaves will have different heights

Want to pick the deepest leave

Assume a < b



58

What is the depth of the tree?

Assume a < b

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

$$\ldots$$

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

59

Cost of the tree

Cost of each level ≤ cn

?

60

Cost of the tree

Cost of each level ≤ cn

Times the maximum depth

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

Why not?

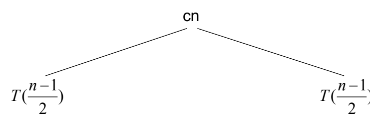
$$\Theta(n \log_{\frac{a+b}{b}} n)$$

61

Quicksort average case: take 2

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

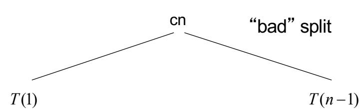
"good" 50/50 split



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

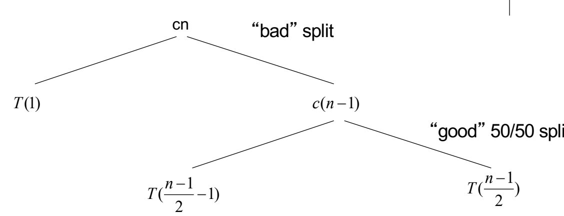
62

Quicksort average case: take 2



63

Quicksort average case: take 2



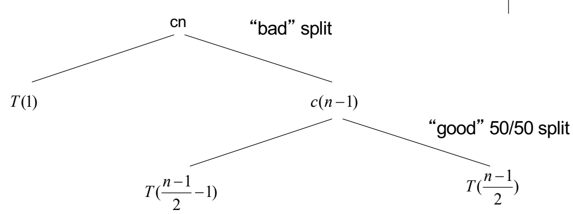
recursion cost

partition cost

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

64

Quicksort average case: take 2



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

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

65

How can we avoid the worst case?

Inject randomness into the data

RANDOMIZED-PARTITION(A, p, r)

1 $i \leftarrow$ RANDOM(p, r)
2 swap $A[r]$ and $A[i]$
3 **return** PARTITION(A, p, r)

66

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

67

Sorting bounds

Mergsort is θ(n log n)

Quicksort is O(n log n) on average

Can we do better?

76

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

77

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?

78

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

79

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]

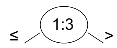
Can we do better than O(n log n) for comparison based sorting approaches?

80

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 |

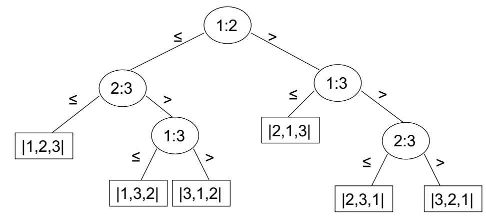
[3, 12, 7] ⇨ | 1,3,2 | ⇨ [3, 7, 12]

[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

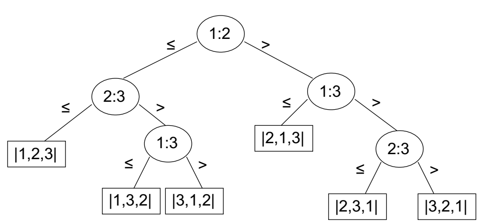
81

A decision tree model



82

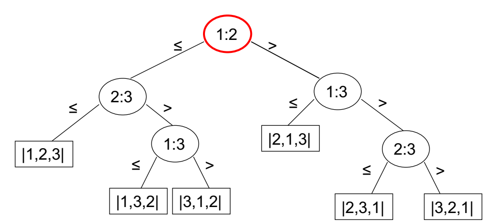
A decision tree model



[12, 7, 3]

83

A decision tree model

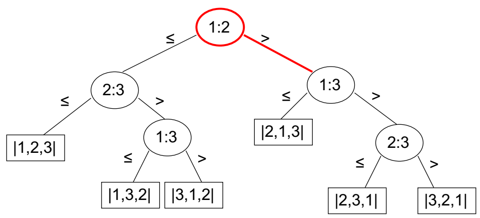


[12, 7, 3]

Is 12 ≤ 7 or is 12 > 7?

84

A decision tree model

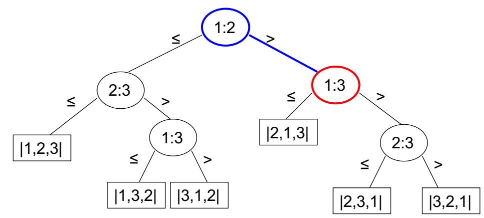


[12, 7, 3]

Is 12 ≤ 7 or is 12 > 7?

85

A decision tree model

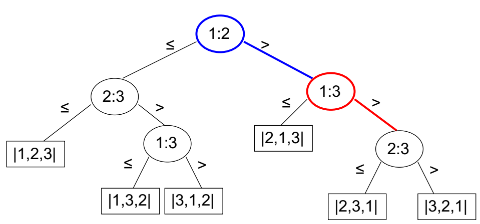


[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

86

A decision tree model

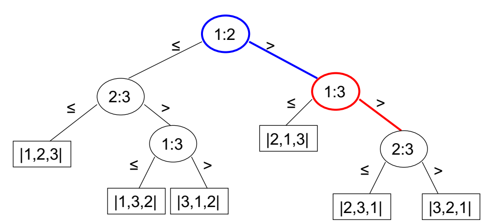


[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

87

A decision tree model

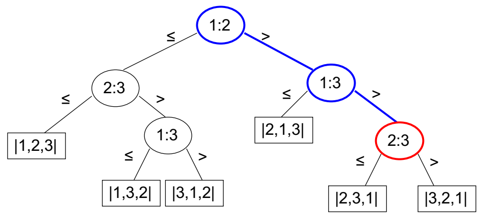


[12, 7, 3]

Is 12 ≤ 3 or is 12 > 3?

88

A decision tree model

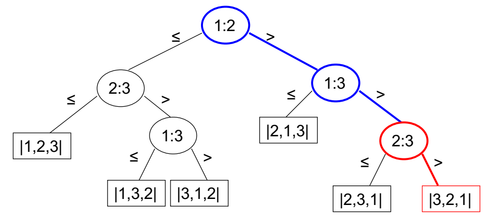


[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

89

A decision tree model

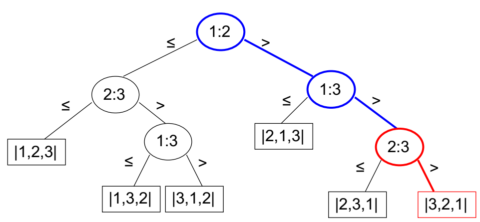


[12, 7, 3]

Is 7 ≤ 3 or is 7 > 3?

90

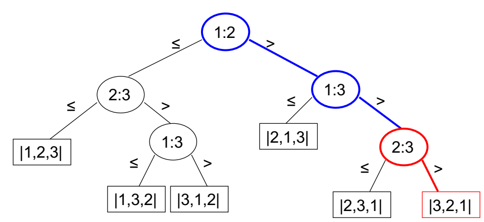
A decision tree model



[12, 7, 3] ⇒ 3, 2, 1

91

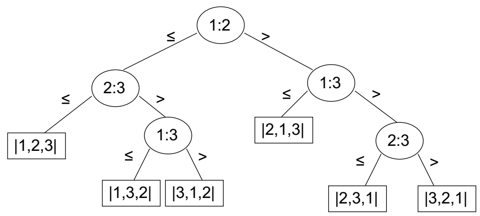
A decision tree model



[12, 7, 3] ⇒ [3, 7, 12]
3, 2, 1

92

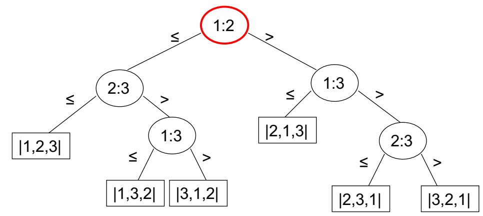
A decision tree model



[7, 12, 3]

93

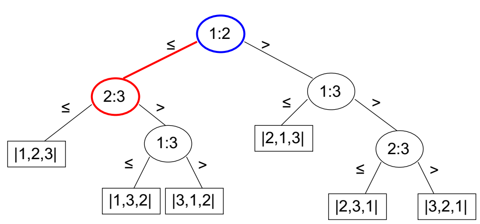
A decision tree model



[7, 12, 3]

94

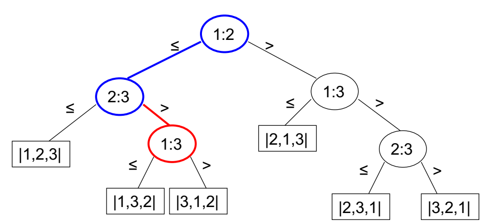
A decision tree model



[7, 12, 3]

95

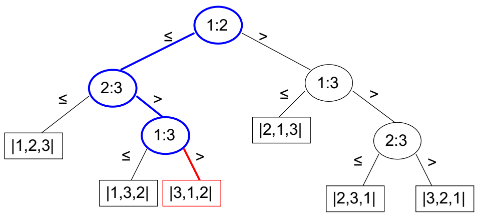
A decision tree model



[7, 12, 3]

96

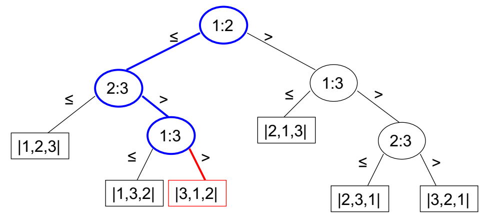
A decision tree model



[7, 12, 3]

97

A decision tree model



[7, 12, 3] ⇒ [3, 7, 12]

3, 1, 2

98

How many leaves are in a decision tree?

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

What if decision tree model didn't?

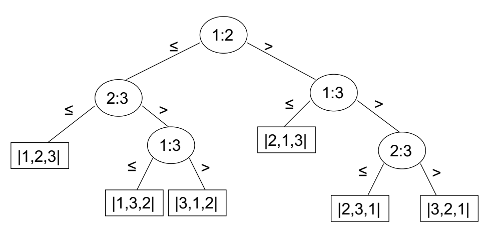
Some input would exist that didn't have a correct reordering

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

99

A lower bound

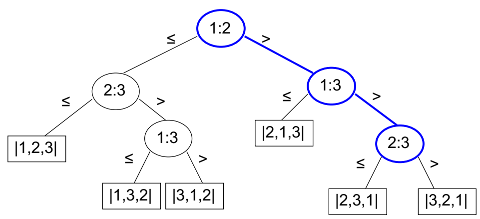
What is the worst-case number of comparisons for a tree?



100

A lower bound

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



101

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! ☺

102

Can we do better?

103