QuickSort

Pontus Ekberg

Uppsala University

(Based on previous material by Mohamed Faouzi Atig and Parosh Aziz Abdulla)

Sorting Algorithms

- Problem: Sort an array A of n elements in non-decreasing order

Algorithm	Worst-Case	Average-Case	Best-Case	In place?
Insertion Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Merge Sort	$\Theta(nlog(n))$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	No

Observe that an algorithm sorts in place if it rearranges the numbers within the array A, with at most a constant number of them sorted outside the array at any time.

Sorting Algorithms

- Problem: Sort an array A of n elements in non-decreasing order

Algorithm	Worst-Case	Average-Case	Best-Case	In place?
Insertion Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$	Yes
Merge Sort	$\Theta(nlog(n))$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	No
Quicksort	$\Theta(n^2)$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	Yes

Observe that an algorithm sorts in place if it rearranges the numbers within the array A, with at most a constant number of them sorted outside the array at any time.

Quicksort

- Developed by Tony Hoare in 1960.
- Top 10 algorithms of the 20th Century.
- A famous example of "Divide-and-Conquer" method
- Sorting in place algorithm.

Quicksort

- Divide: Partition the input array A[p..r] into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that:
  - $A[k] \leq A[q]$ for each $k : p \leq k \leq q-1$
  - $A[k] > A[q]$ for each $k : q+1 \leq k \leq r$

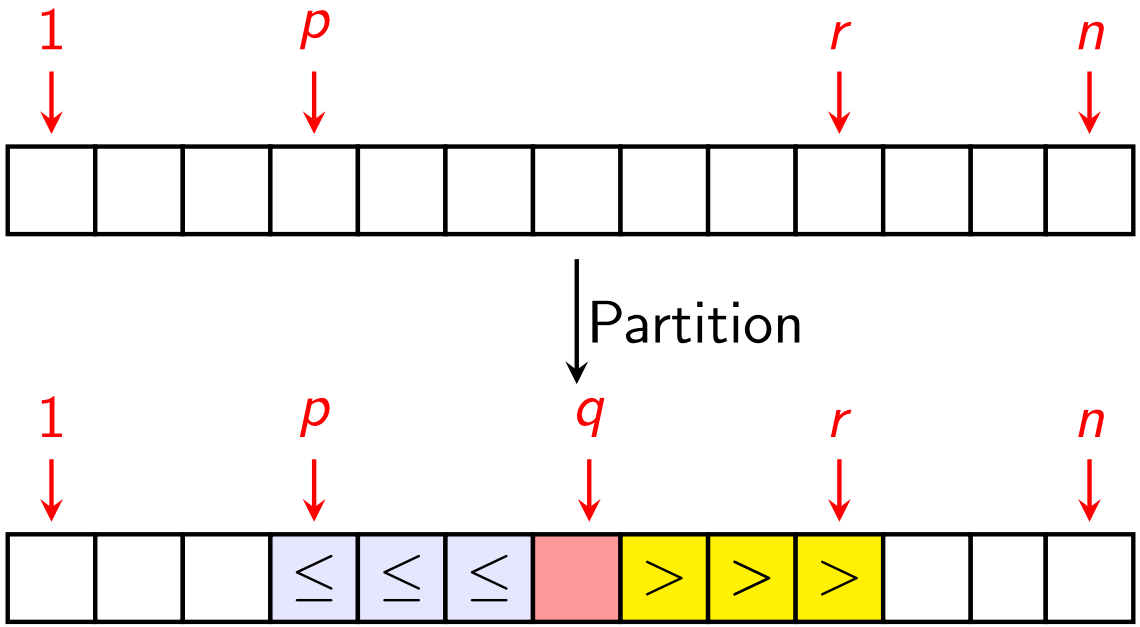
  $A[q]$ is called the pivot element.
- Conquer: Sort each of the two subarrays $A[p..q-1]$ and $A[q+1..r]$ recursively.
- Combine: Trivial.
- Base Cases: When $p \geq r$

Quicksort

- Divide: Partition the input array A[p..r] into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that:
  - $A[k] \leq A[q]$ for each $k : p \leq k \leq q-1$
  - $A[k] > A[q]$ for each $k : q+1 \leq k \leq r$

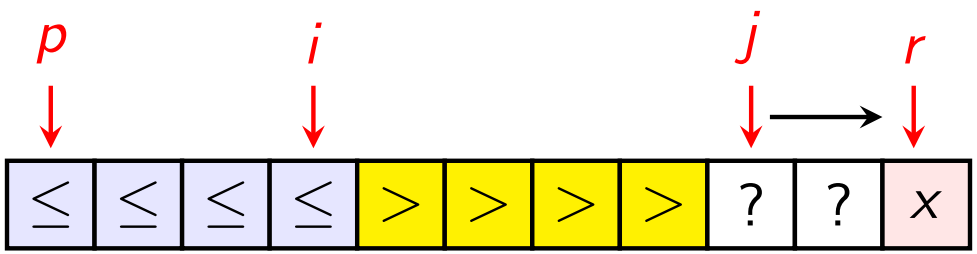
  $A[q]$ is called the pivot element.

The Partition Procedure



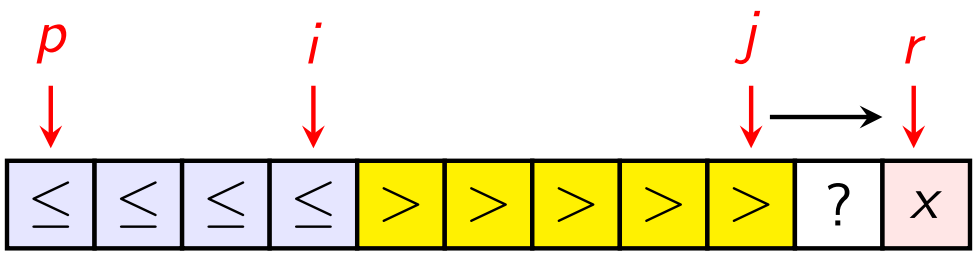
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$



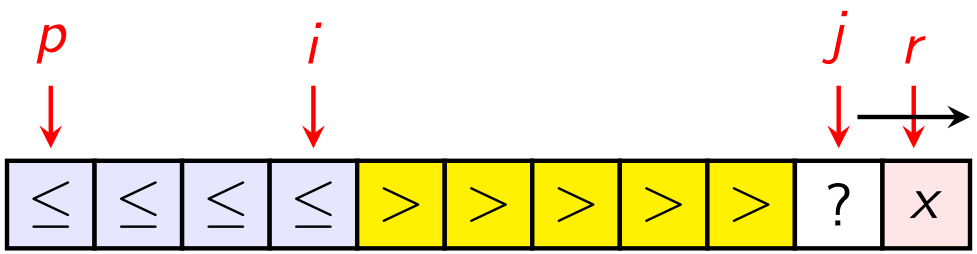
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j



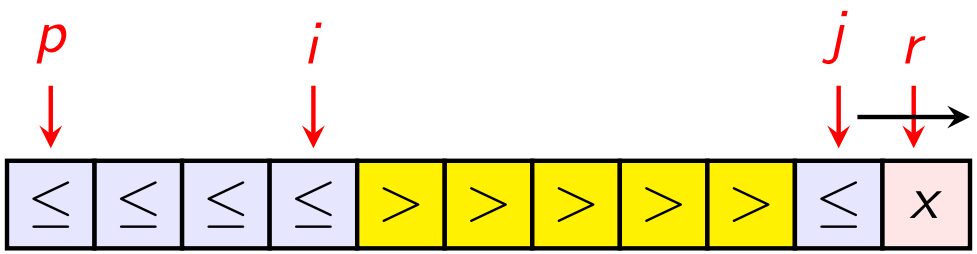
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j



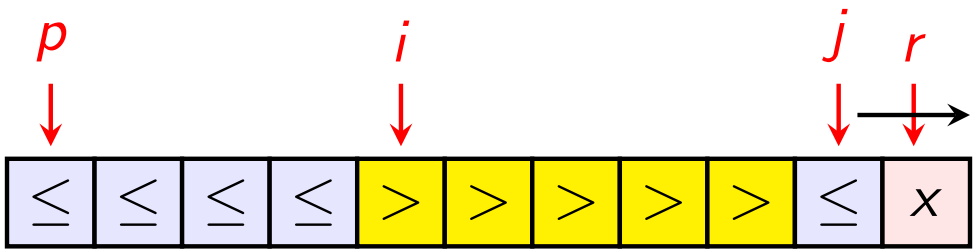
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j
- If $A[j] \leq A[r]$ then: (1) Increase i, (2) Swap $A[j]$ and $A[i]$, and (3) Increase j.



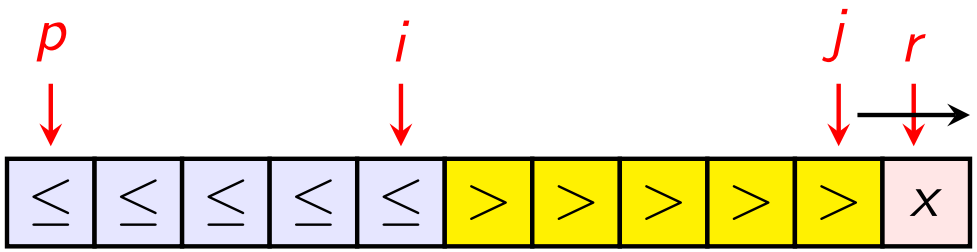
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j
- If $A[j] \leq A[r]$ then: (1) Increase i, (2) Swap $A[j]$ and $A[i]$, and (3) Increase j.



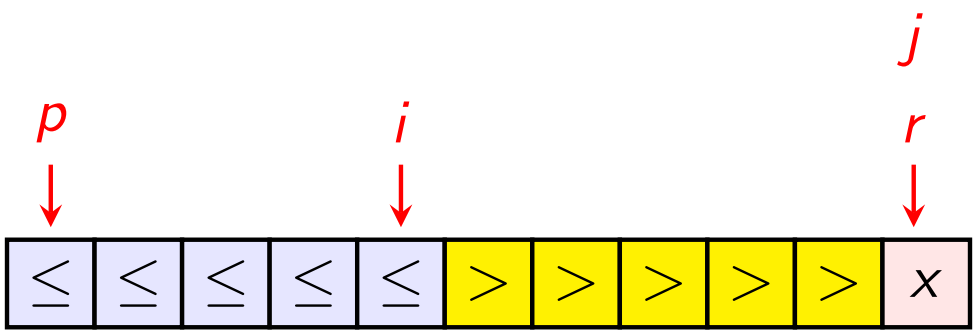
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j
- If $A[j] \leq A[r]$ then: (1) Increase i, (2) Swap $A[j]$ and $A[i]$, and (3) Increase j.



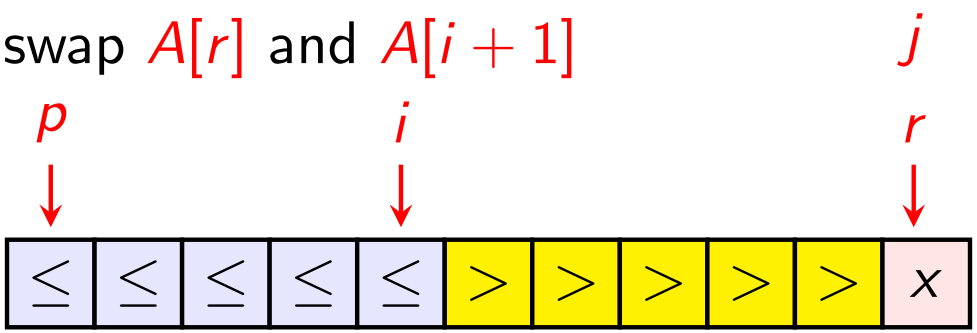
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j
- If $A[j] \leq A[r]$ then: (1) Increase i, (2) Swap $A[j]$ and $A[i]$, and (3) Increase j.



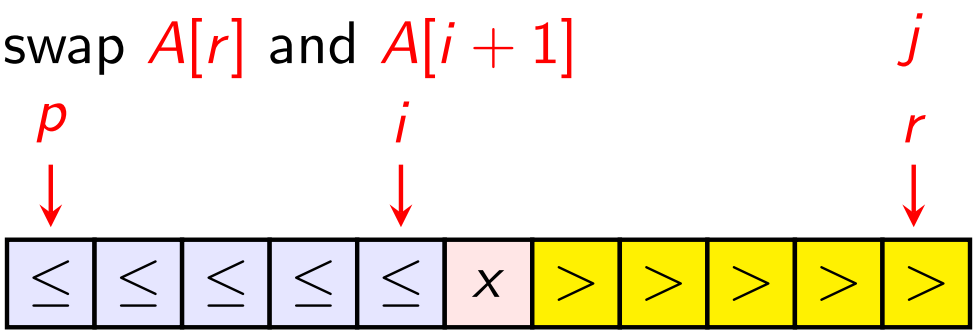
Partition: Principle

- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j
- If $A[j] \leq A[r]$ then: (1) Increase i, (2) Swap $A[j]$ and $A[i]$, and (3) Increase j.
- If $j = r$ then swap $A[r]$ and $A[i + 1]$



Partition: Principle

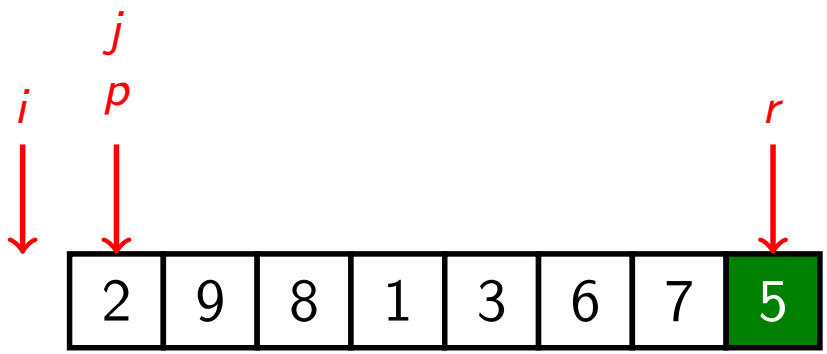
- Select the last element of the array as the pivot element.
- Initialize i to $p - 1$
- Traverse the array from left to right with an index j from p to $r - 1$
- If $A[j] > A[r]$ then increase j
- If $A[j] \leq A[r]$ then: (1) Increase i, (2) Swap $A[j]$ and $A[i]$, and (3) Increase j.
- If $j = r$ then swap $A[r]$ and $A[i + 1]$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

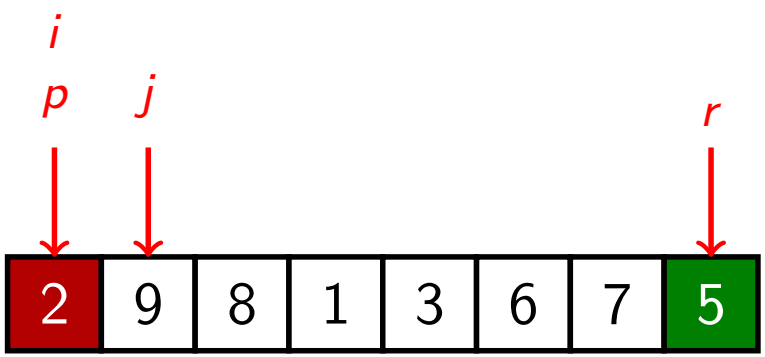
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

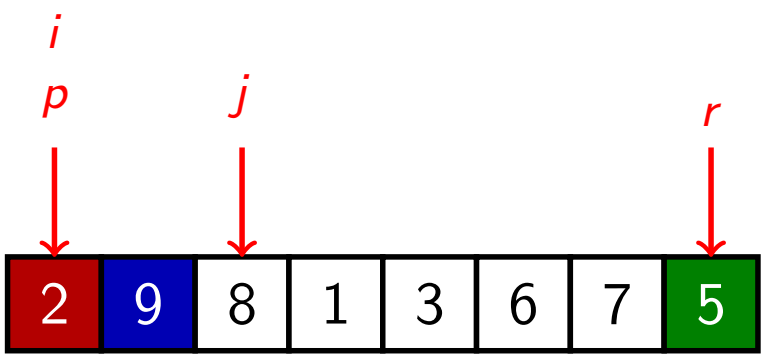
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

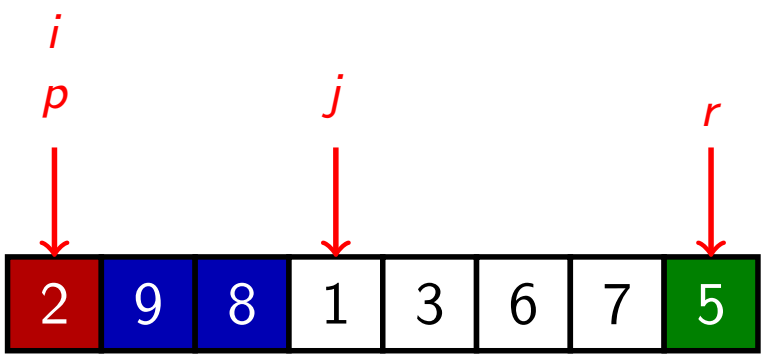
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

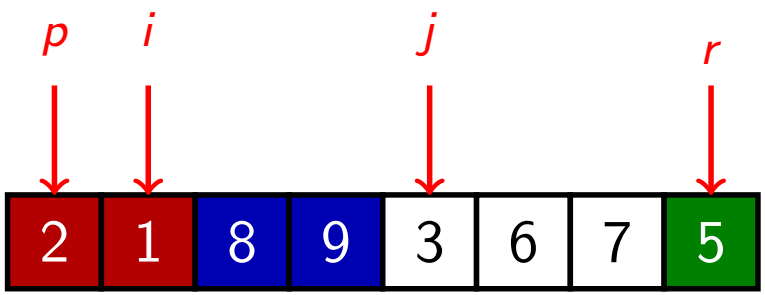
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

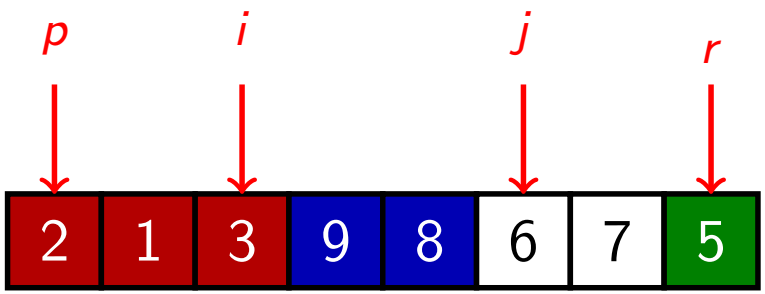
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

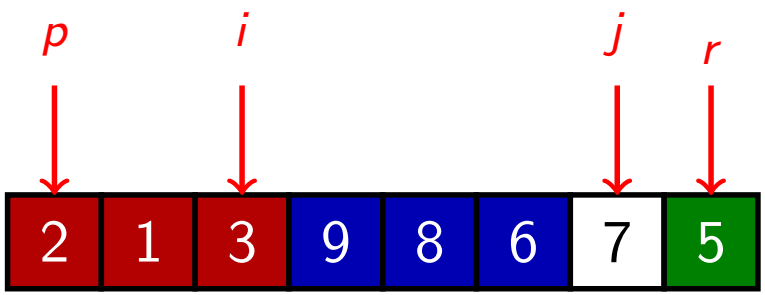
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

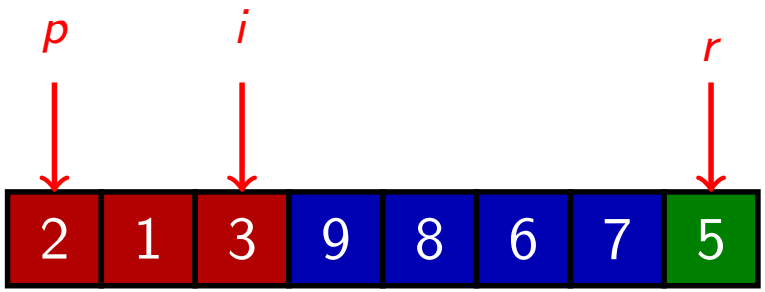
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

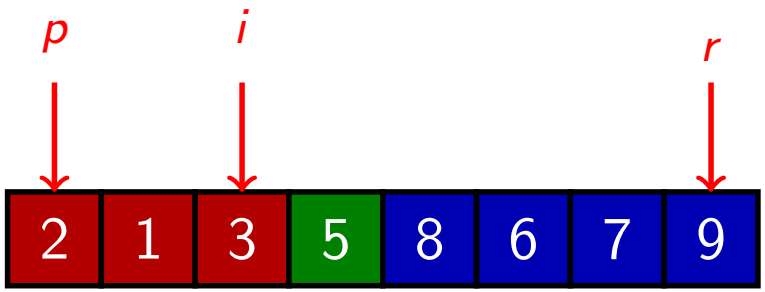
PARTITION$(A, 1, 8)$



Partition

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

PARTITION$(A, 1, 8)$



Complexity of the Partition Procedure

- Let $n = r - p + 1$

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

Complexity of the Partition Procedure

- Let $n = r - p + 1$
- Each of lines 1-2 and 7-8 takes constant time.

PARTITION(A, p, r)

```
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

Complexity of the Partition Procedure

- Let $n = r - p + 1$
- Each of lines 1-2 and 7-8 takes constant time.
- The **for** loop of lines 3-6 takes $n - 1$-iterations, each of which takes constant time.

PARTITION(A, p, r)

```
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

Complexity of the Partition Procedure

- Let $n = r - p + 1$
- Each of lines 1-2 and 7-8 takes constant time.
- The **for** loop of lines 3-6 takes $n - 1$-iterations, each of which takes constant time.

PARTITION(A, p, r)

```
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

The Partition Procedure runs in $\Theta(n)$

Quicksort

- Divide: Partition the input array A[p..r] into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that:
  - $A[k] \leq A[q]$ for each $k : p \leq k \leq q-1$
  - $A[k] > A[q]$ for each $k : q+1 \leq k \leq r$

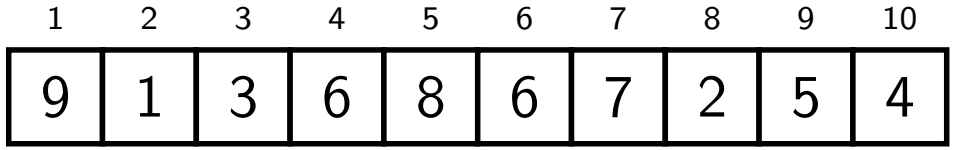
  $A[q]$ is called the pivot element.
- Conquer: Sort each of the two subarrays $A[p..q-1]$ and $A[q..r]$ recursively.
- Combine: Trivial.
- Base Cases: When $p \geq r$

Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

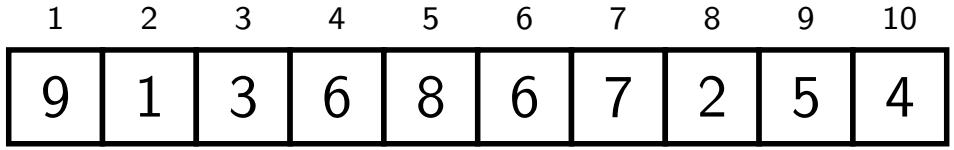
Quick Sort

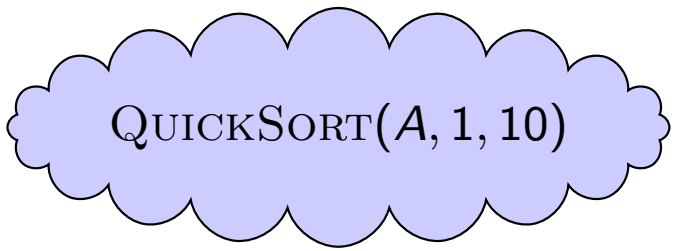
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

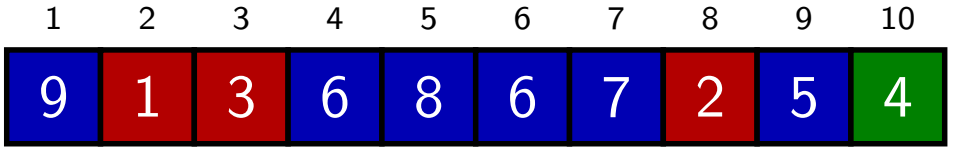




(A,1,10)

Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



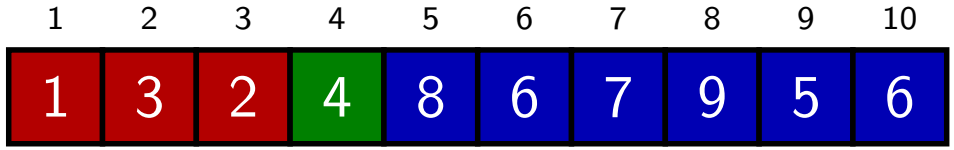
(A,1,10)

QuickSort(A, 1, 10)

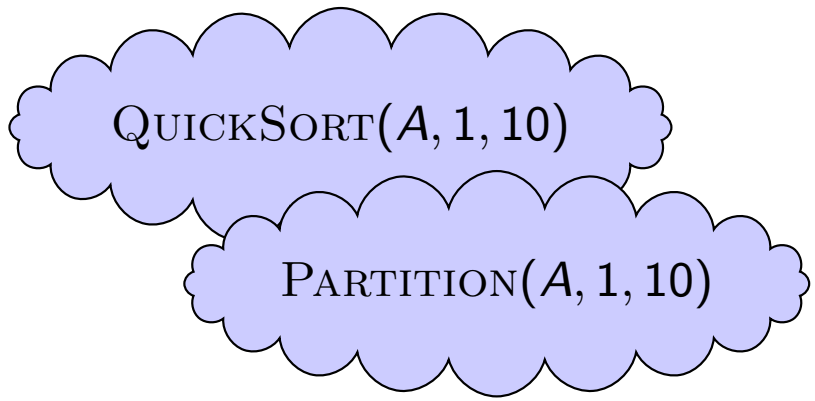
Partition(A, 1, 10)

Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

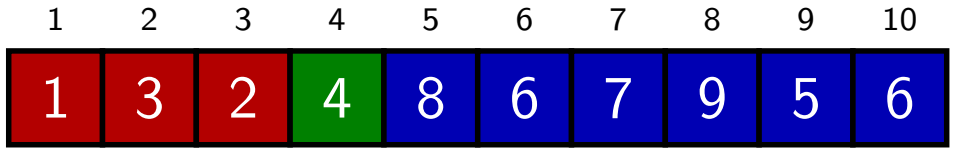


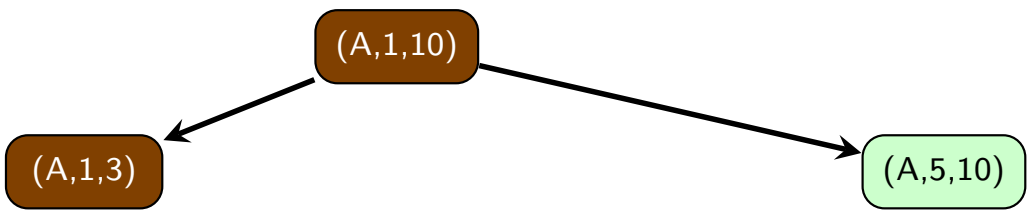
(A,1,10)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



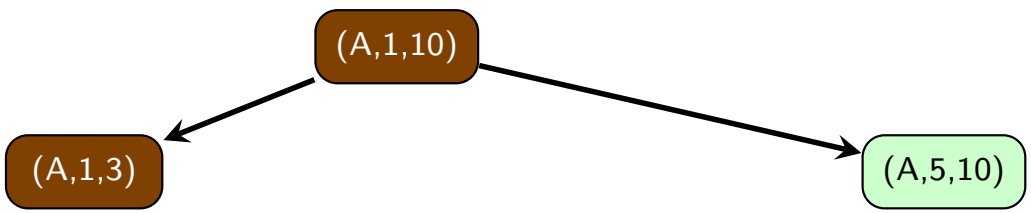


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	3	2	4	8	6	7	9	5	6

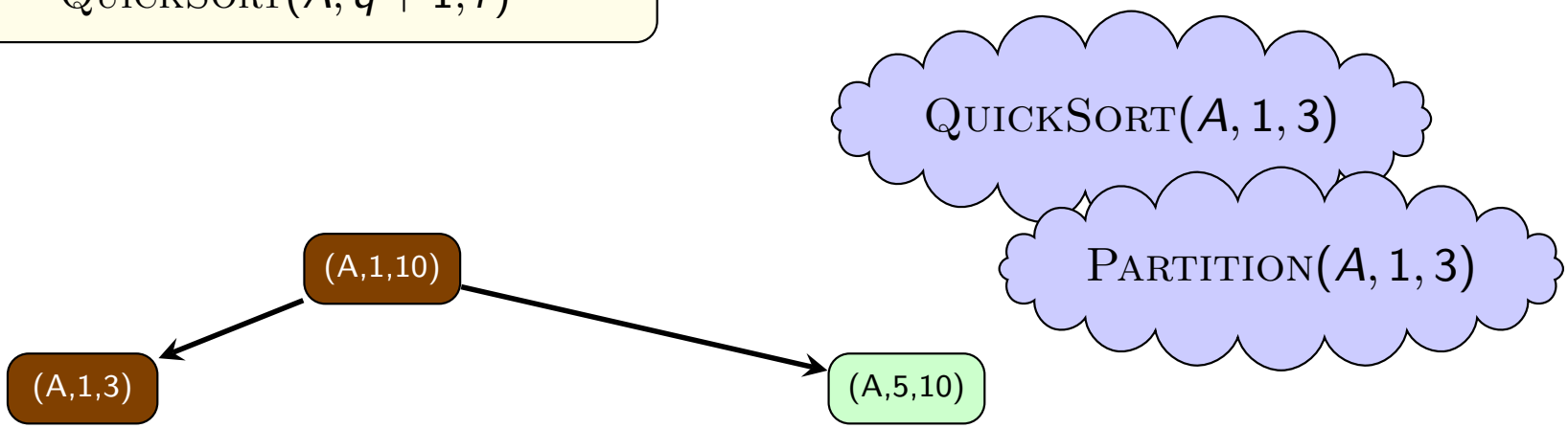
QuickSort(A, 1, 3)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

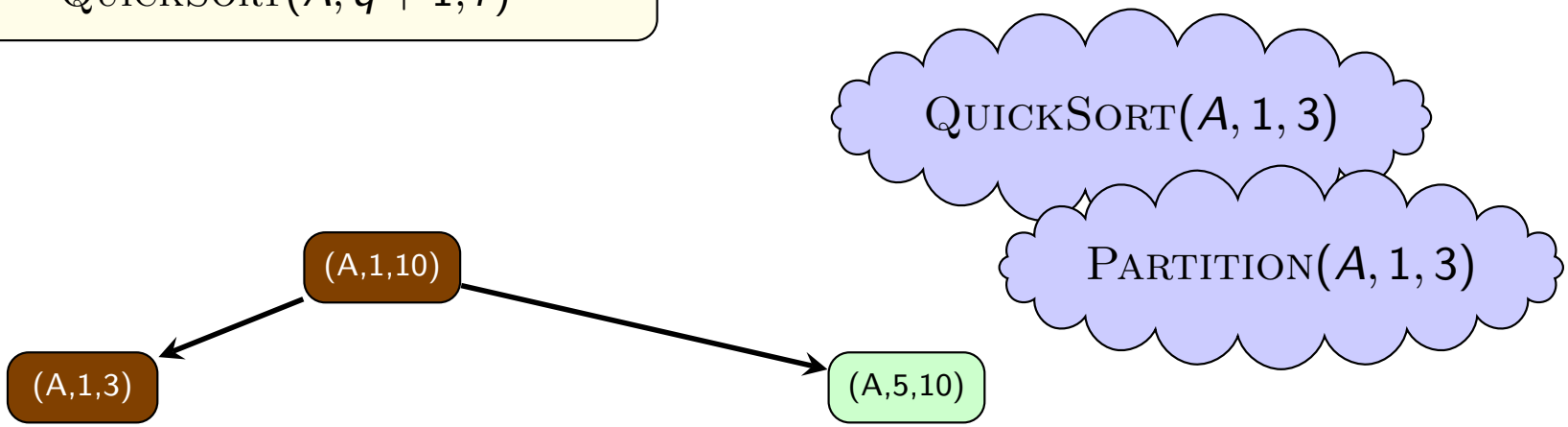
1	2	3	4	5	6	7	8	9	10
1	3	2	4	8	6	7	9	5	6



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

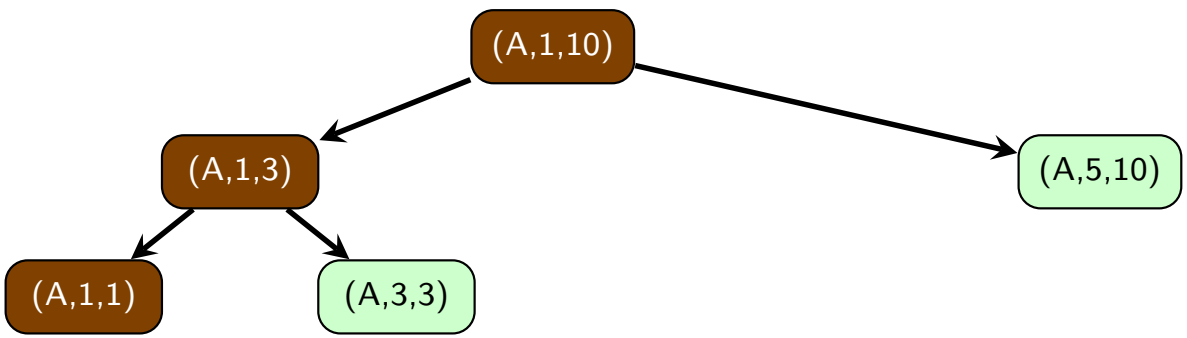
1	2	3	4	5	6	7	8	9	10
1	2	3	4	8	6	7	9	5	6



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	8	6	7	9	5	6

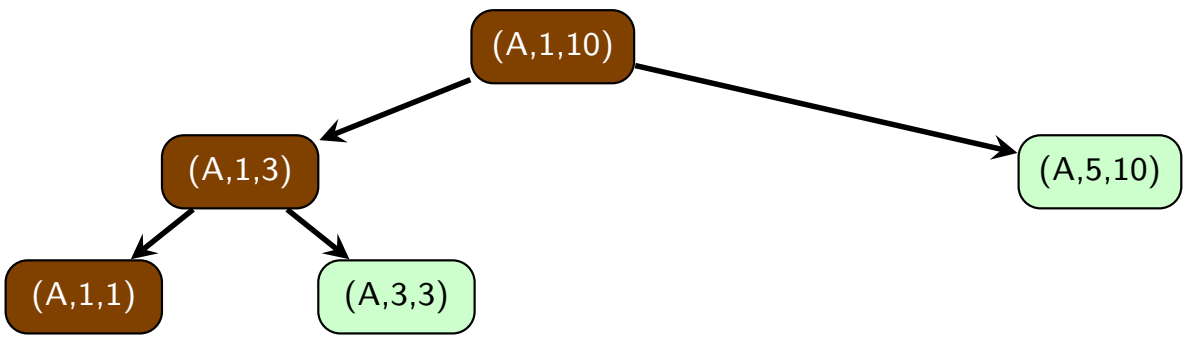


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	8	6	7	9	5	6

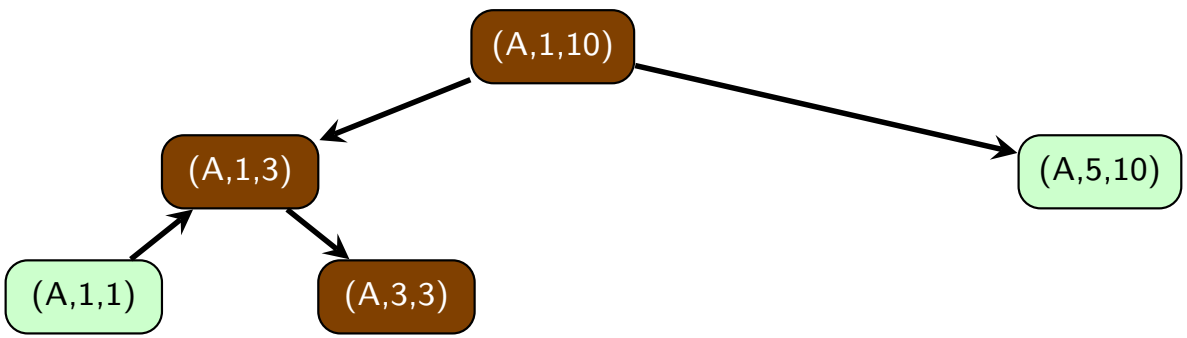
QuickSort(A, 1, 1)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	8	6	7	9	5	6

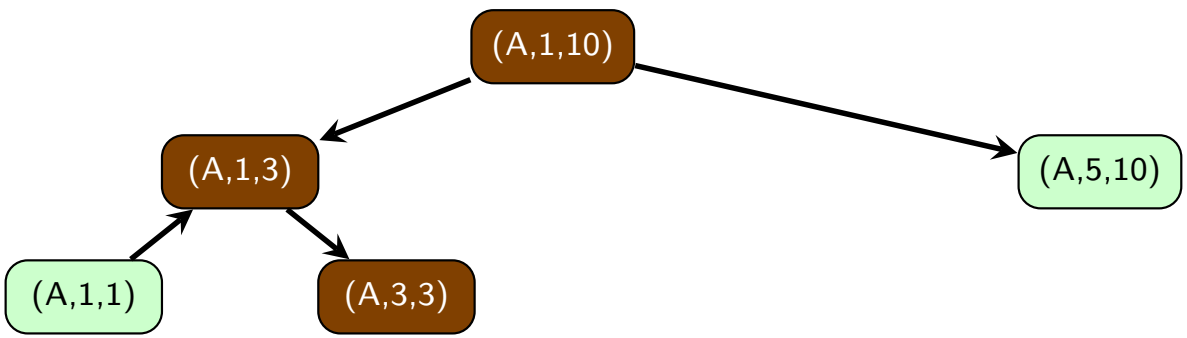


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	8	6	7	9	5	6

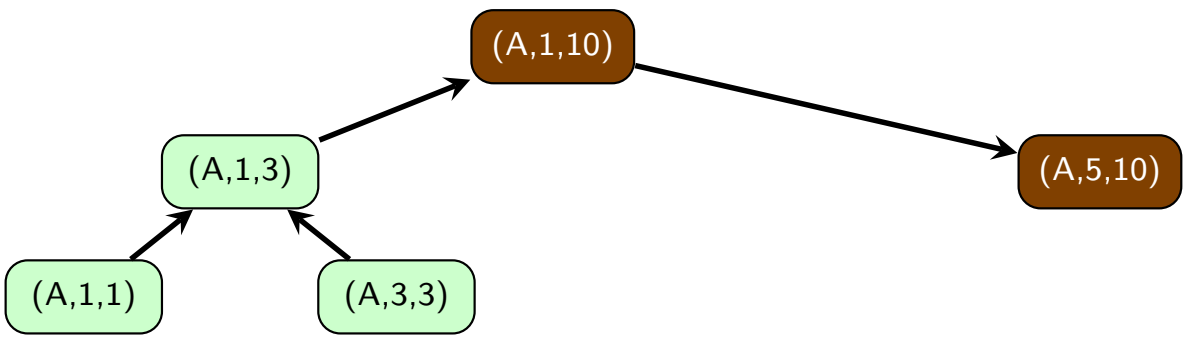
QuickSort(A, 3, 3)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	8	6	7	9	5	6

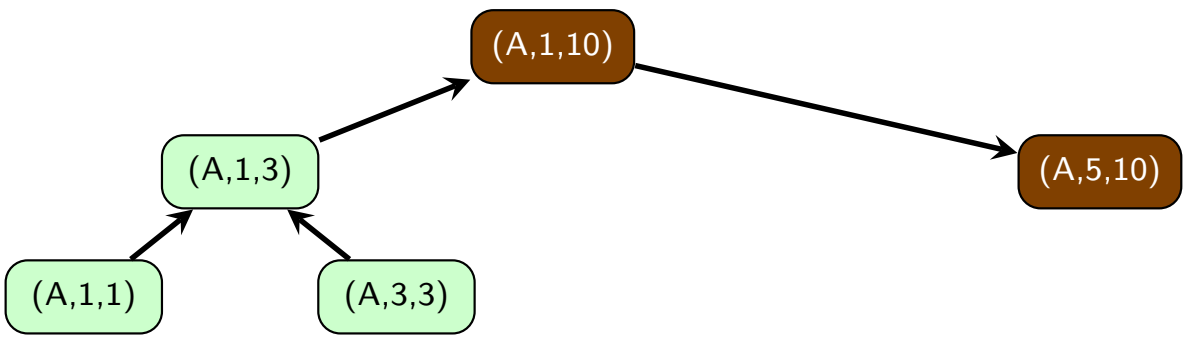


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

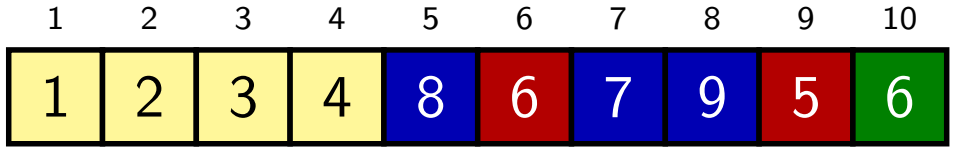
1	2	3	4	5	6	7	8	9	10
1	2	3	4	8	6	7	9	5	6

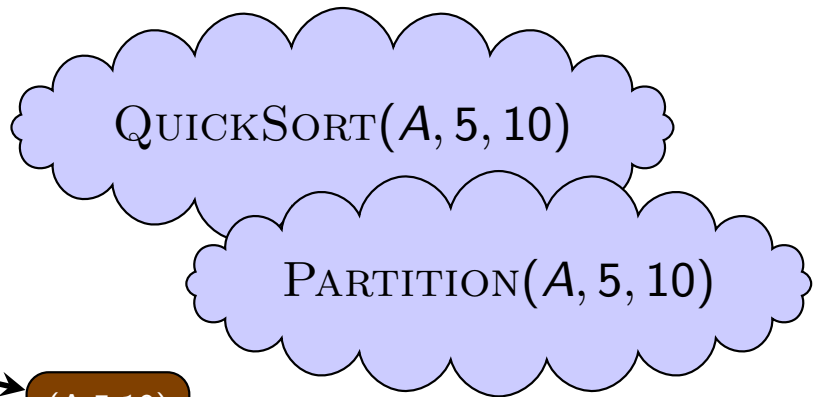
QuickSort$(A, 5, 10)$

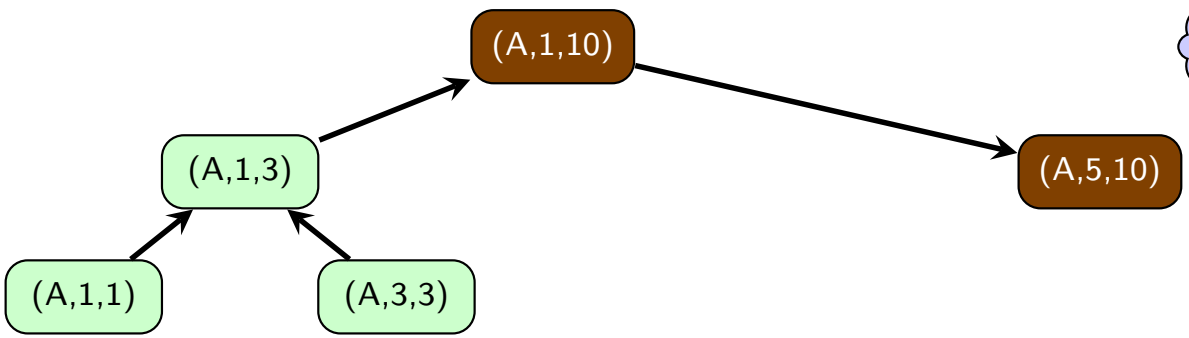


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

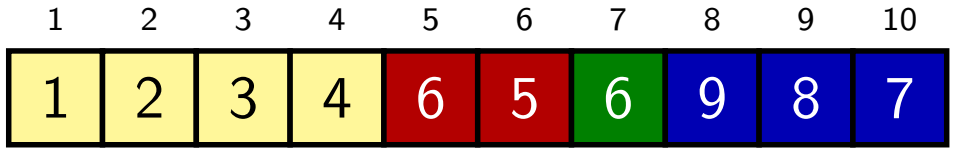


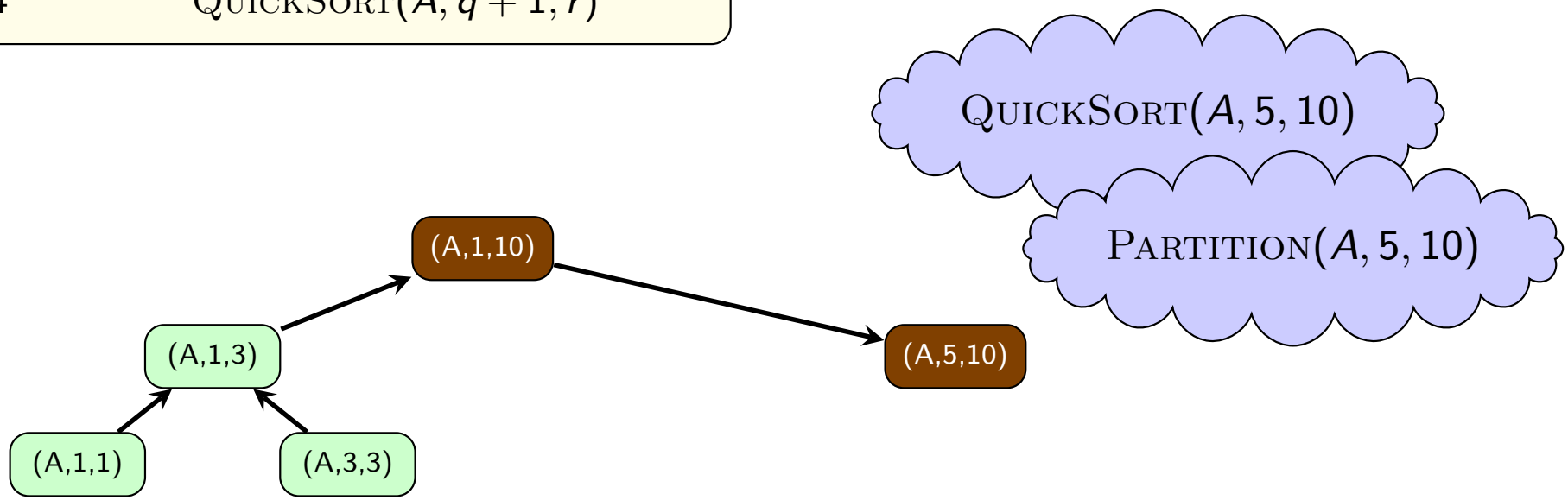




Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

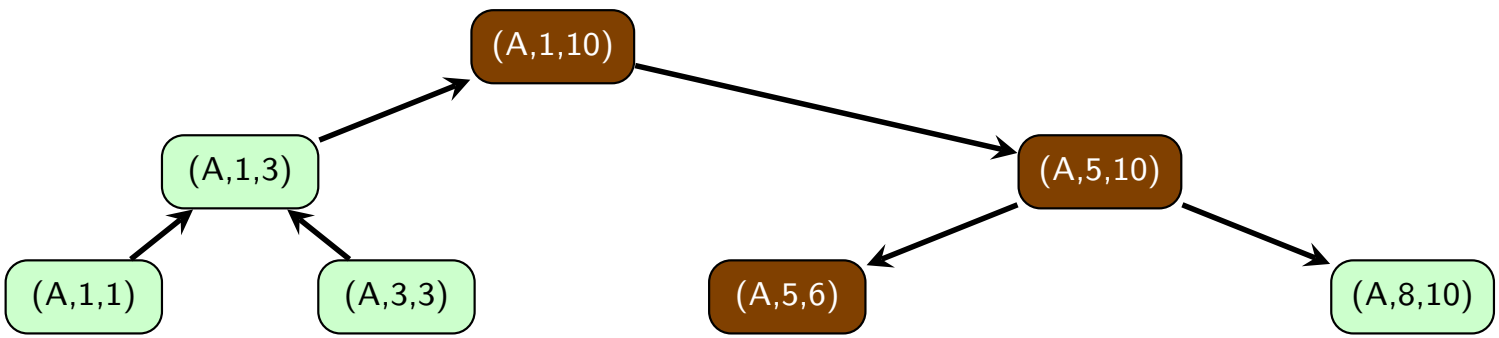




Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	6	5	6	9	8	7

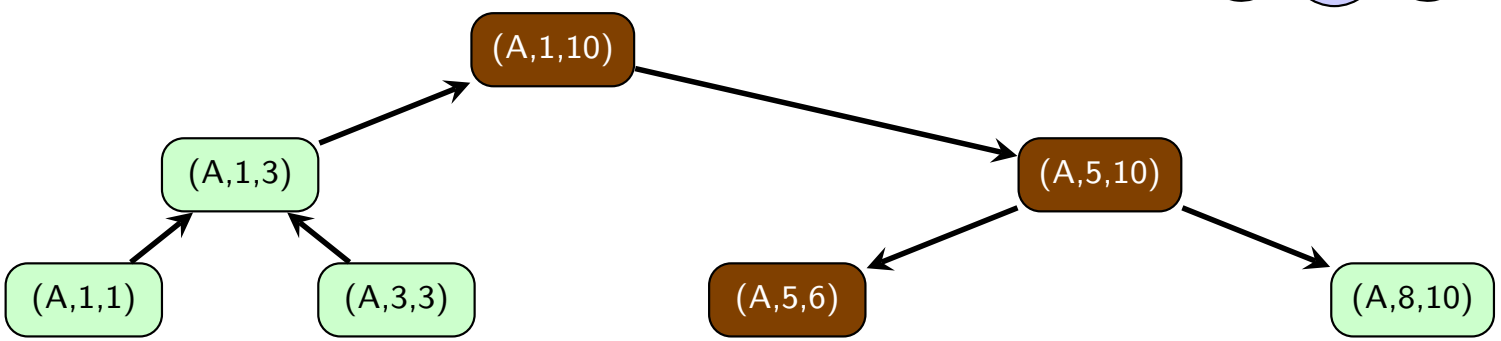


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	6	5	6	9	8	7

QuickSort(A, 5, 6)



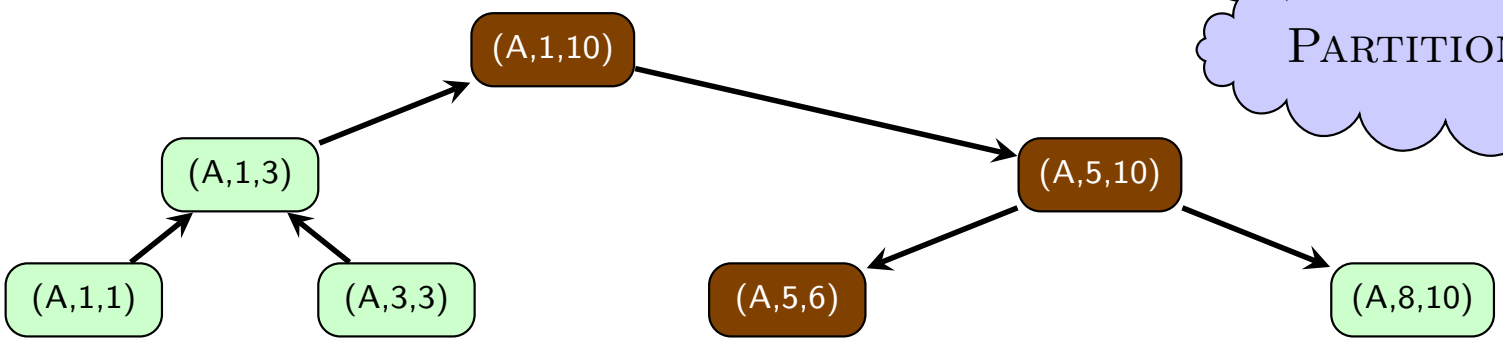
Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	6	5	6	9	8	7

QuickSort(A, 5, 6)

Partition(A, 5, 6)



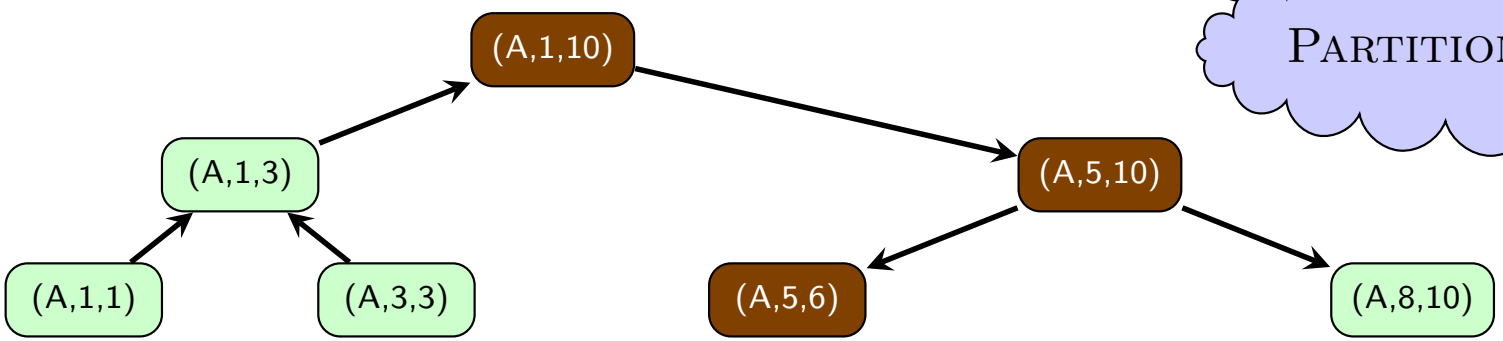
Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

QuickSort(A, 5, 6)

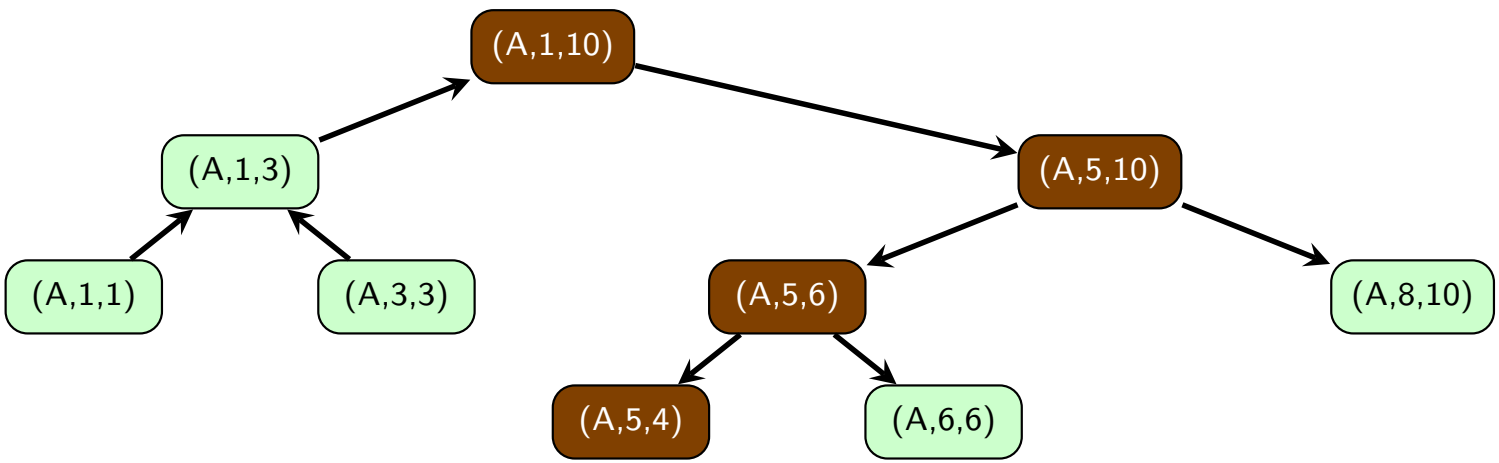
Partition(A, 5, 6)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

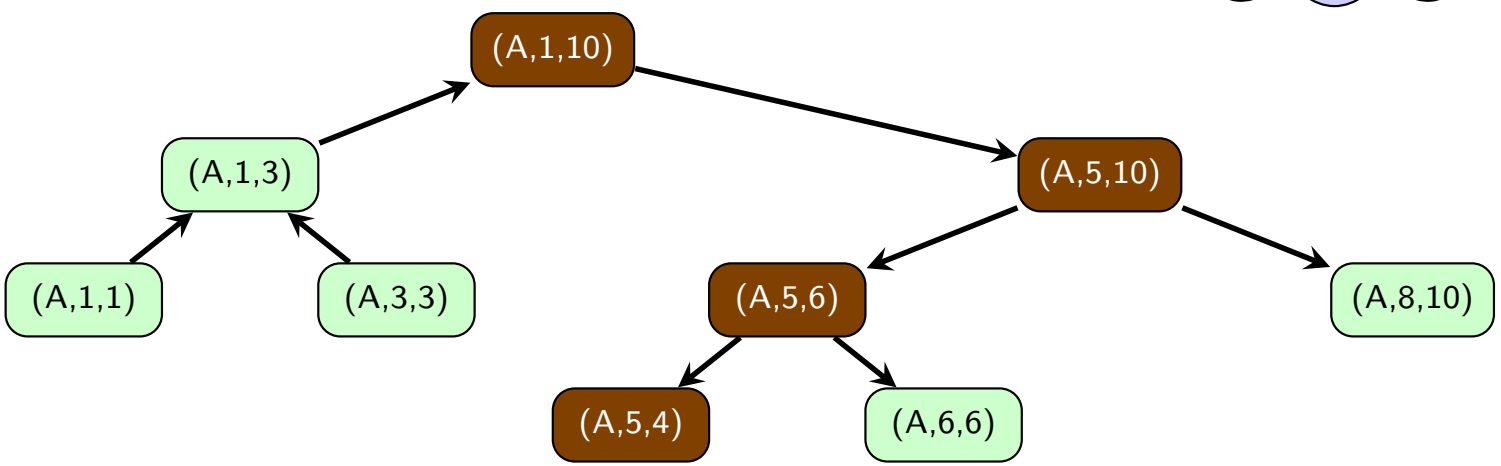


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

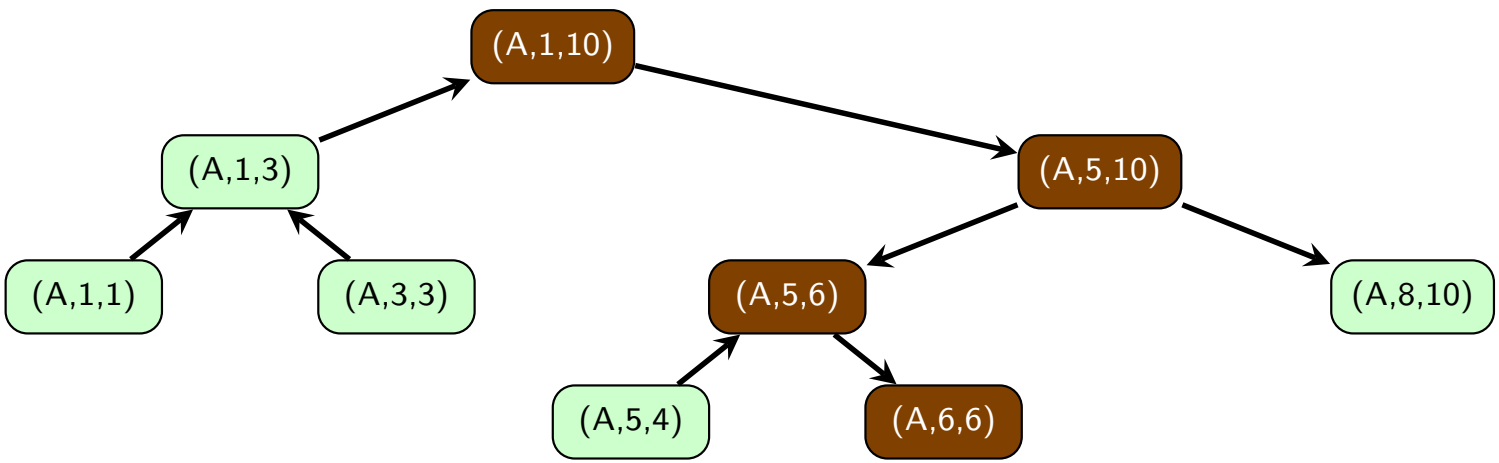
QuickSort(A, 5, 4)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

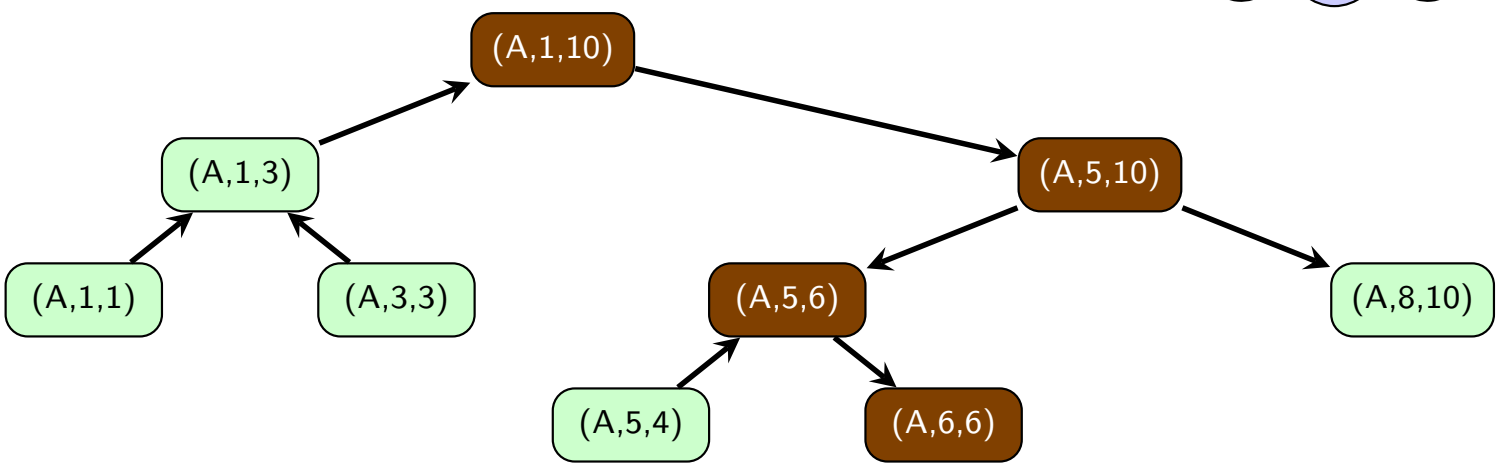


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

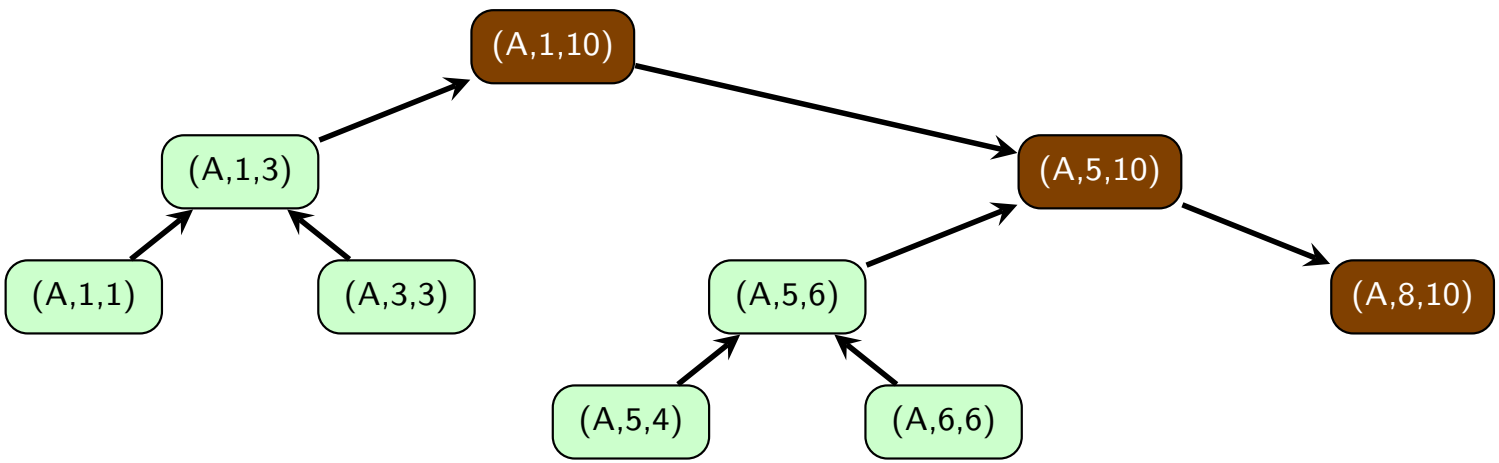
QuickSort(A, 6, 6)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

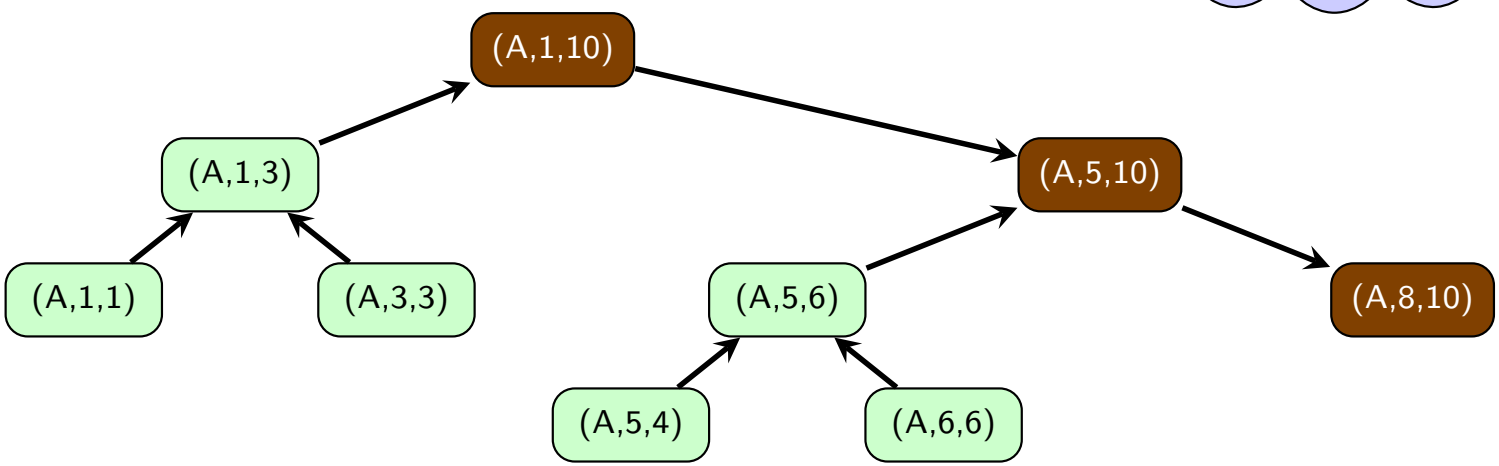


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

QuickSort$(A, 8, 10)$



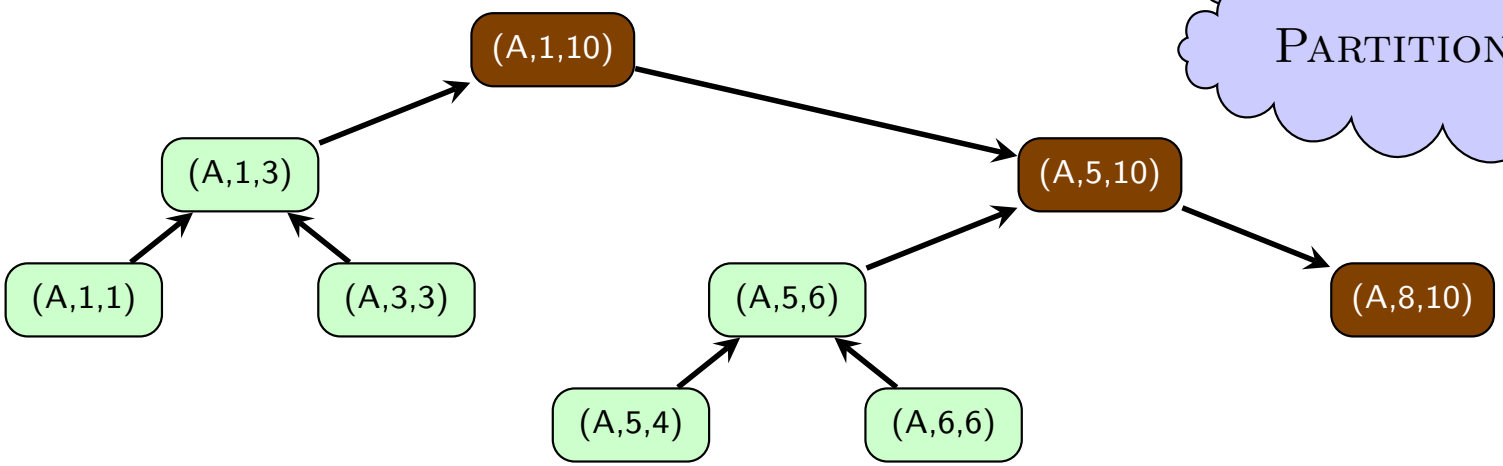
Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	9	8	7

QuickSort(A, 8, 10)

Partition(A, 8, 10)



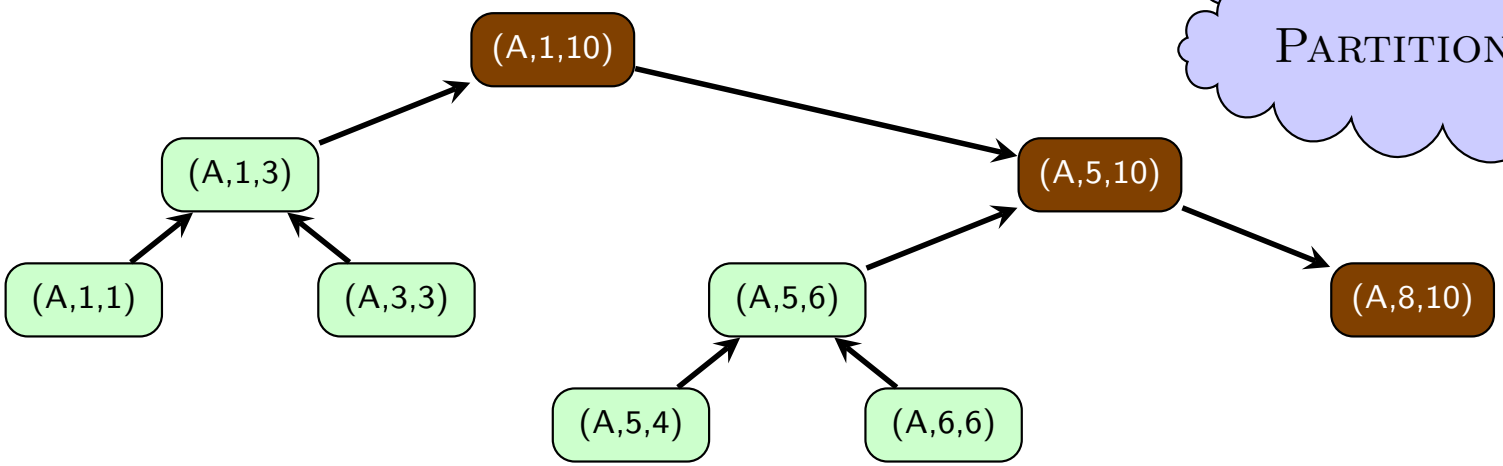
Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

QuickSort(A, 8, 10)

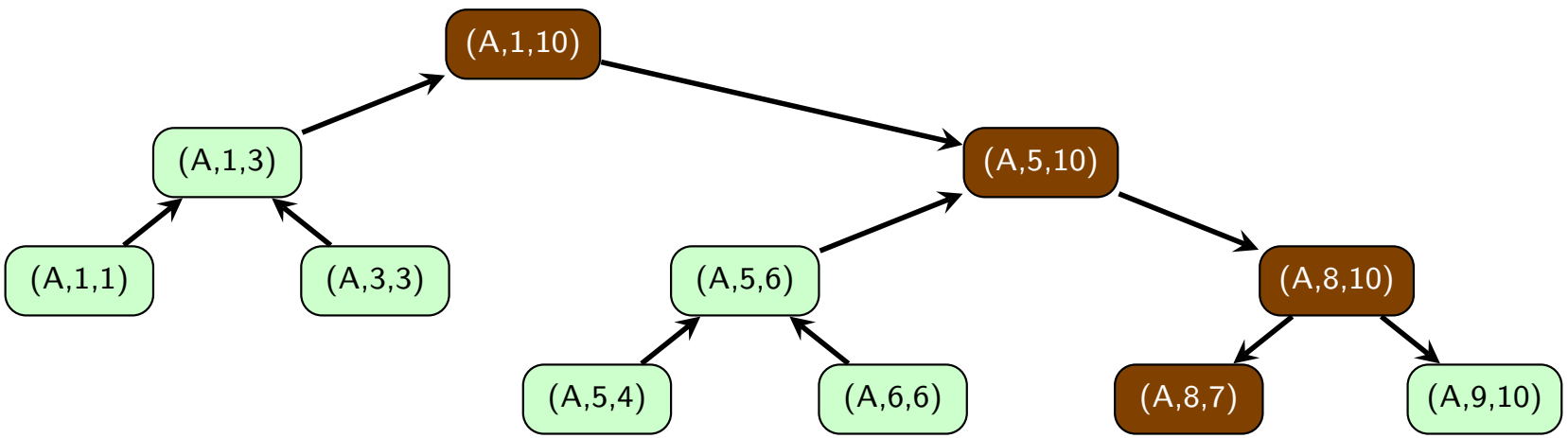
Partition(A, 8, 10)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

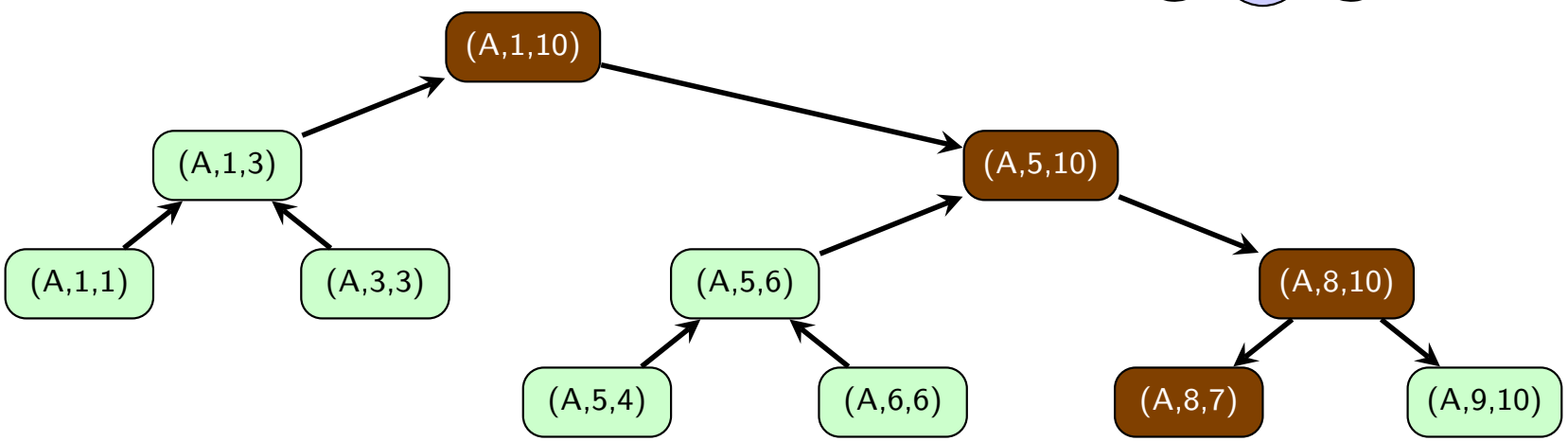


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

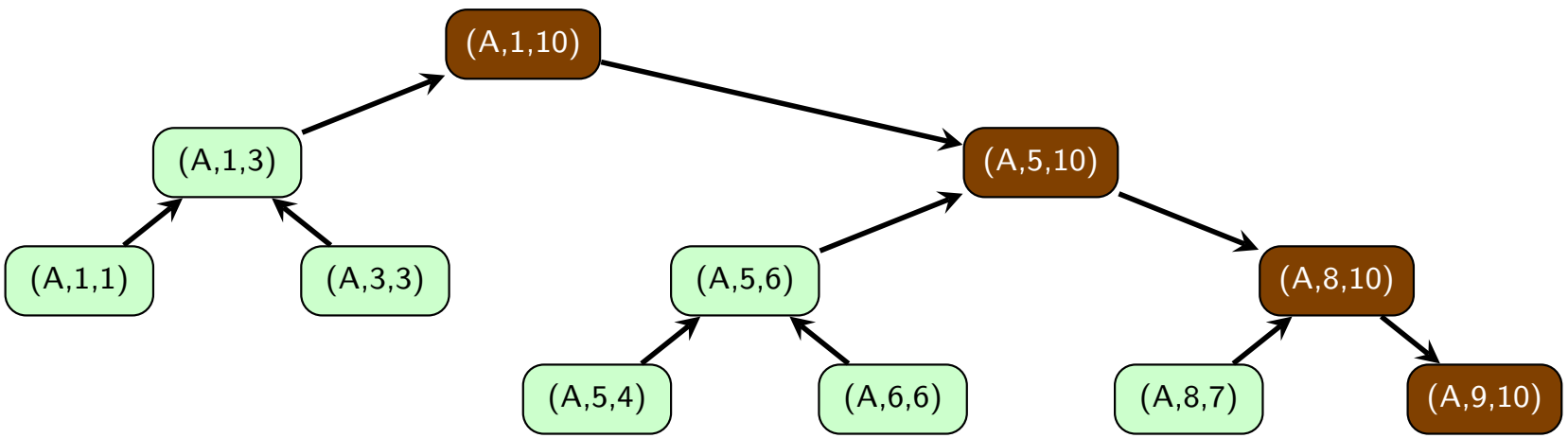
QuickSort(A, 8, 7)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

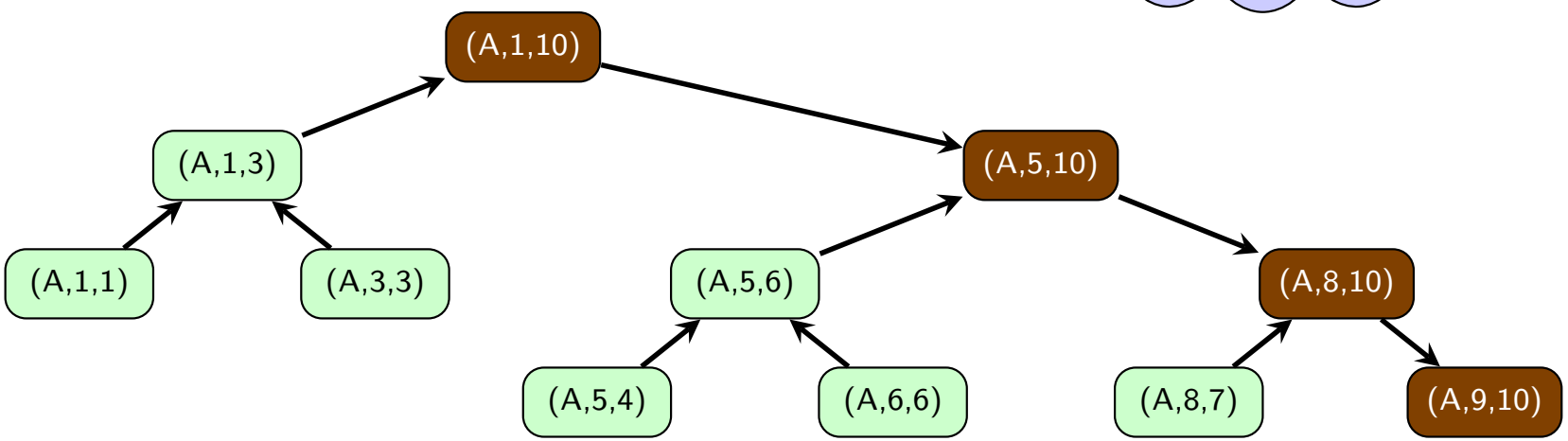


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

QuickSort(A, 9, 10)



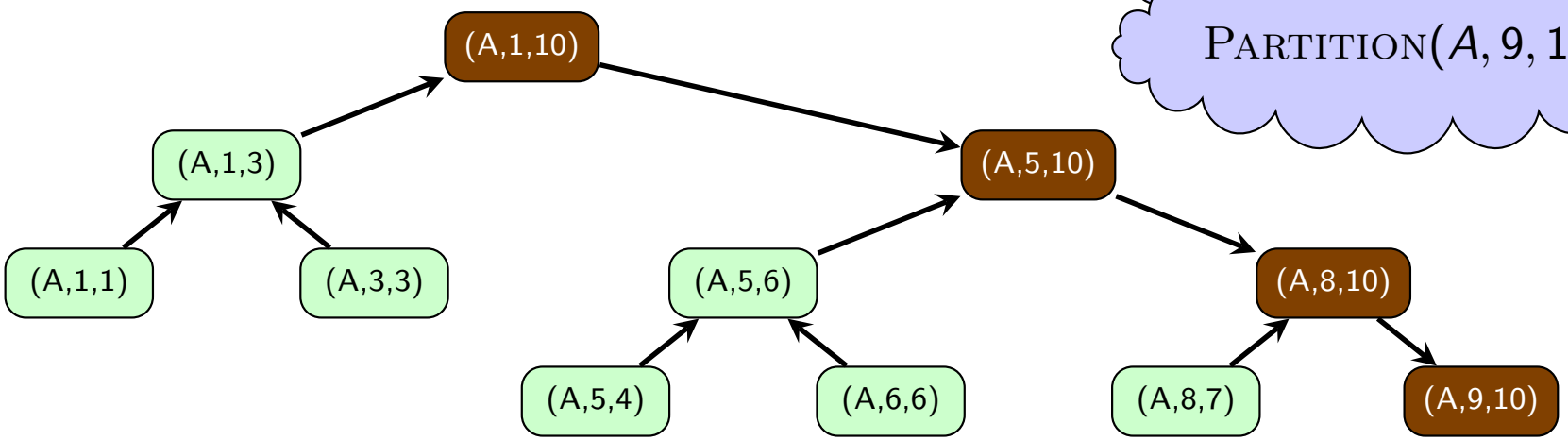
Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

QuickSort(A, 9, 10)

Partition(A, 9, 10)



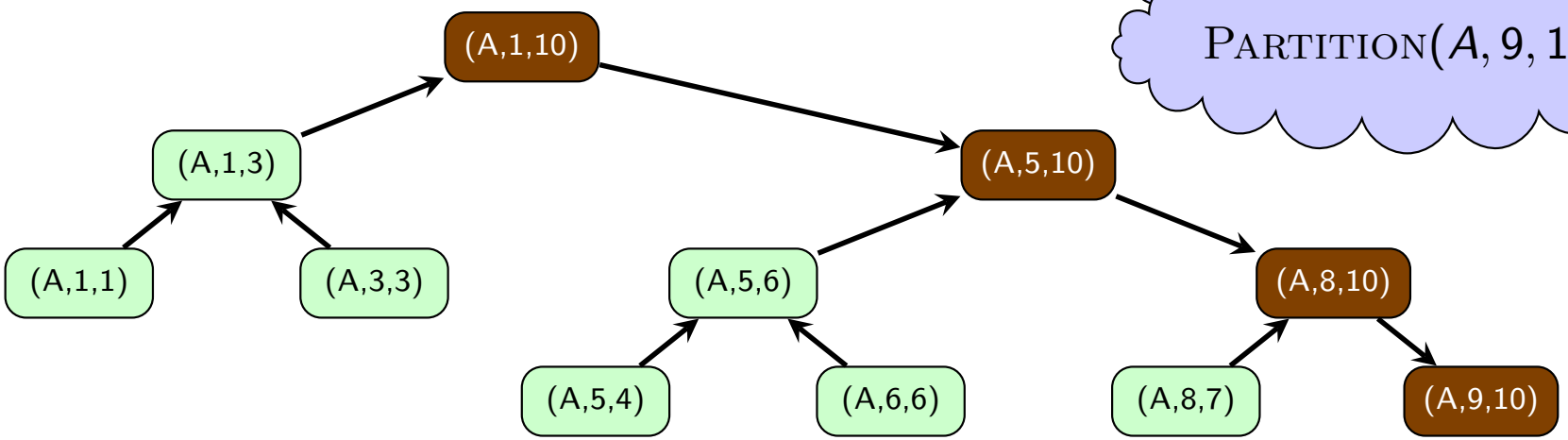
Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

QuickSort(A, 9, 10)

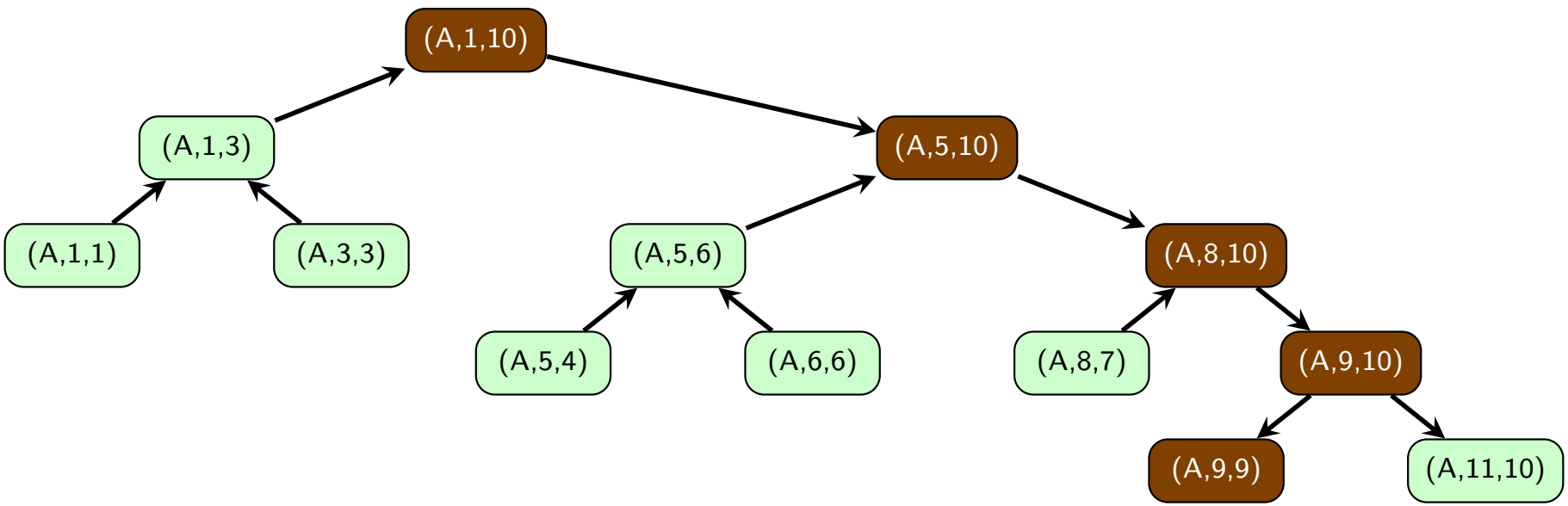
Partition(A, 9, 10)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

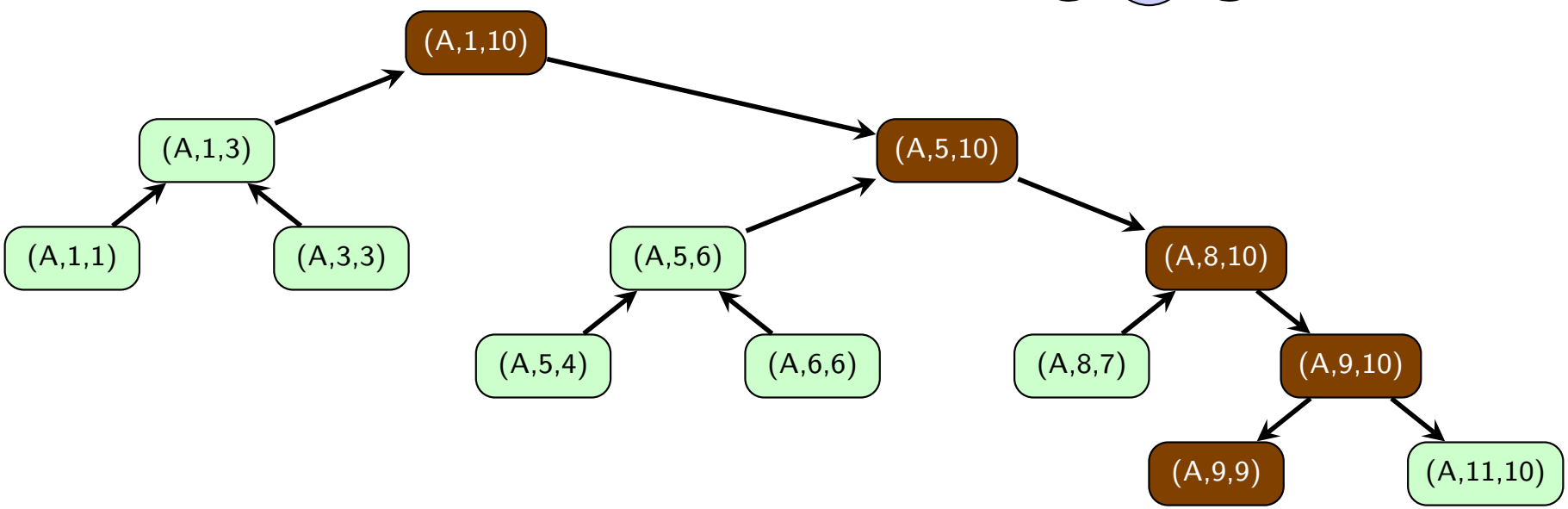


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

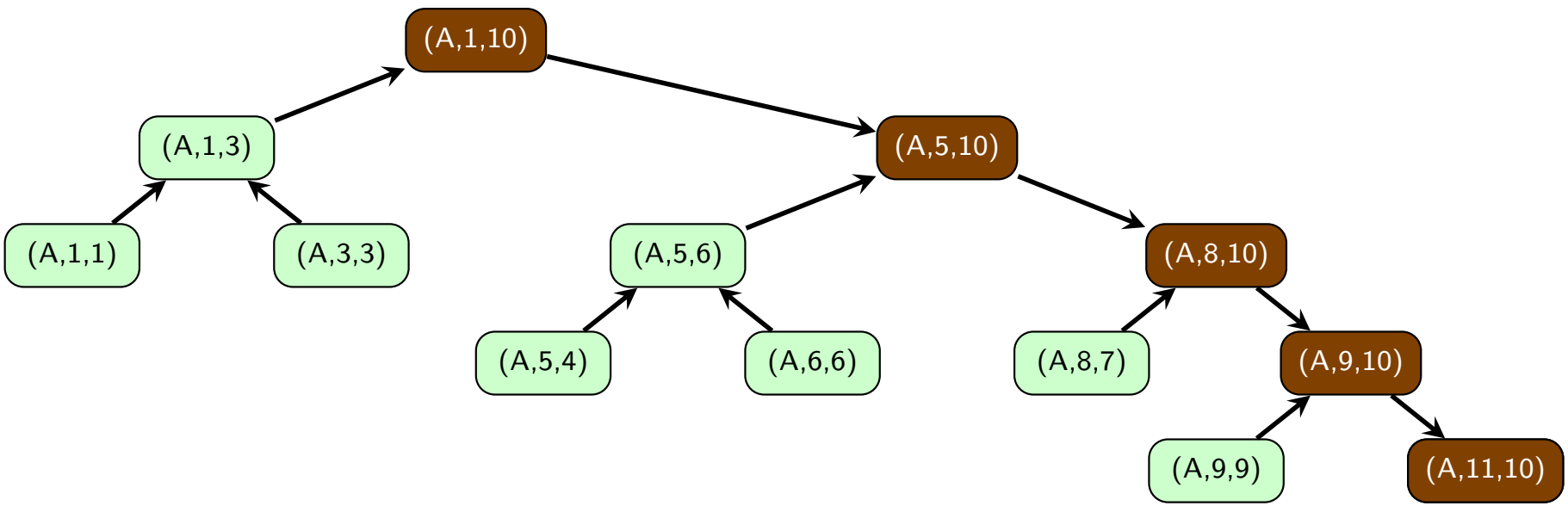
QuickSort(A, 9, 9)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

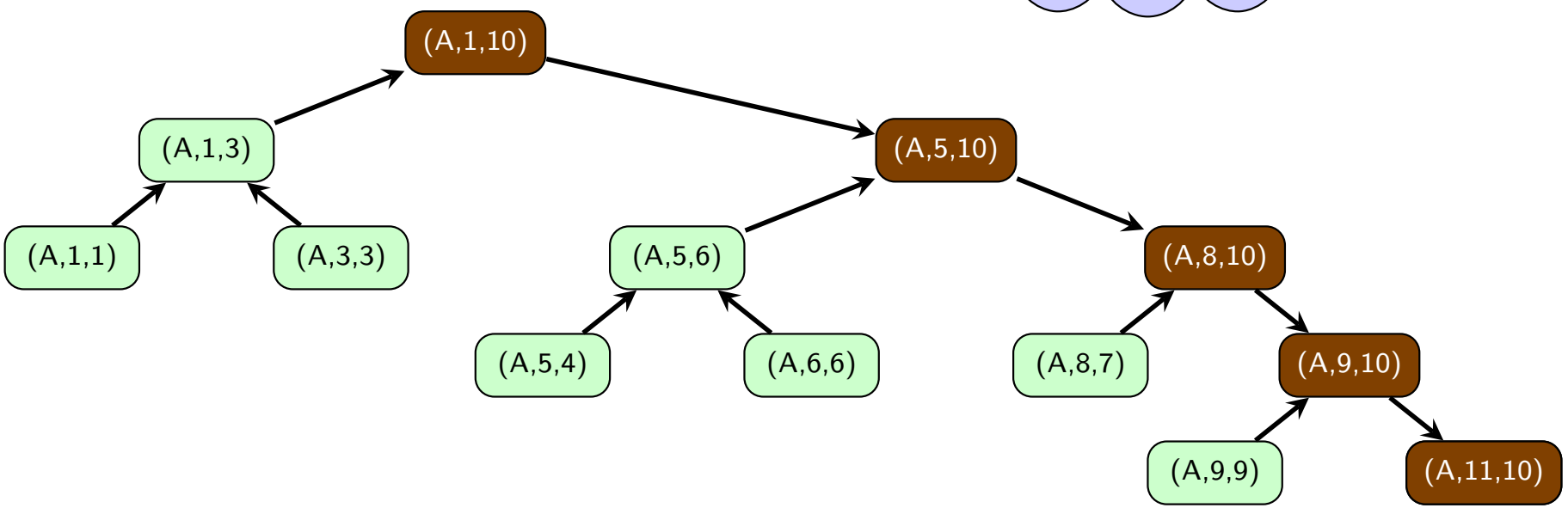


Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9

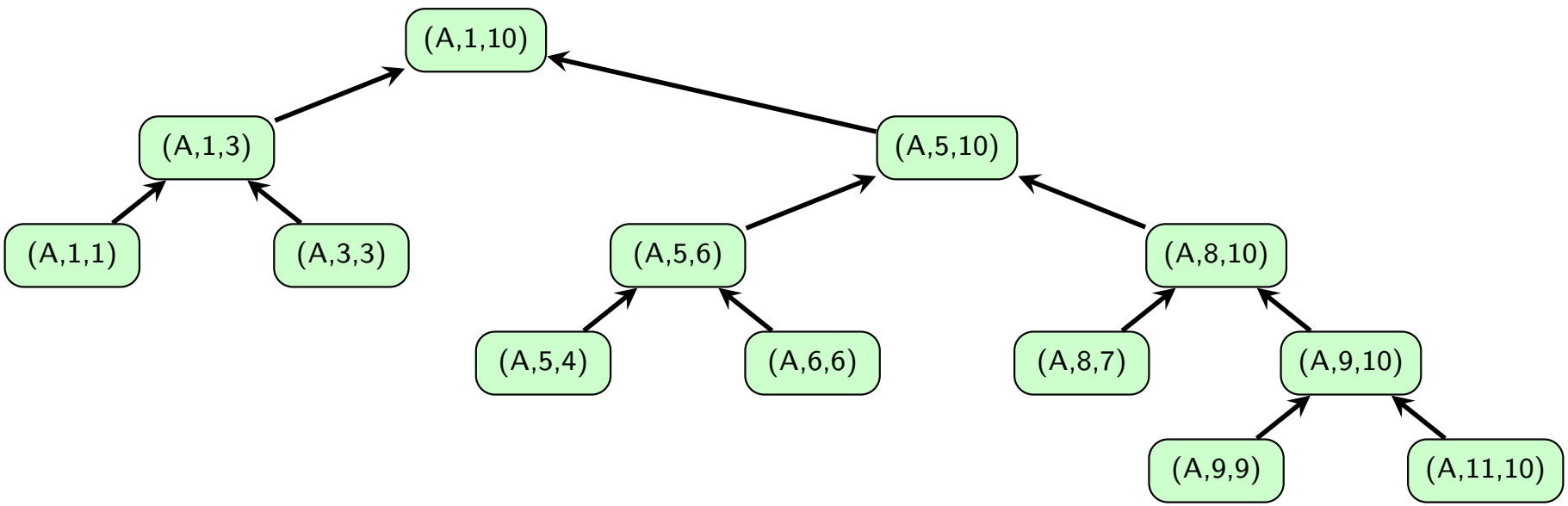
QuickSort(A, 11, 10)



Quick Sort

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	6	7	8	9



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

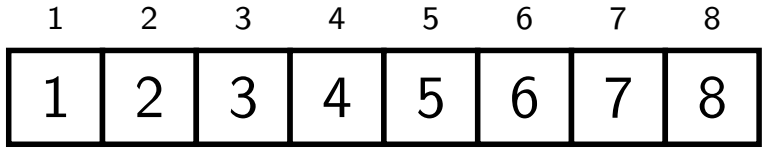
- Worst Case:
  - $q = p$ or $q = r$ (i.e., the input array is (reverse) sorted)
  - The sizes of the two subproblems are $n - 1$ and 0 respectively.

QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

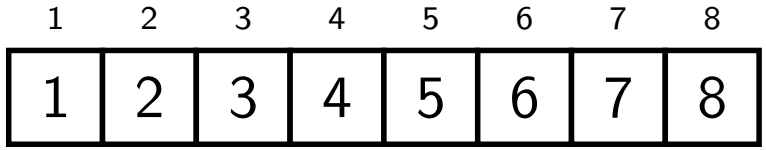
QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



QuickSort: Worst-Case Behavior

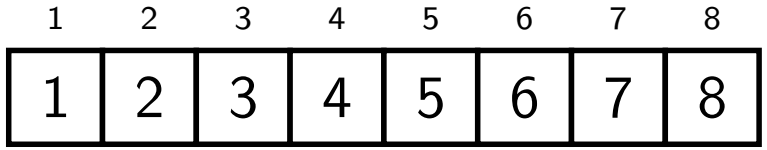
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



(A,1,8)

QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

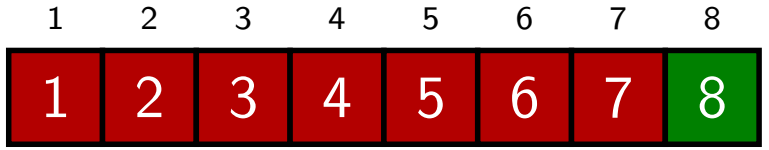


(A,1,8)

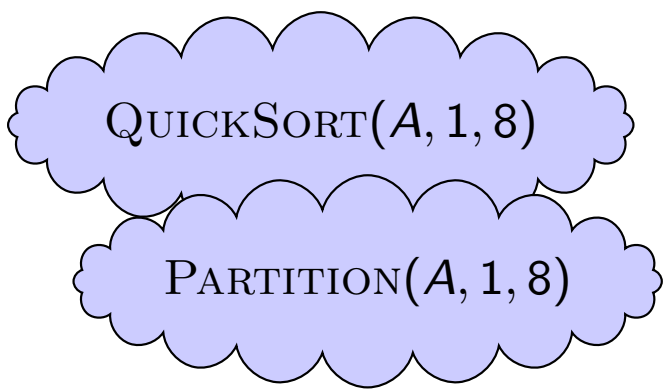
QuickSort(A, 1, 8)

QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

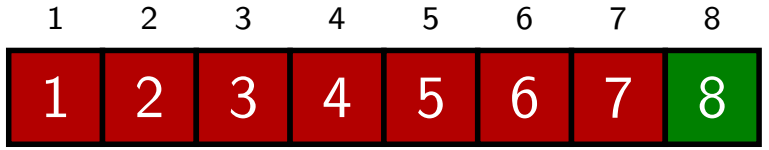


(A,1,8)

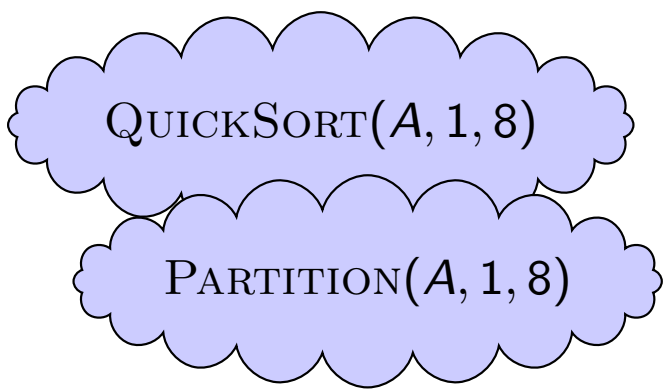


QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

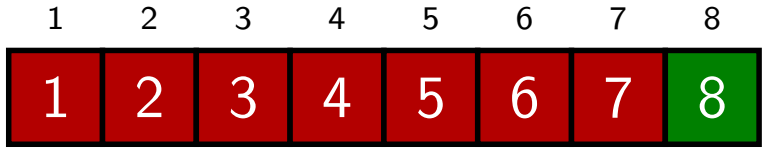


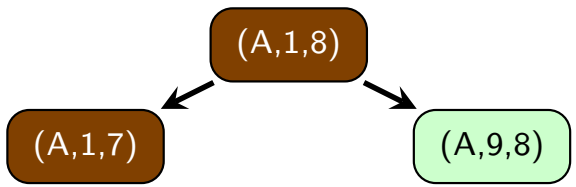
(A,1,8)



QuickSort: Worst-Case Behavior

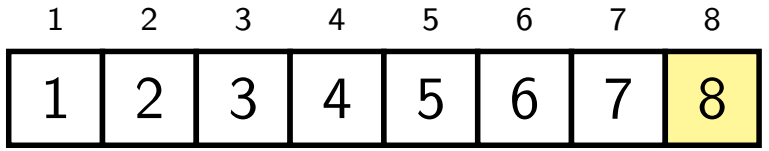
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

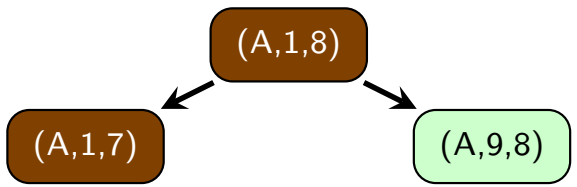


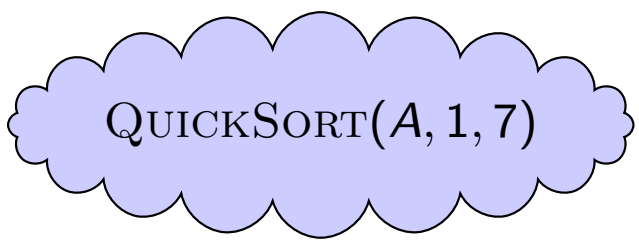


QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

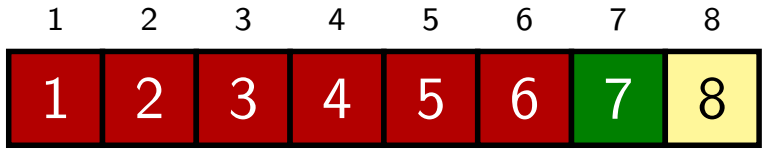


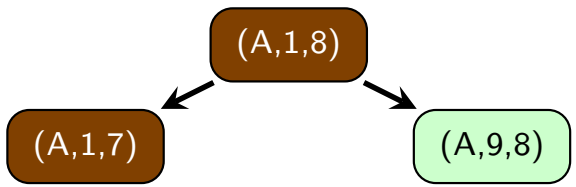


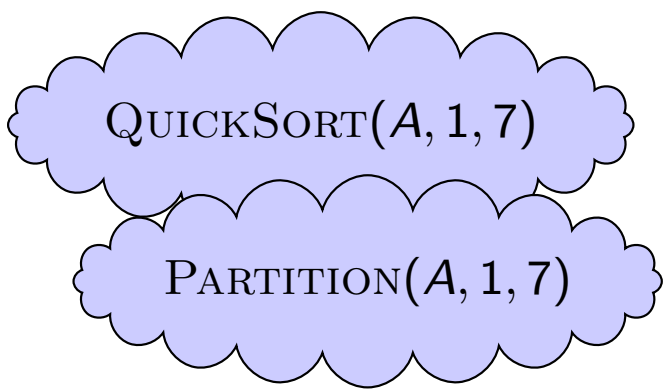


QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

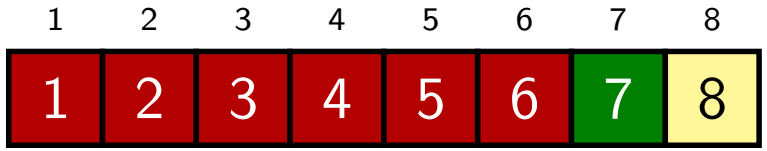


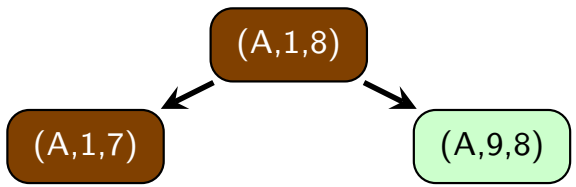


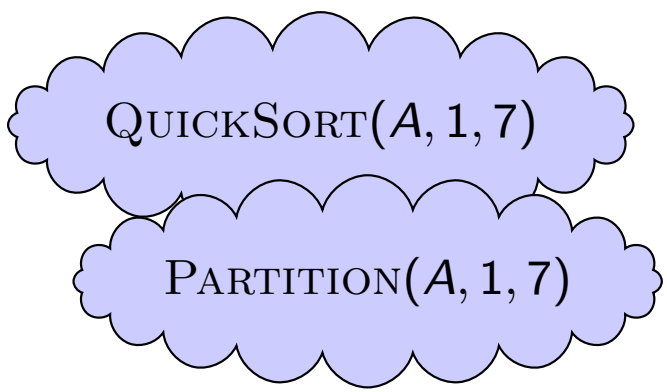


QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

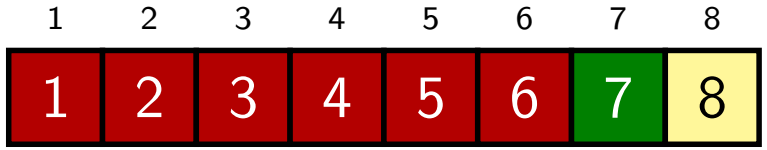


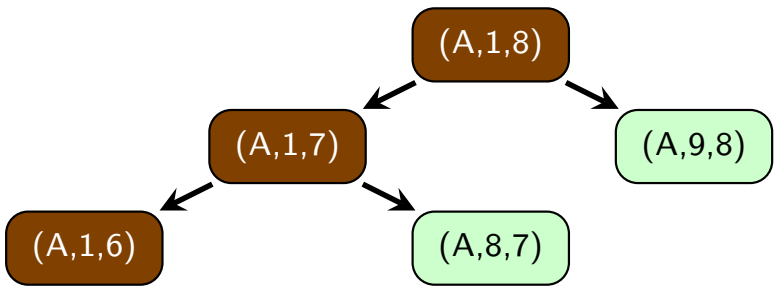




QuickSort: Worst-Case Behavior

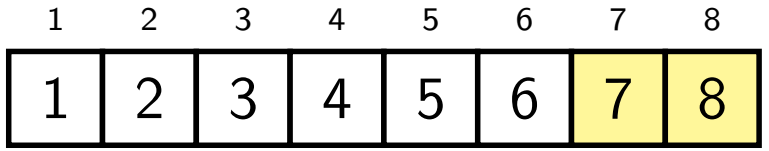
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

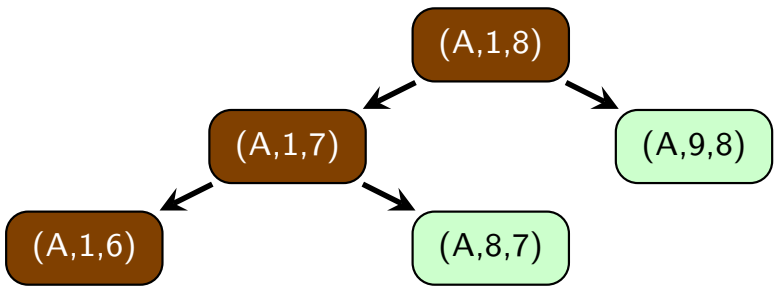




QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

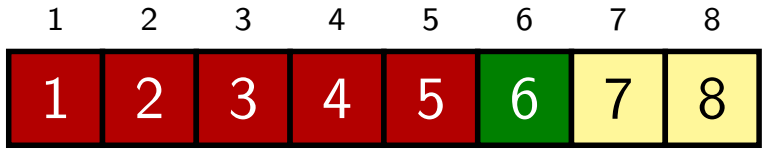


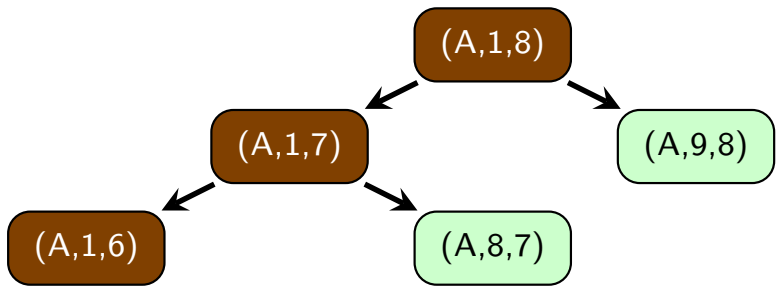


QuickSort(A, 1, 6)

QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



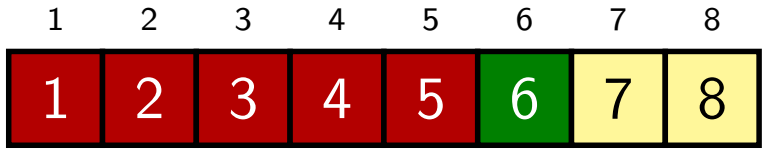


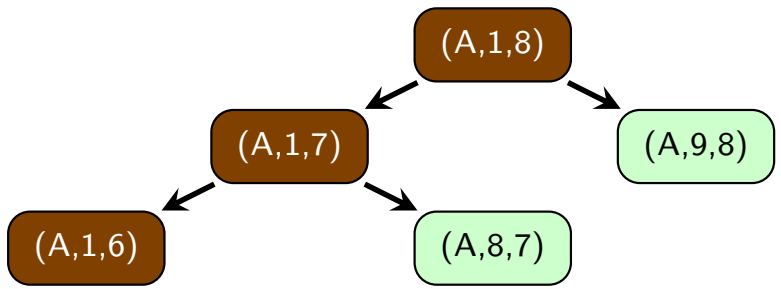
QuickSort$(A, 1, 6)$

Partition$(A, 1, 6)$

QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



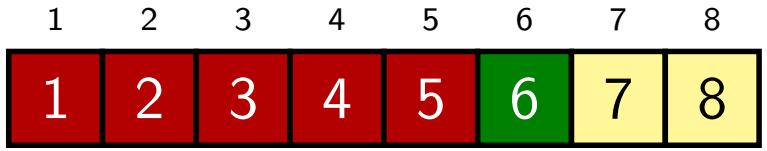


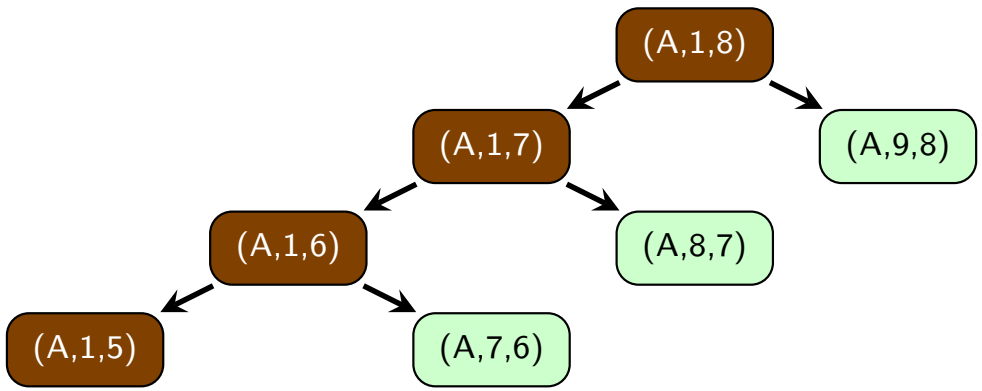
QuickSort$(A, 1, 6)$

Partition$(A, 1, 6)$

QuickSort: Worst-Case Behavior

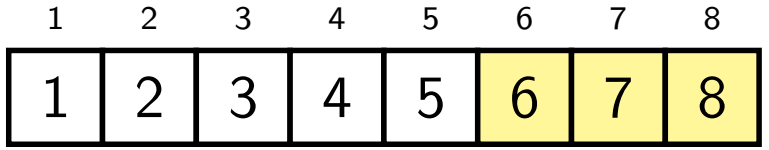
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

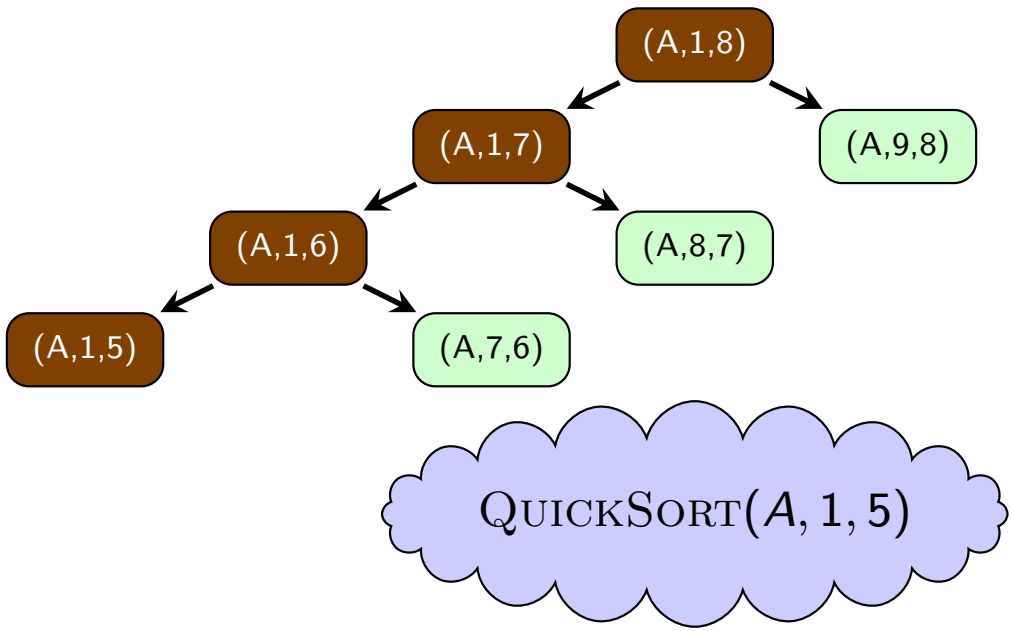




QuickSort: Worst-Case Behavior

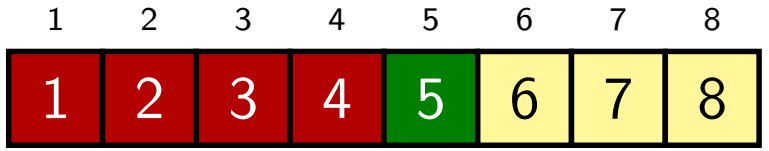
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

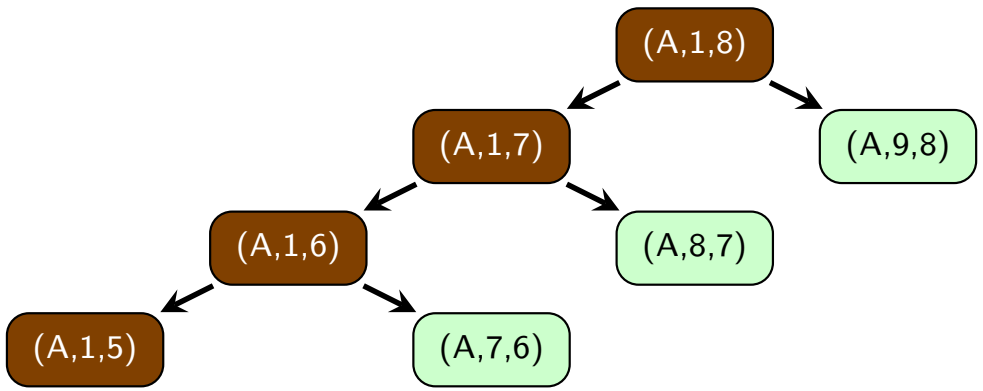




QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



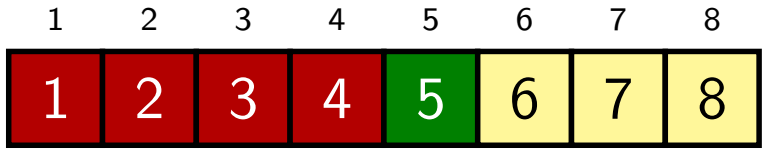


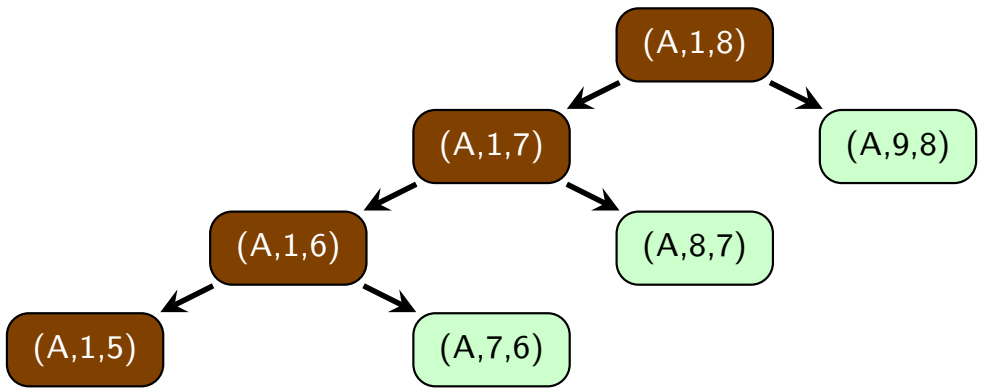
QuickSort$(A, 1, 5)$

Partition$(A, 1, 5)$

QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



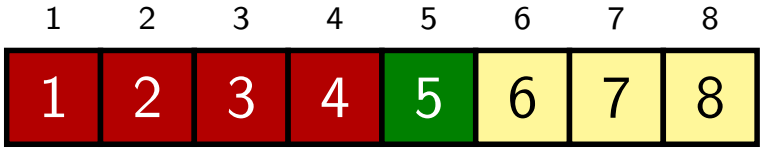


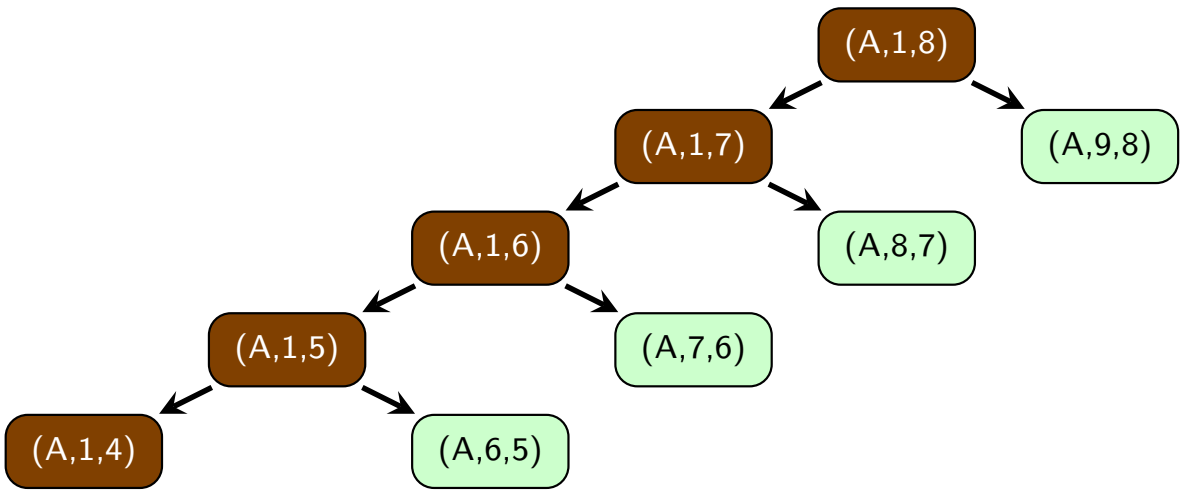
QuickSort$(A, 1, 5)$

Partition$(A, 1, 5)$

QuickSort: Worst-Case Behavior

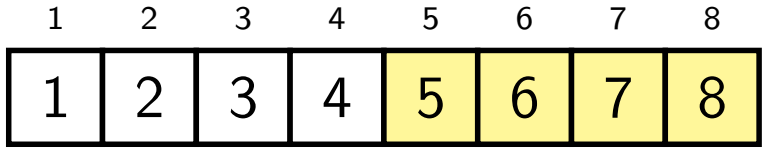
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

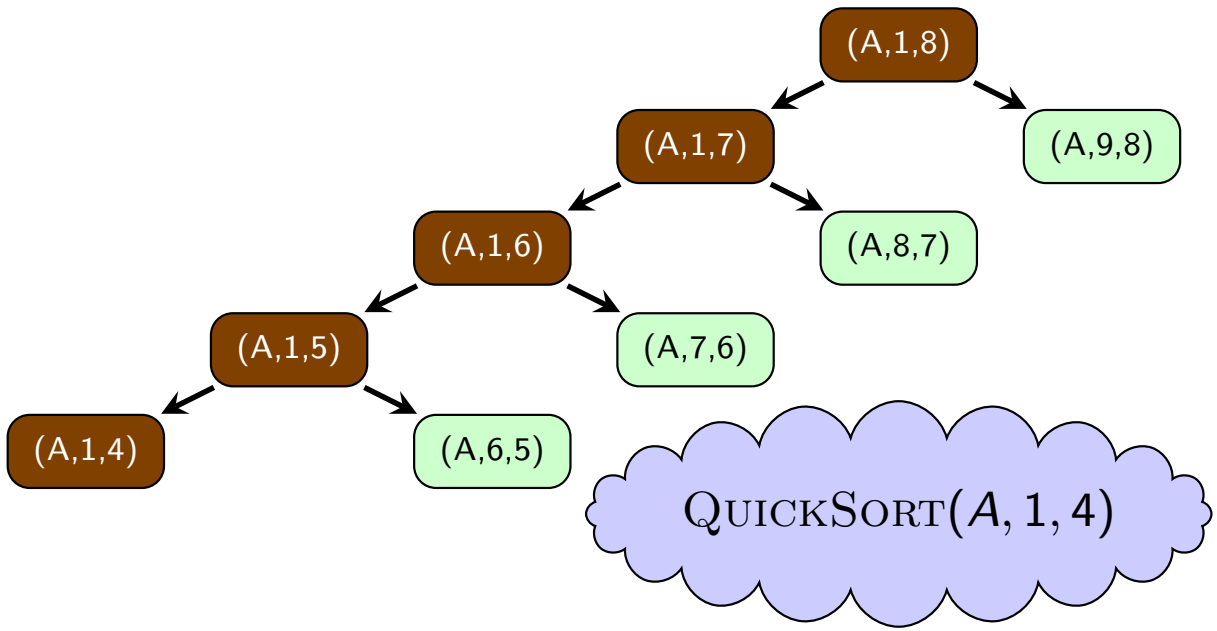




QuickSort: Worst-Case Behavior

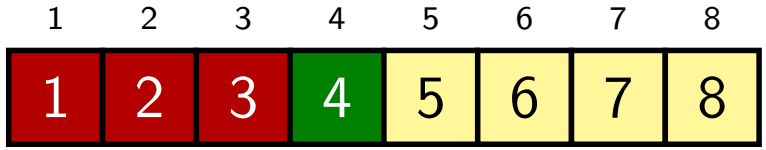
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

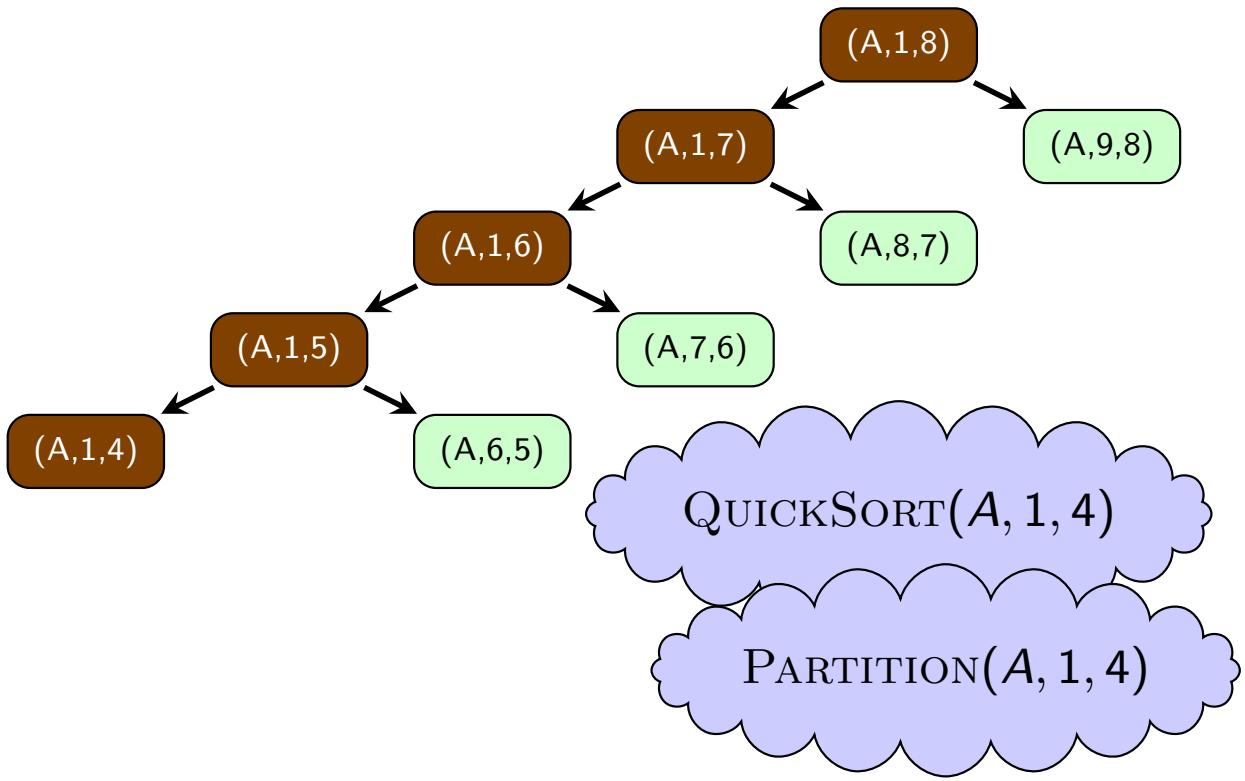




QuickSort: Worst-Case Behavior

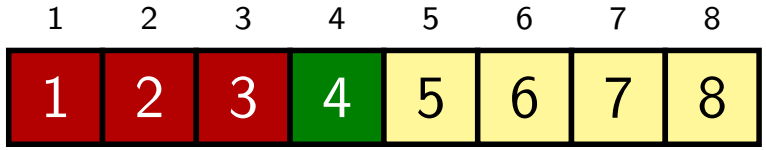
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

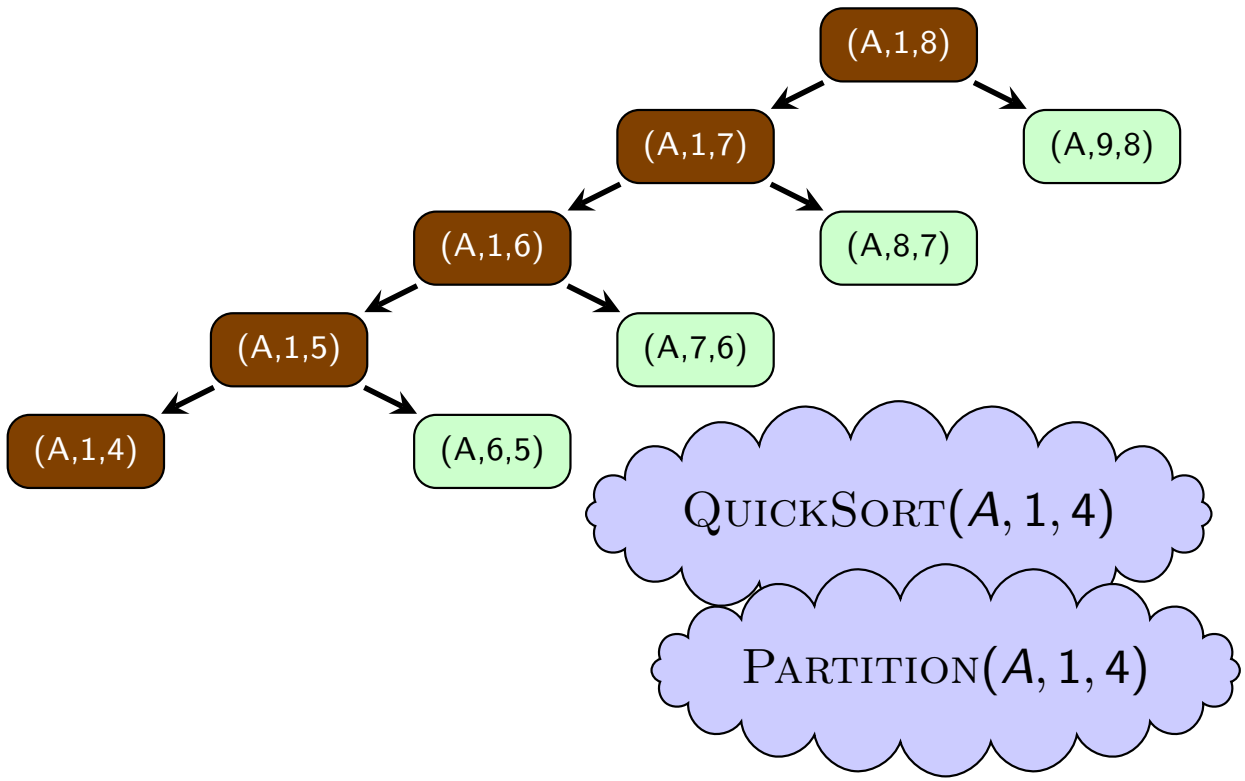




QuickSort: Worst-Case Behavior

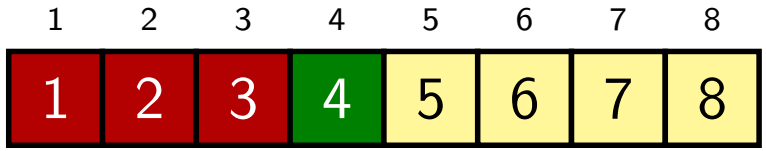
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

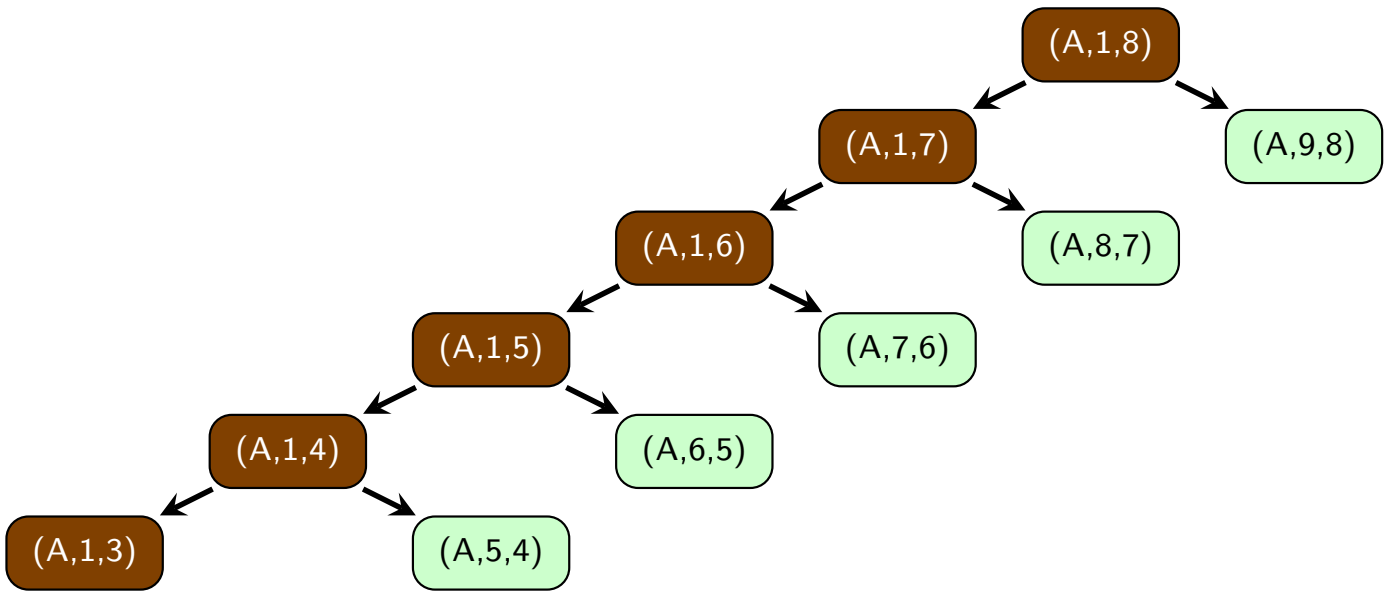




QuickSort: Worst-Case Behavior

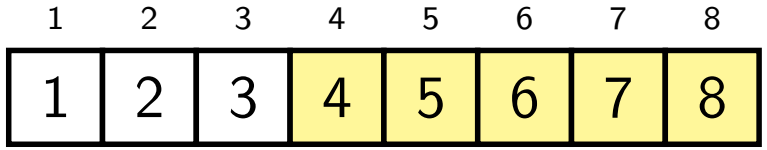
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

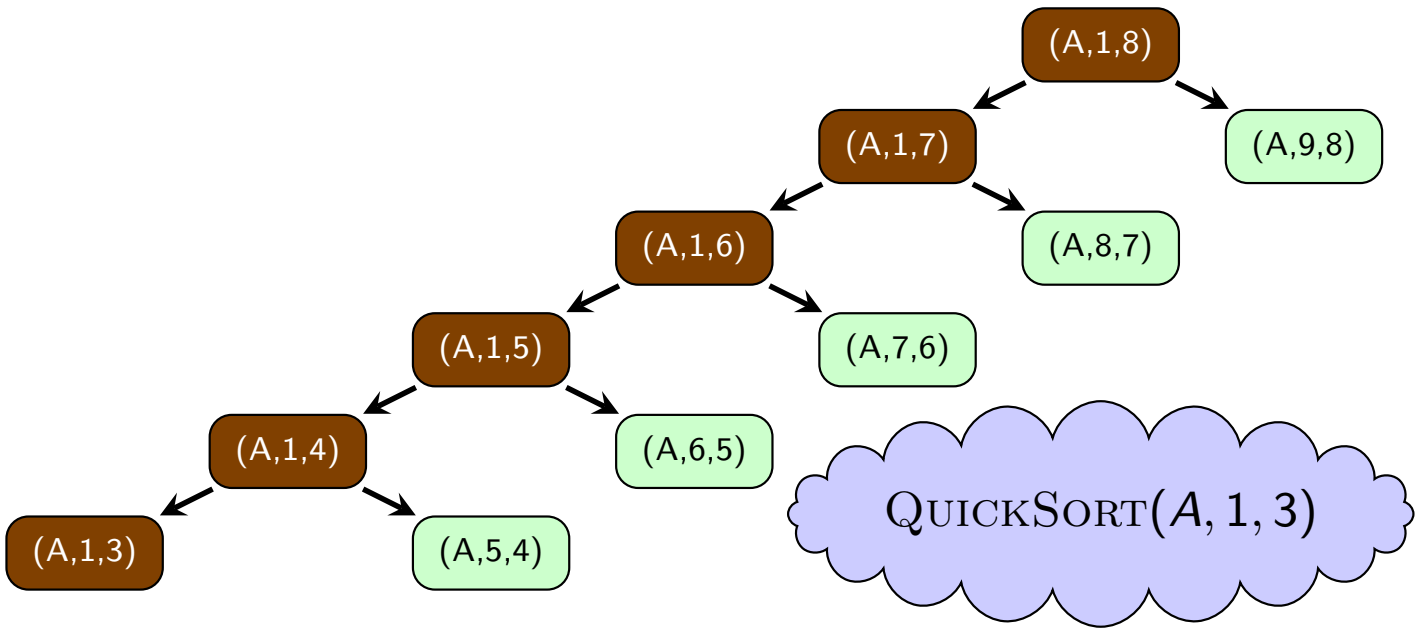




QuickSort: Worst-Case Behavior

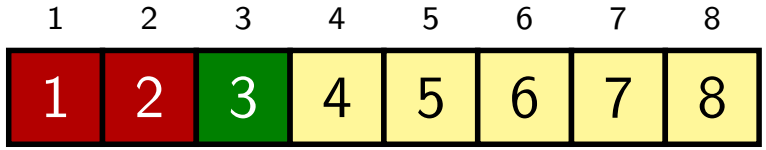
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

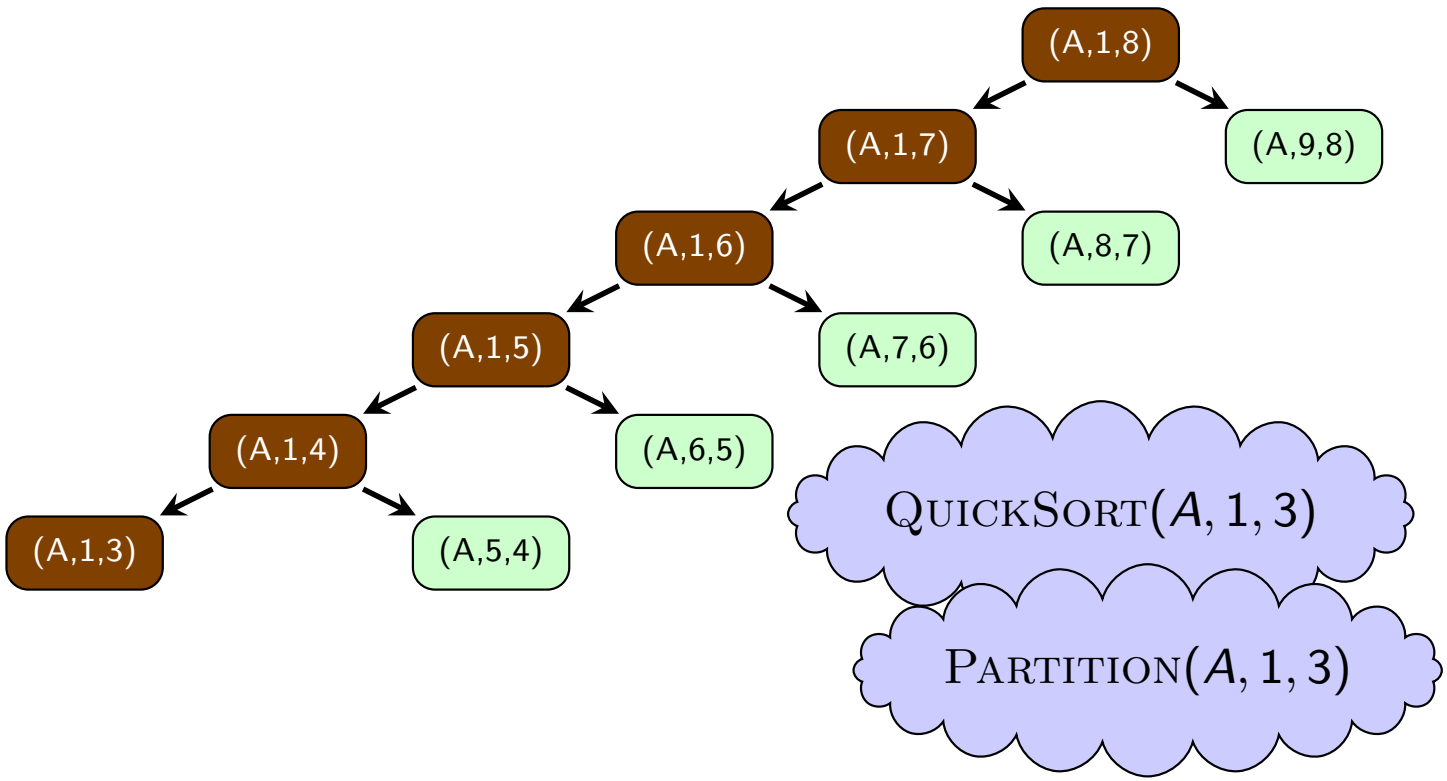




QuickSort: Worst-Case Behavior

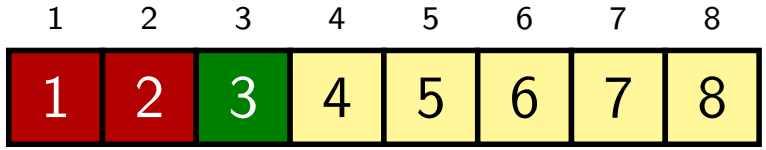
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

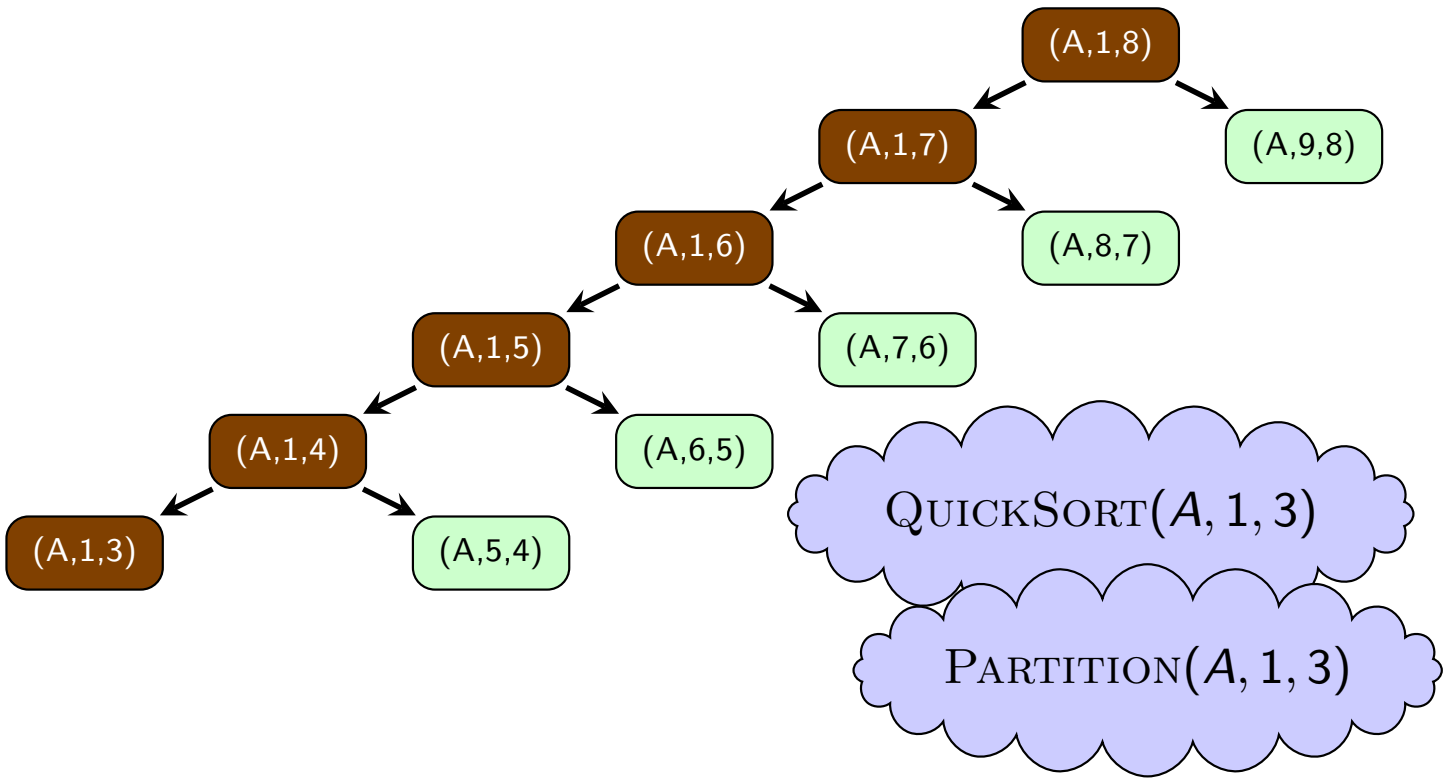




QuickSort: Worst-Case Behavior

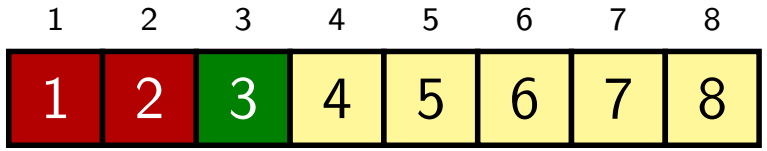
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

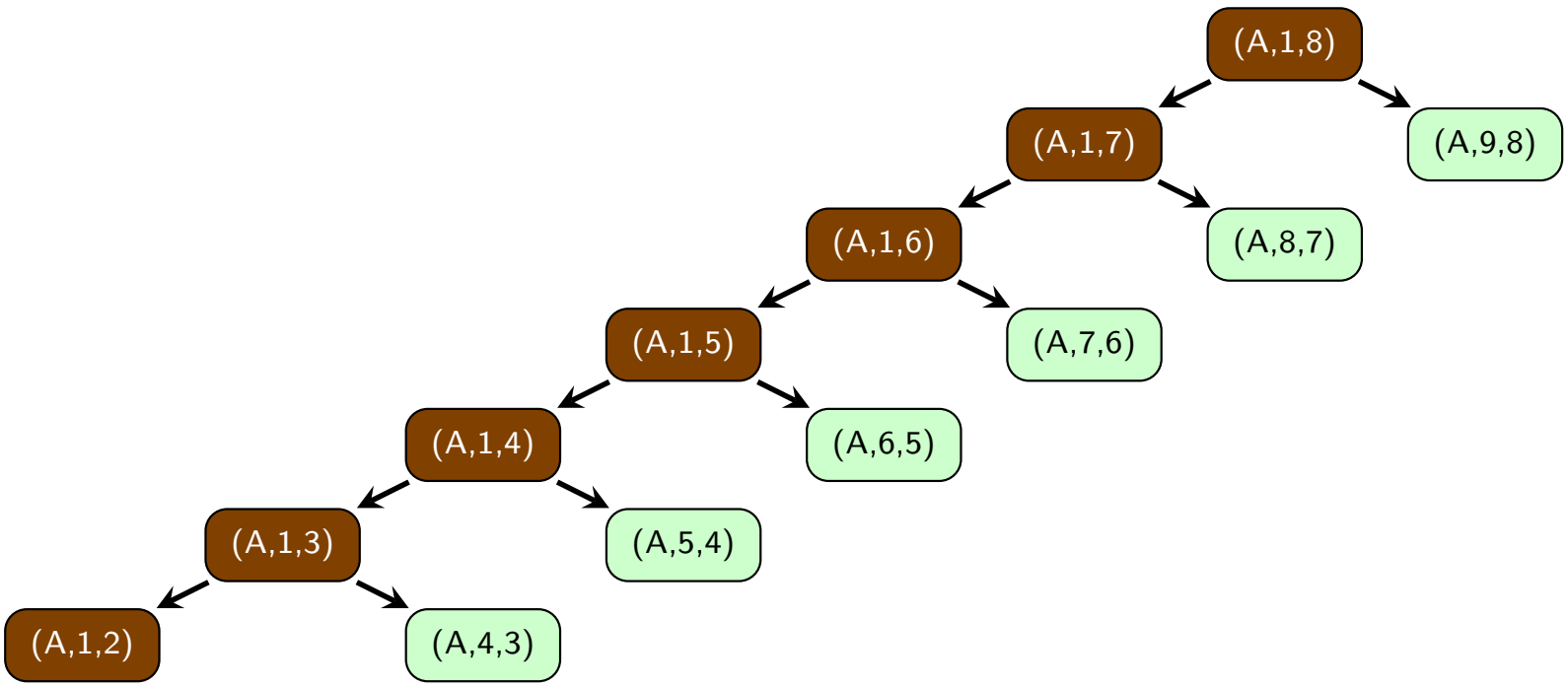




QuickSort: Worst-Case Behavior

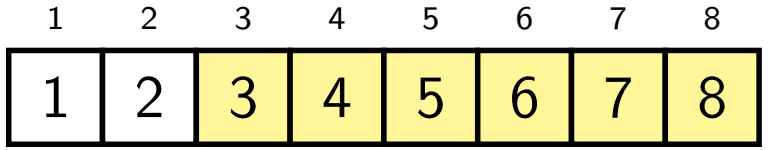
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

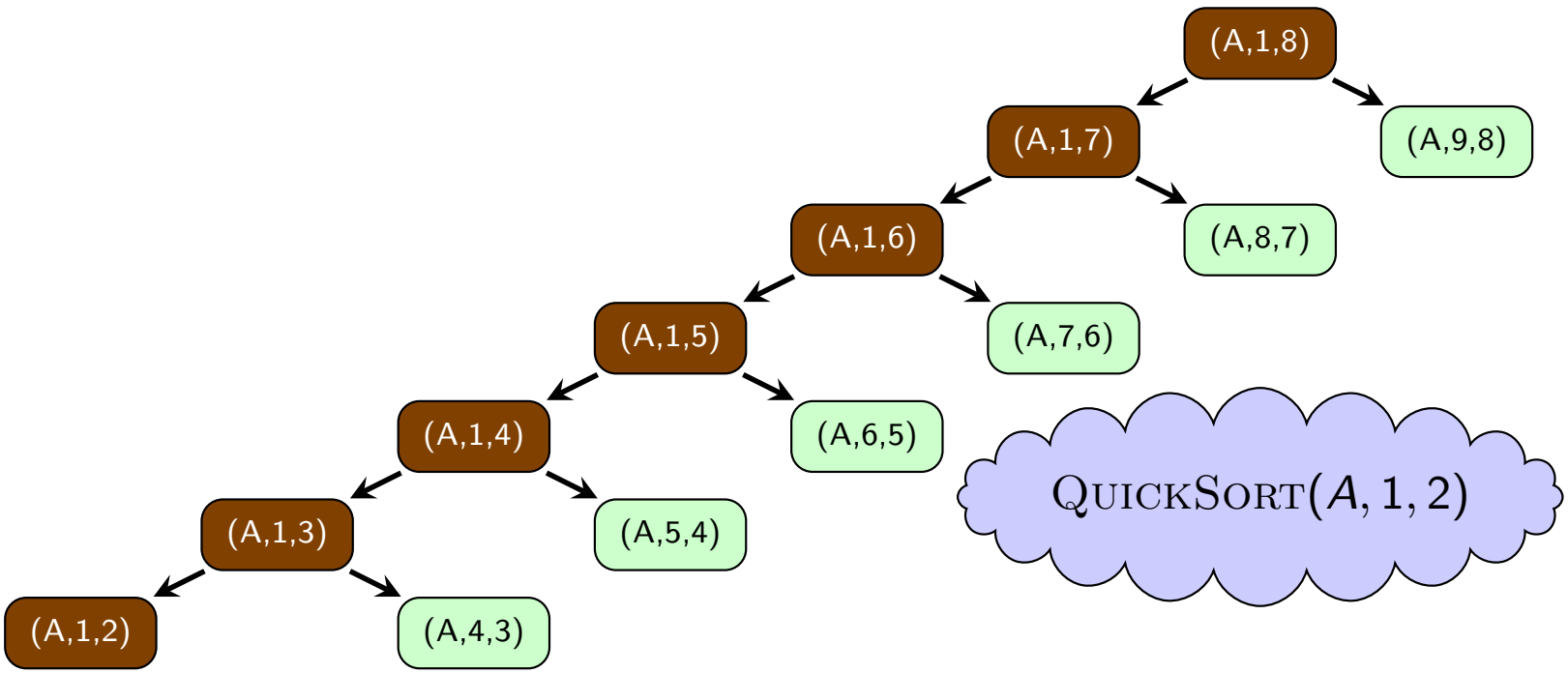




QuickSort: Worst-Case Behavior

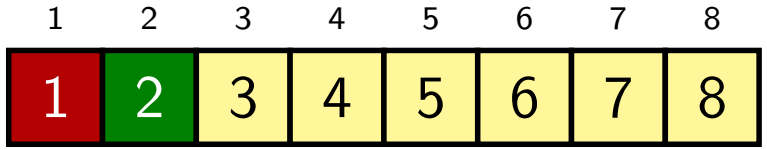
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

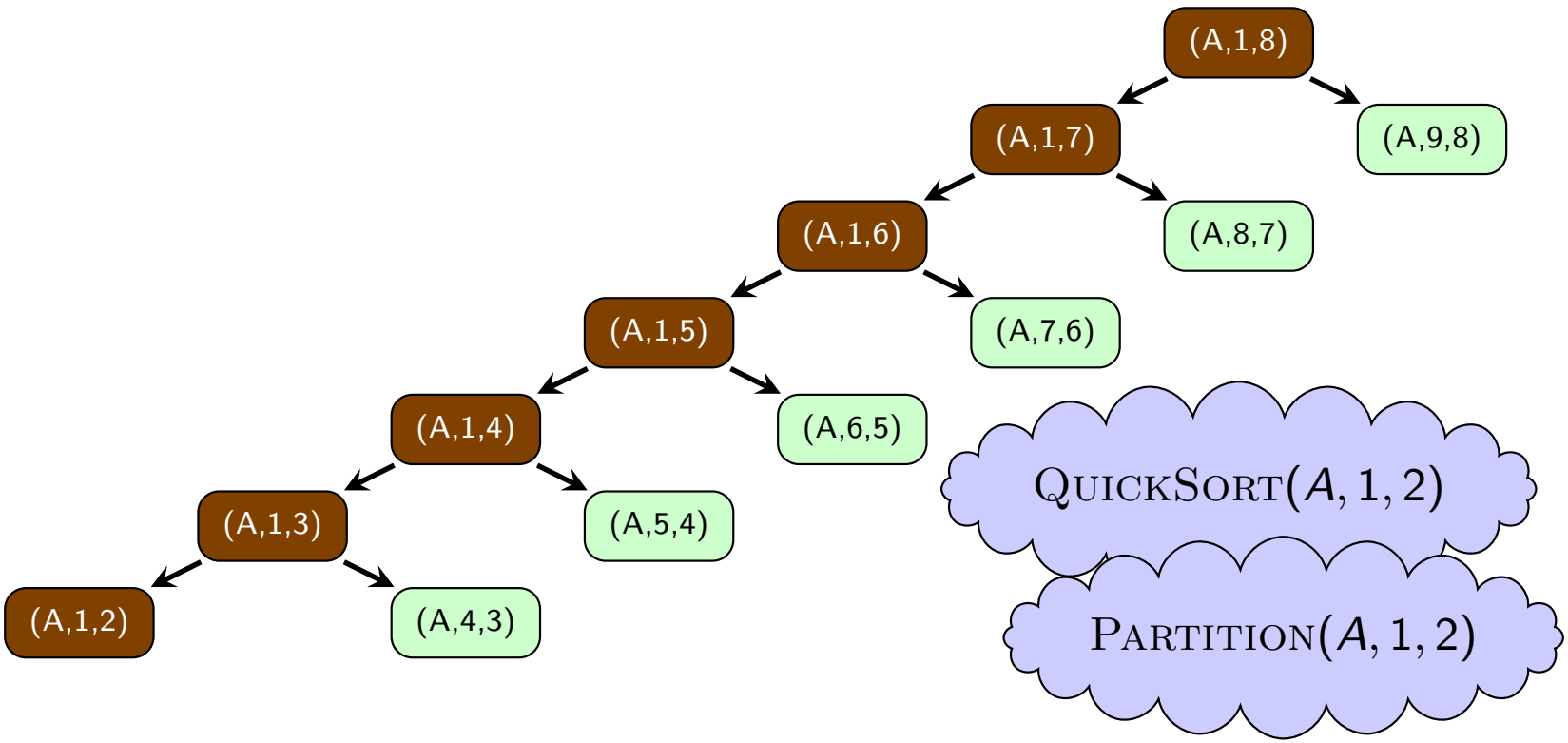




QuickSort: Worst-Case Behavior

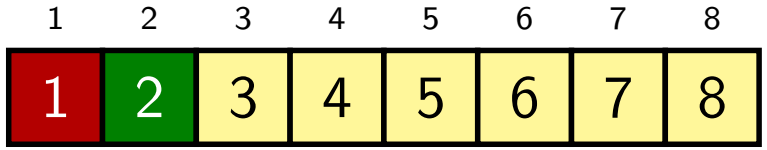
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

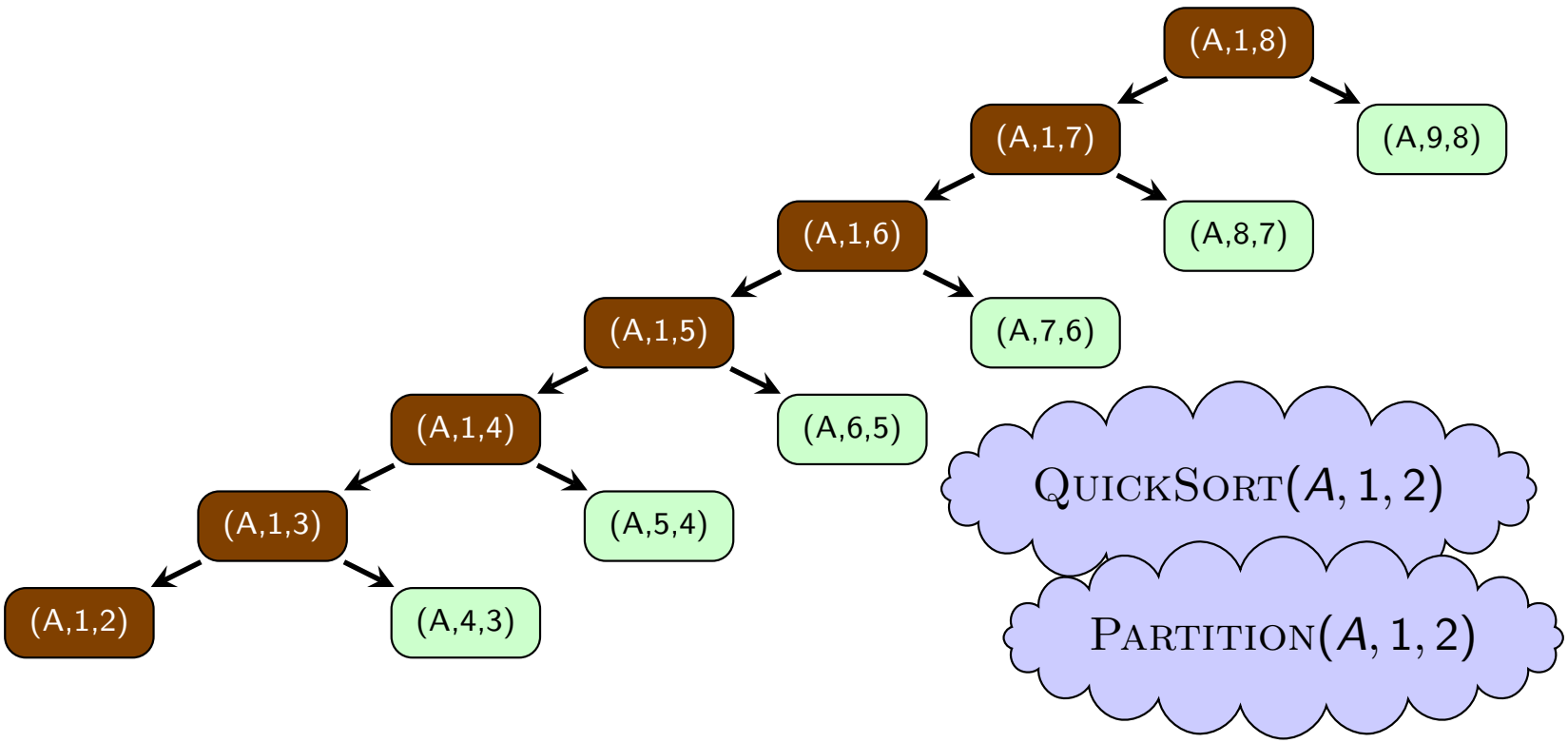




QuickSort: Worst-Case Behavior

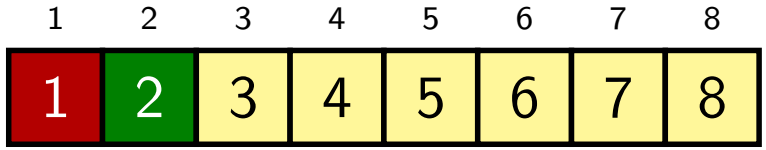
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

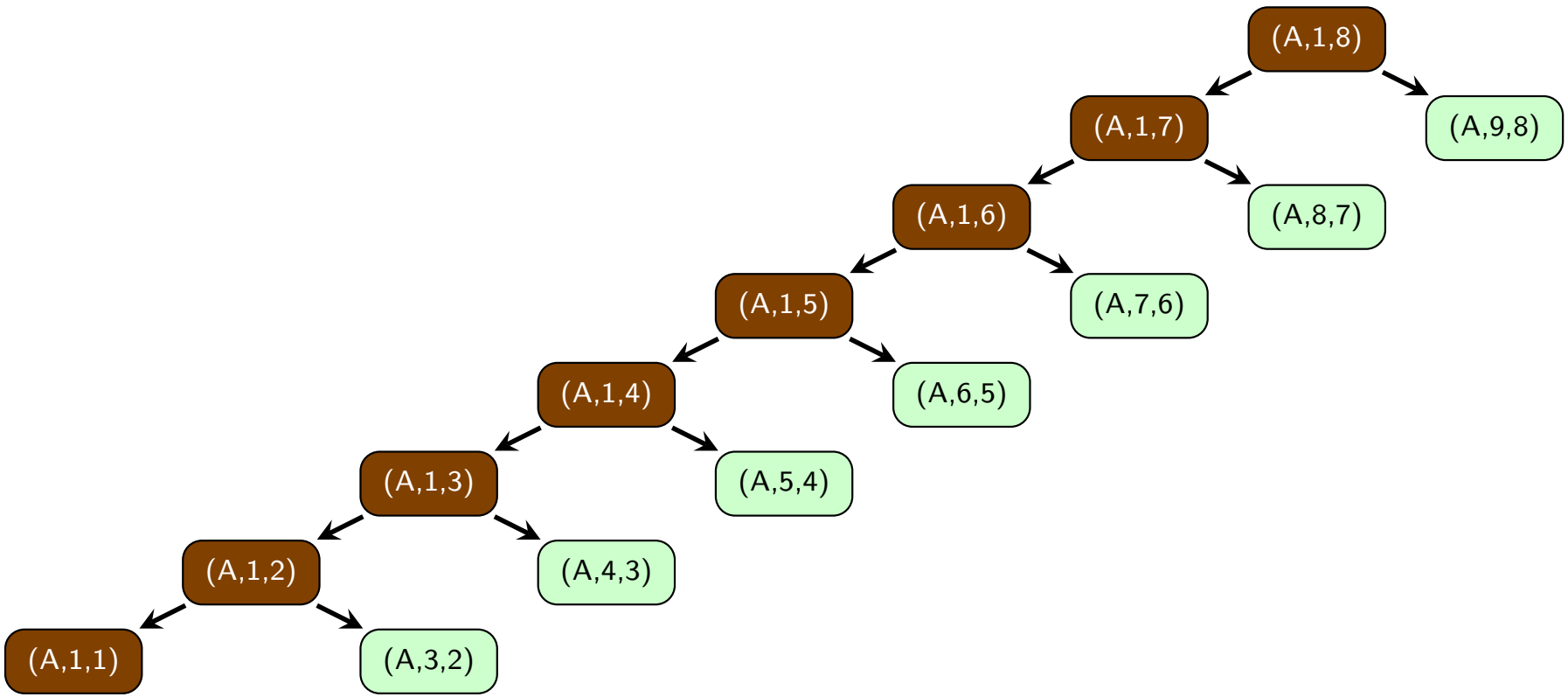




QuickSort: Worst-Case Behavior

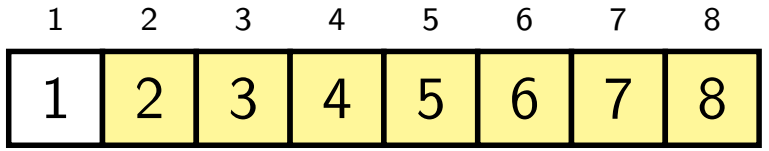
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

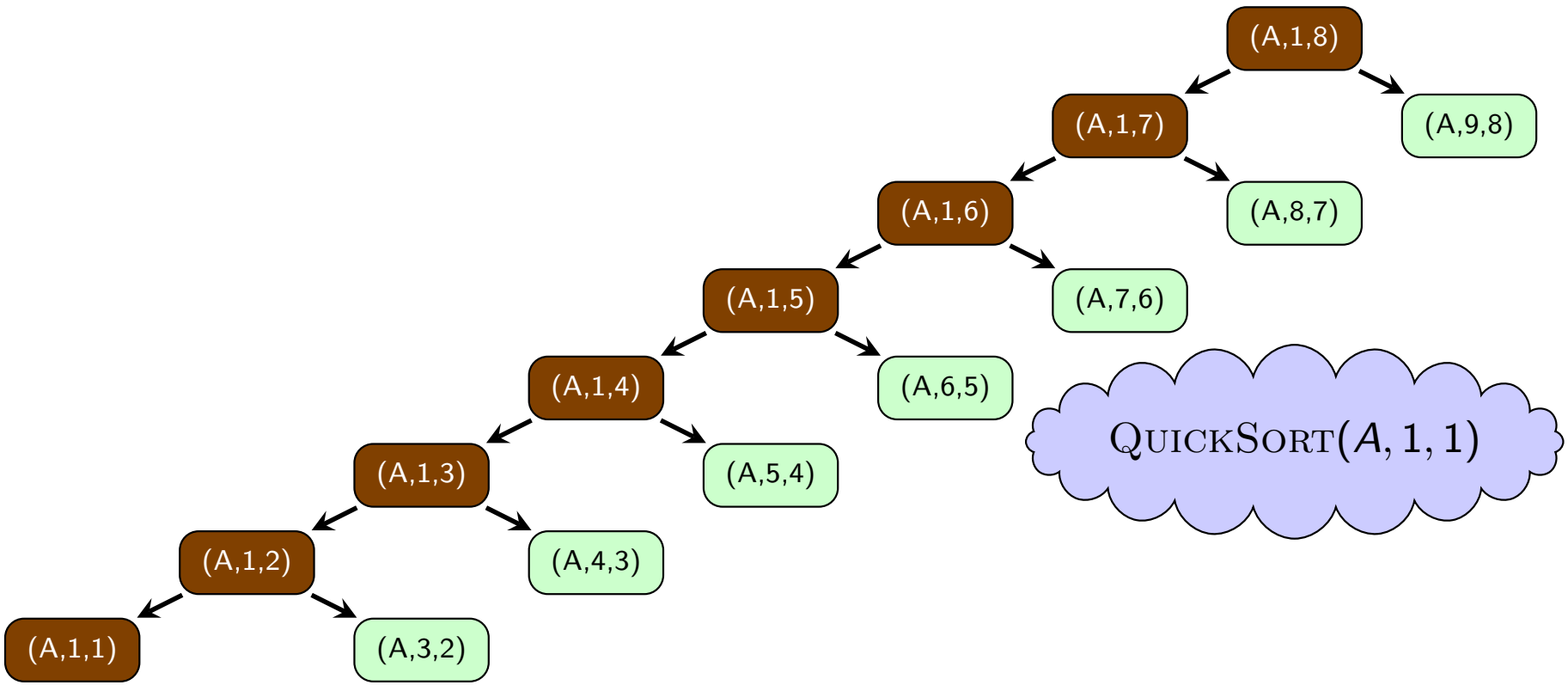




QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

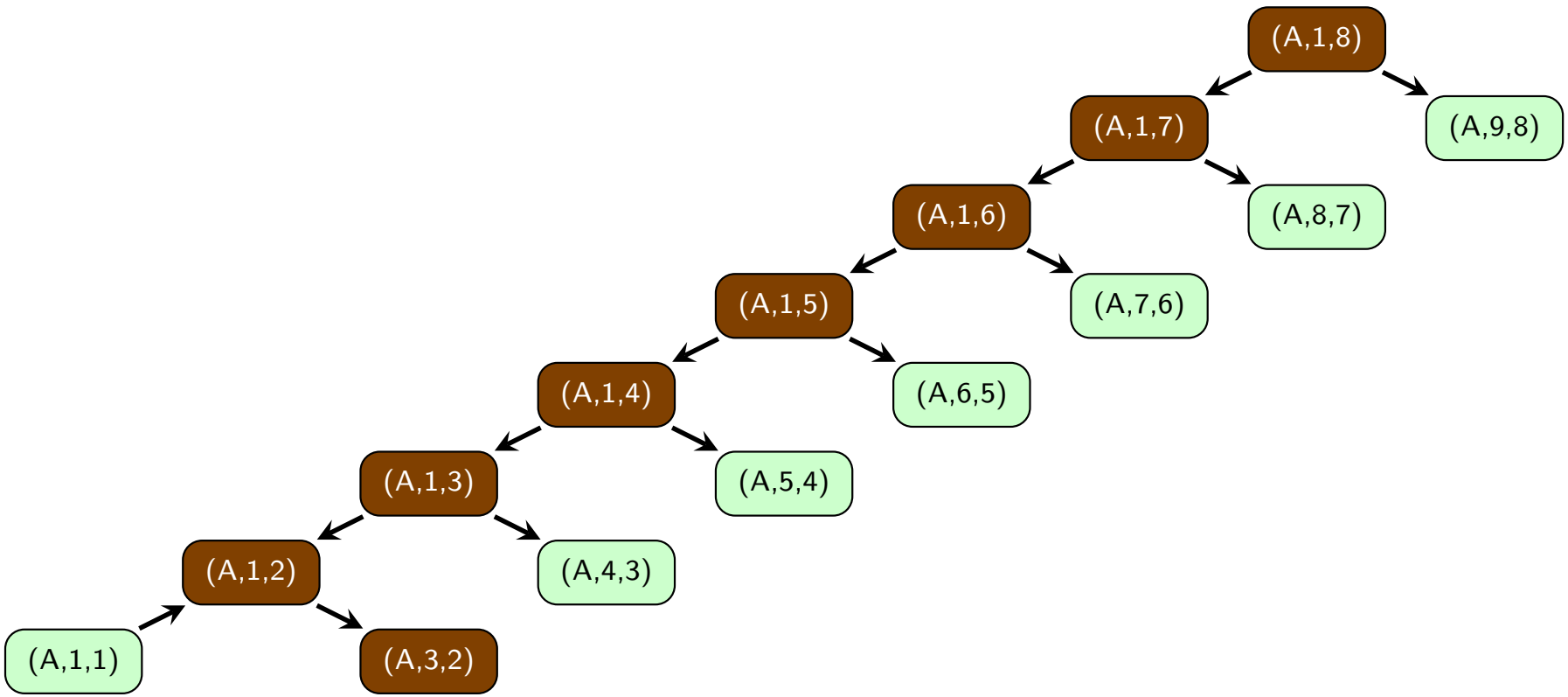




QuickSort: Worst-Case Behavior

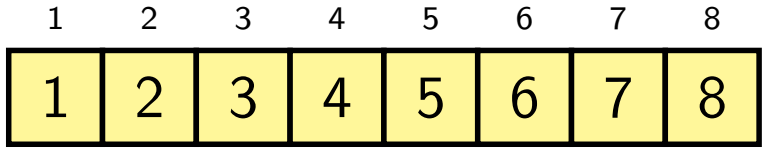
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

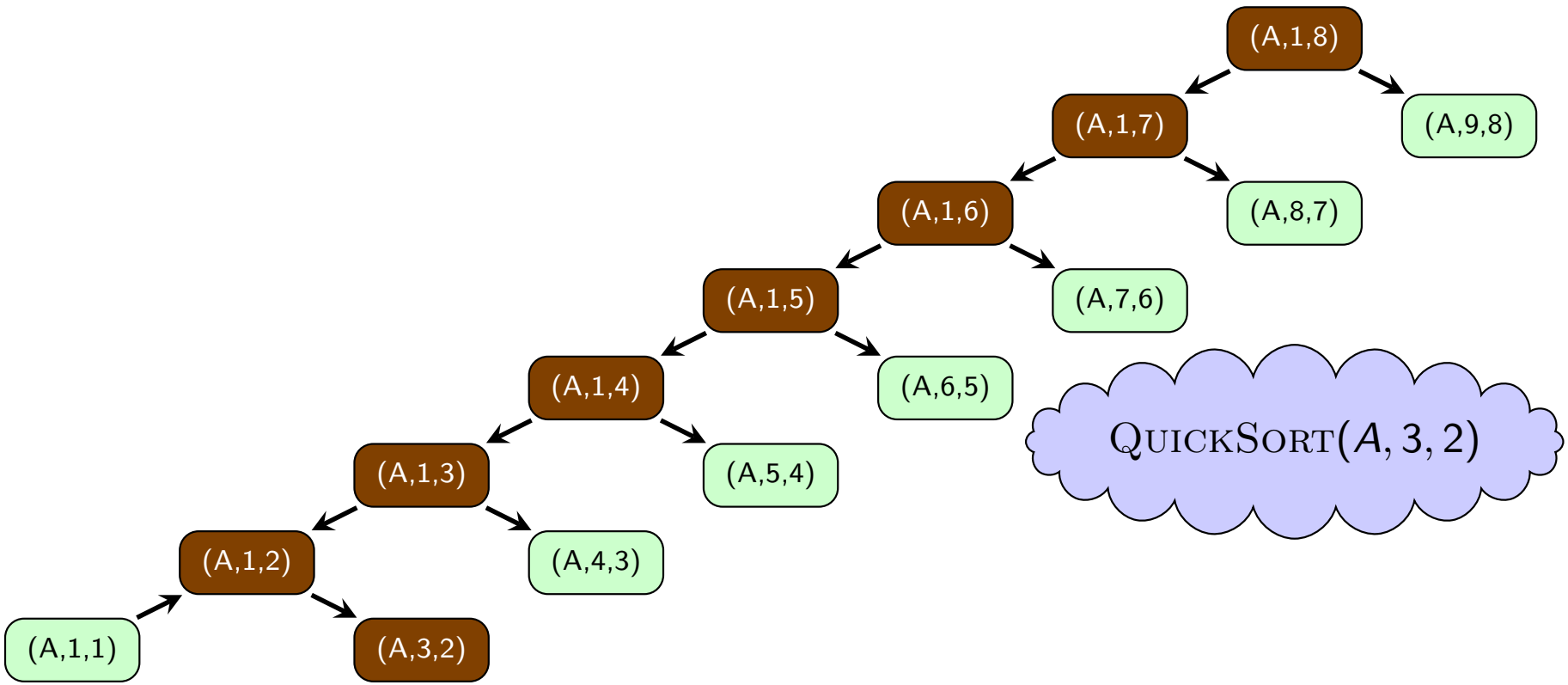
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

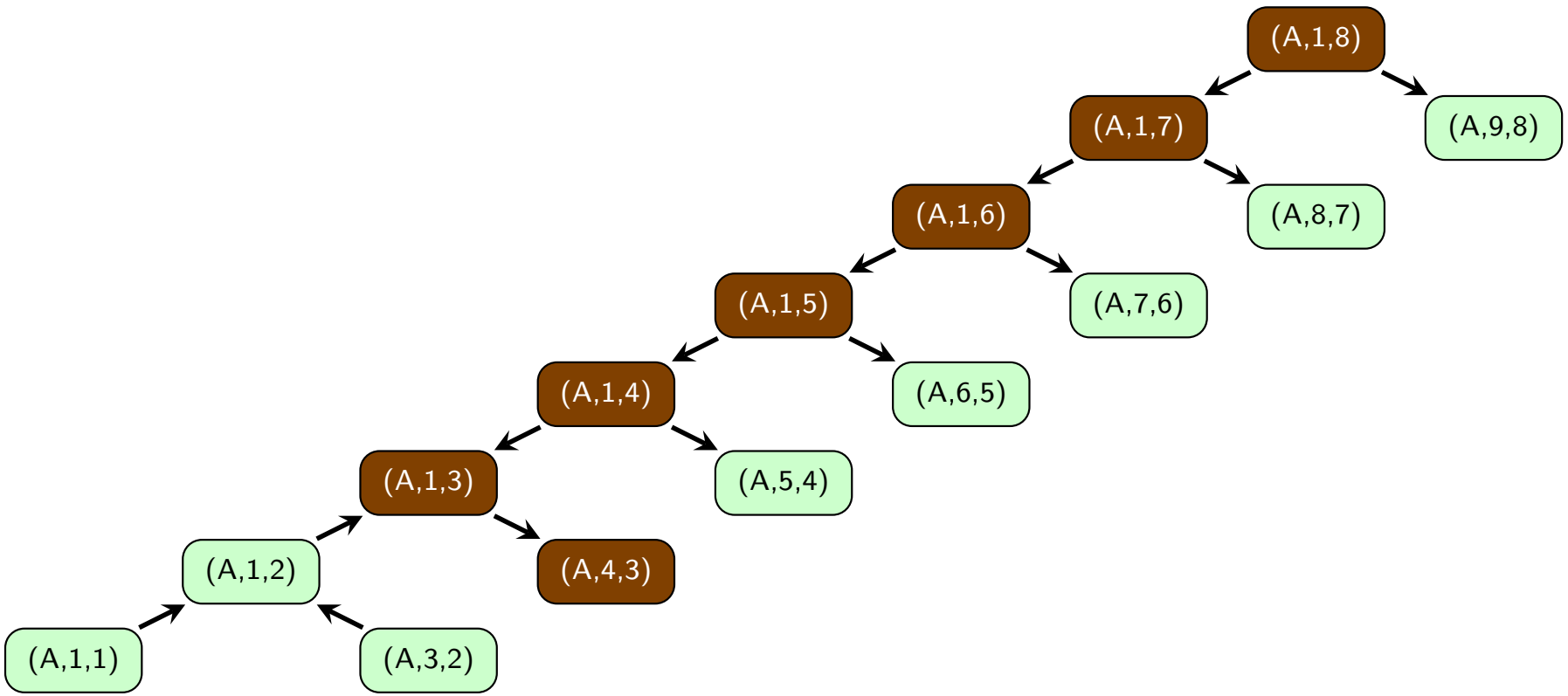




QuickSort: Worst-Case Behavior

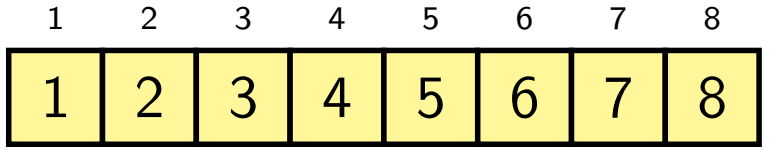
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

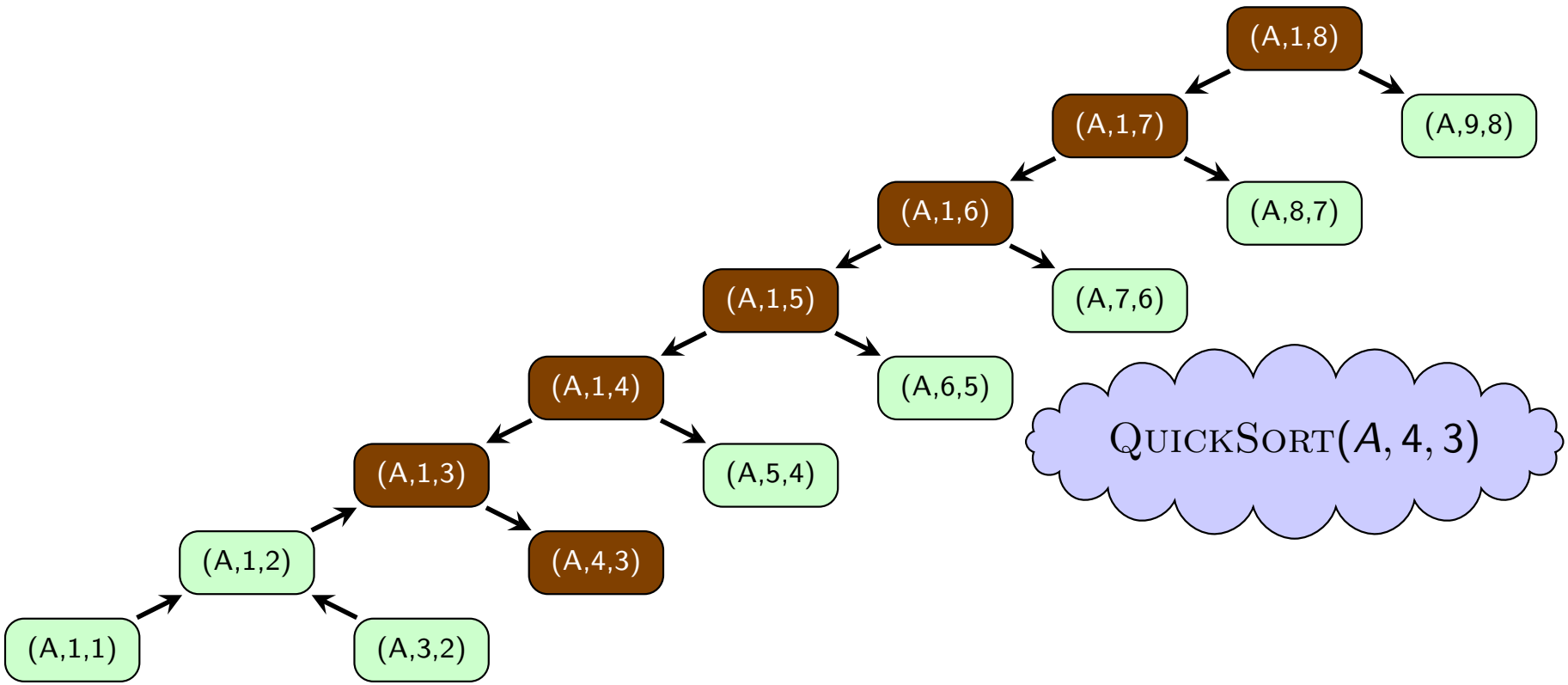
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

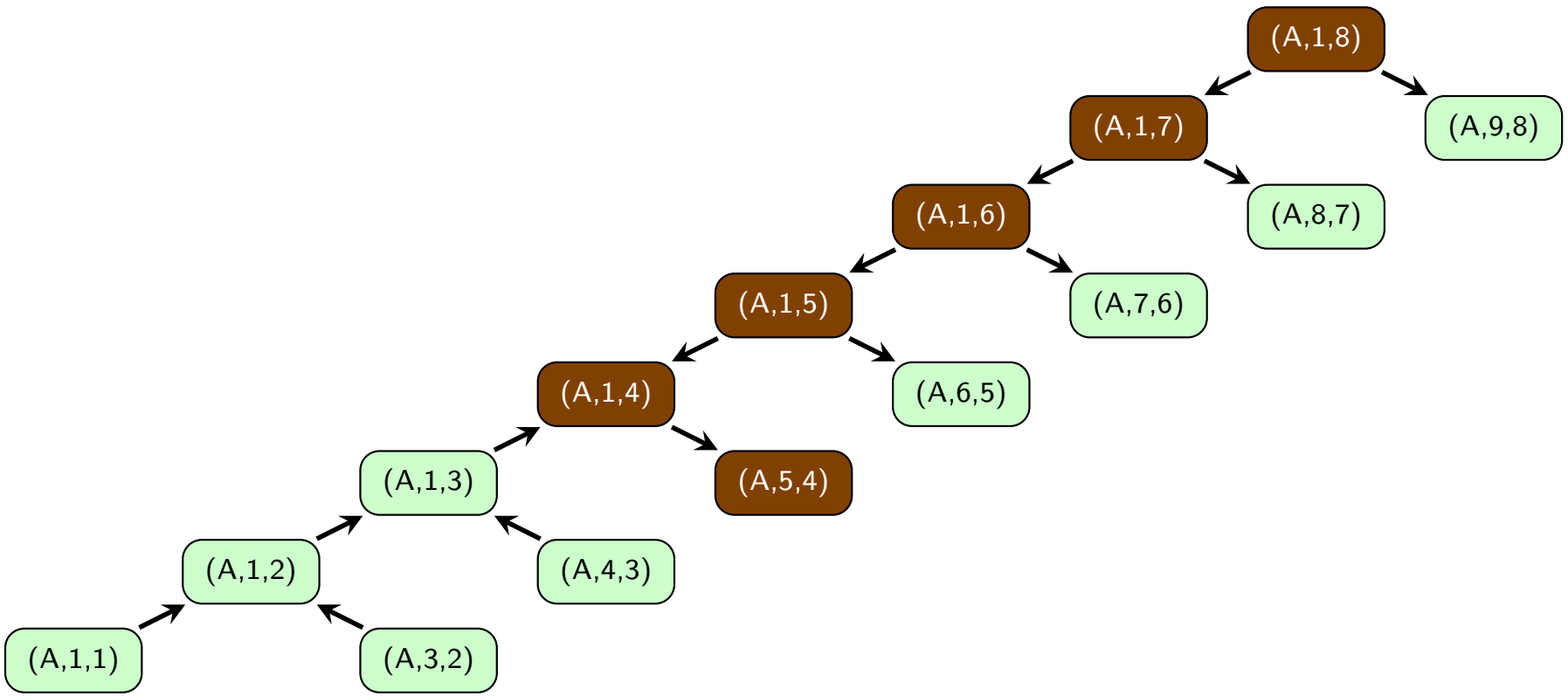




QuickSort: Worst-Case Behavior

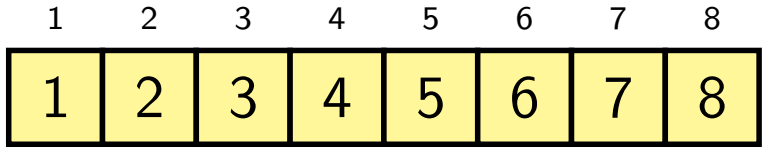
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

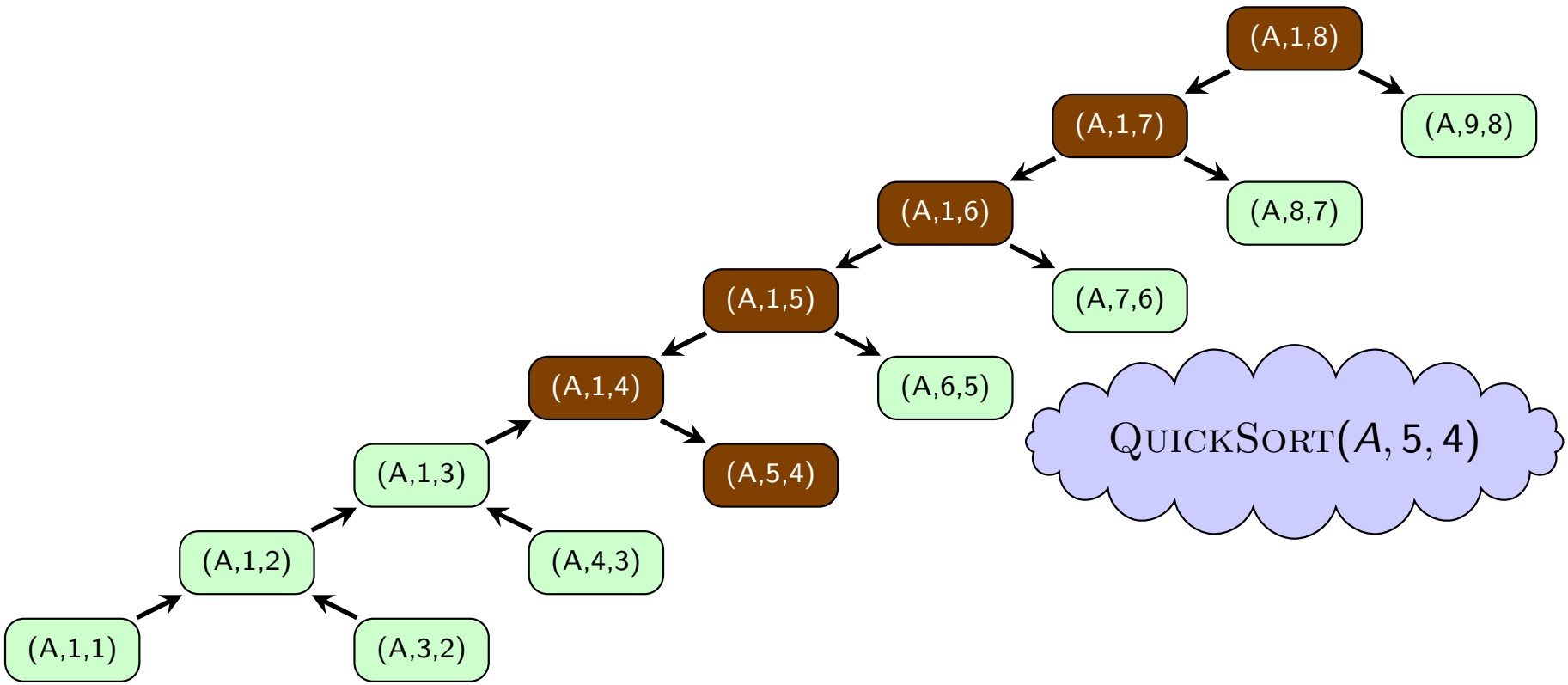
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

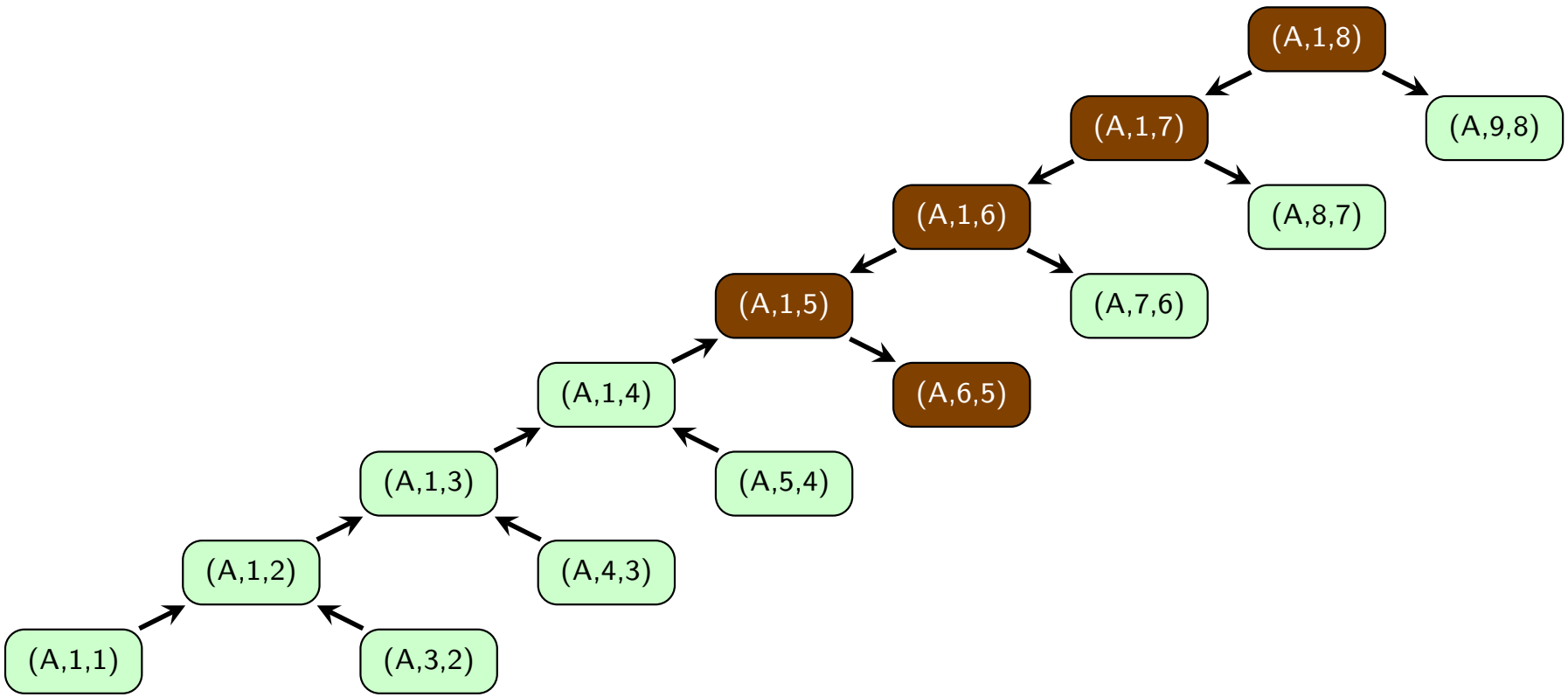




QuickSort: Worst-Case Behavior

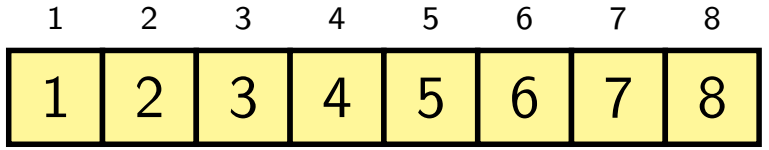
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

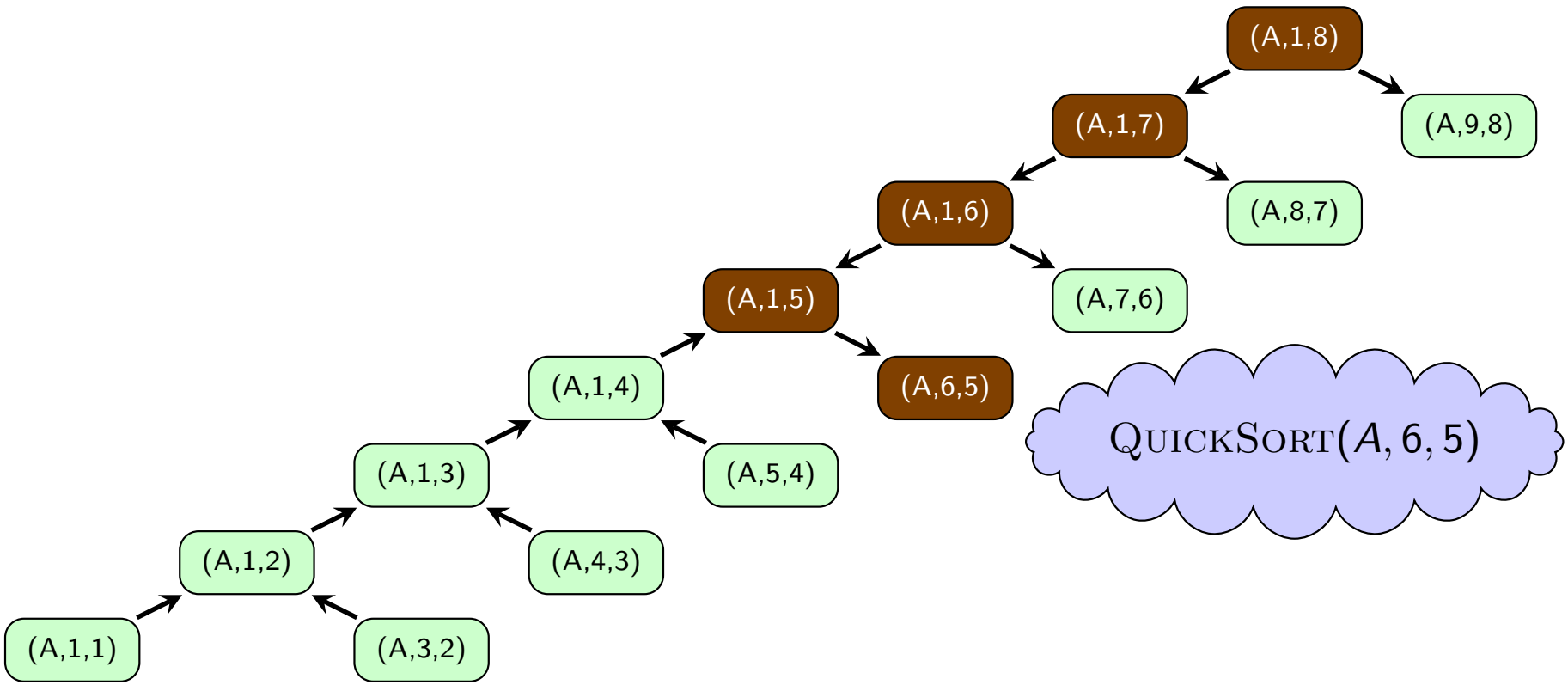
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

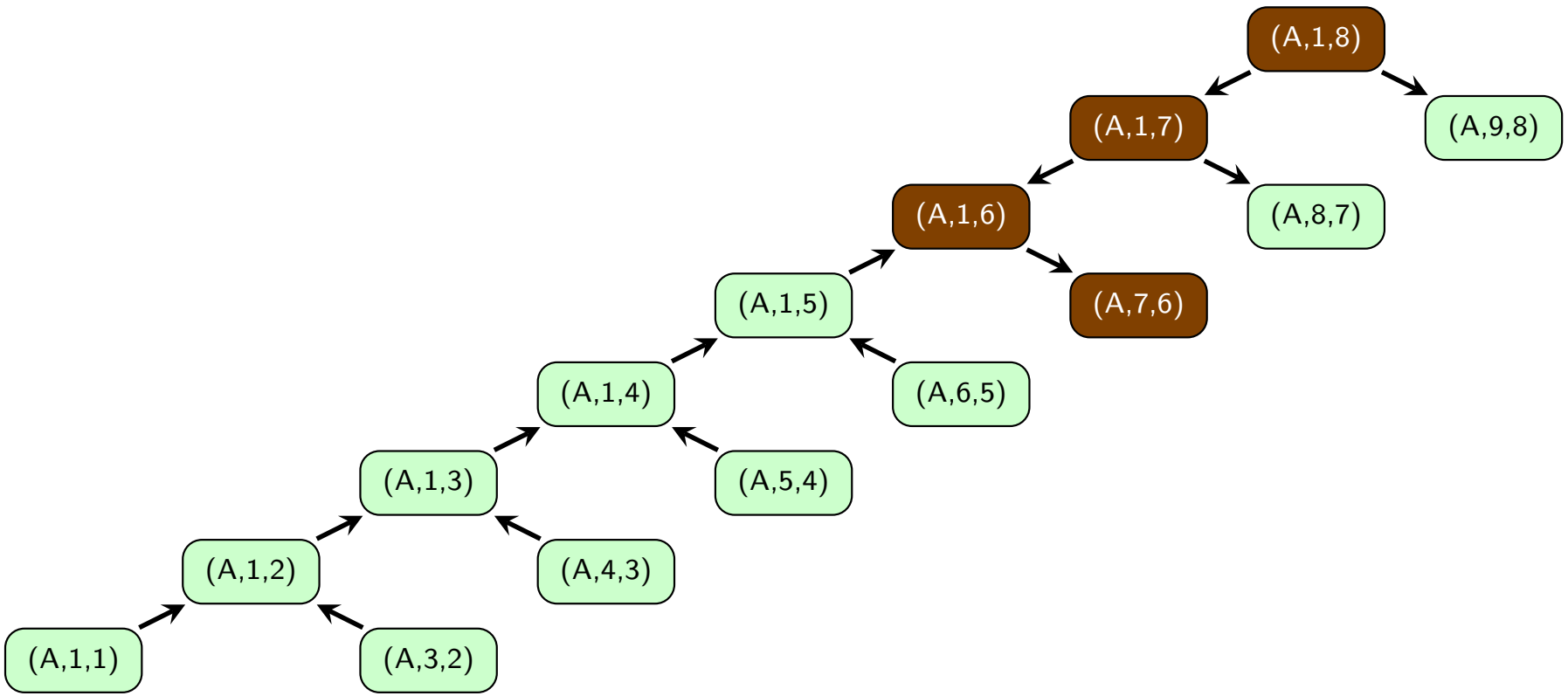




QuickSort: Worst-Case Behavior

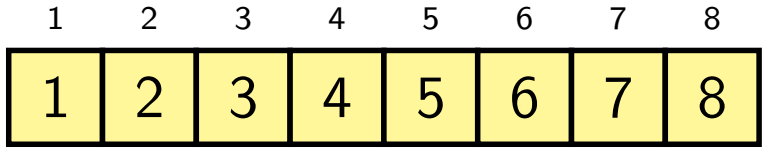
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

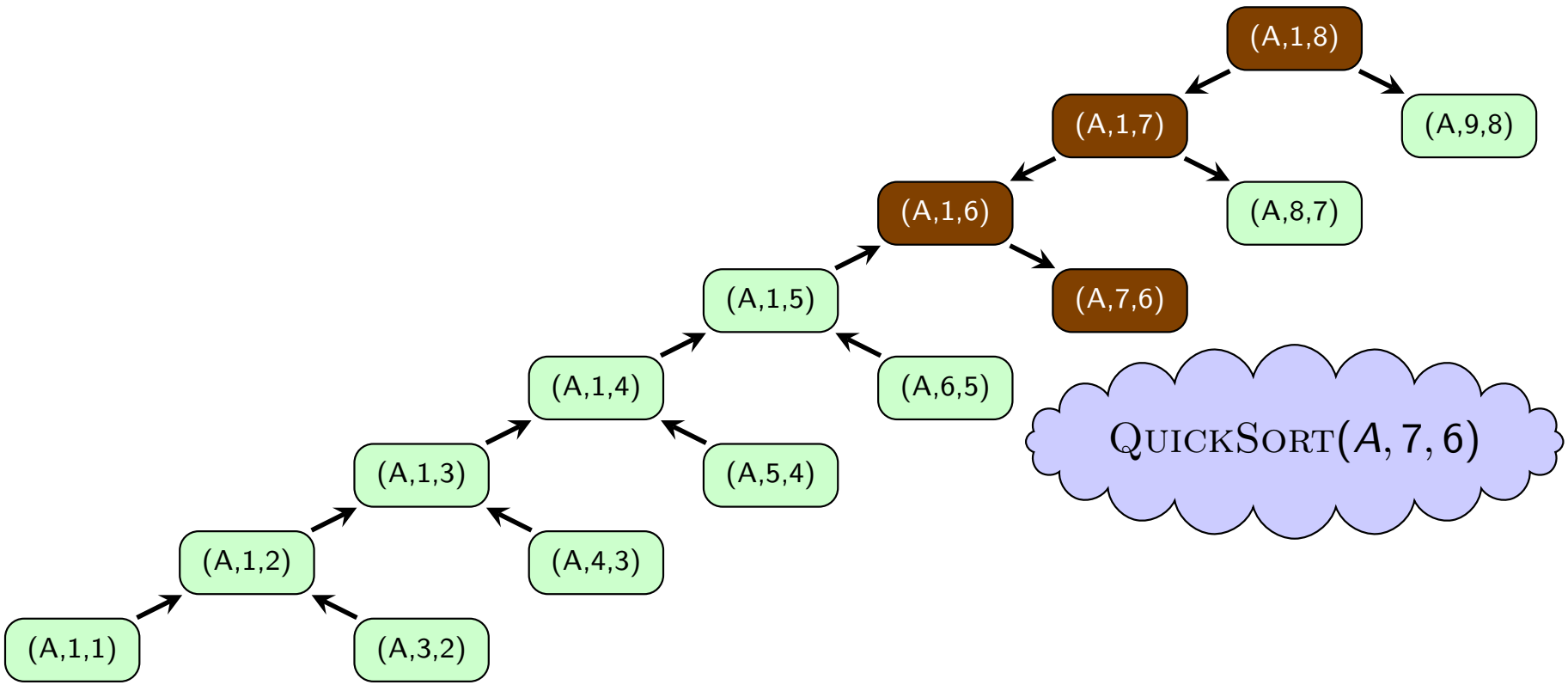
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

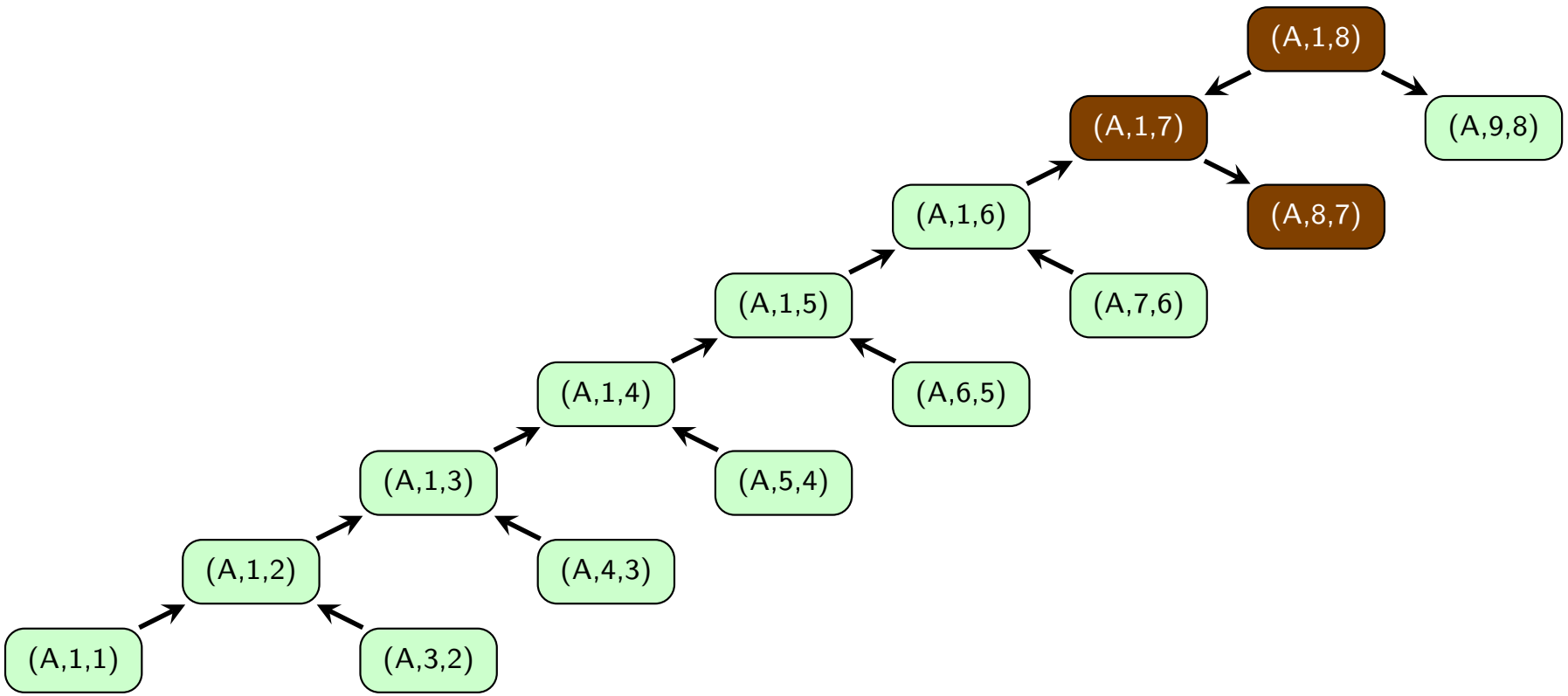




QuickSort: Worst-Case Behavior

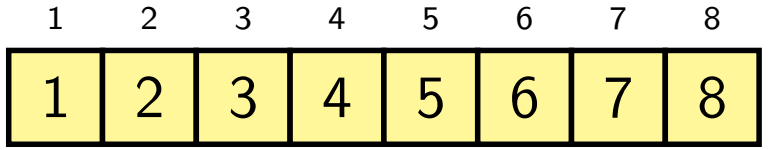
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

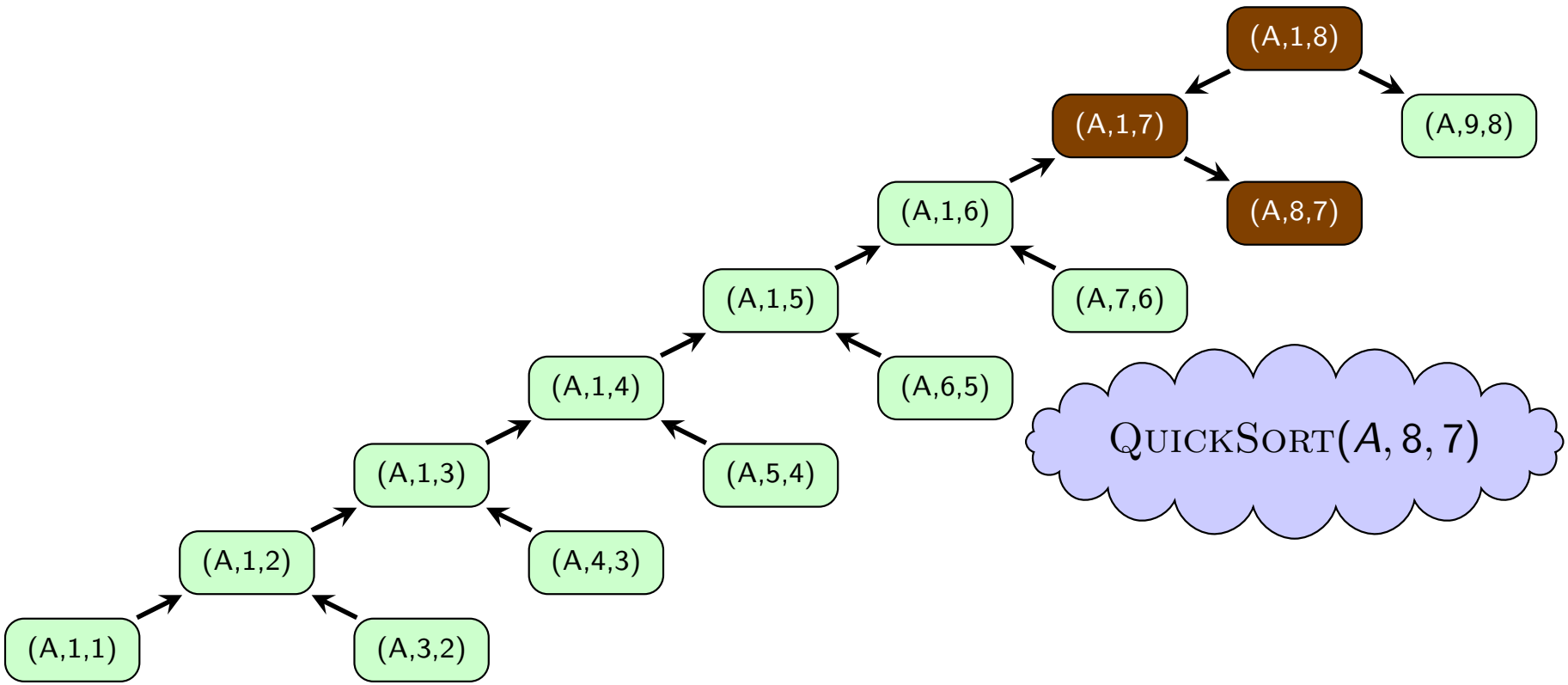
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

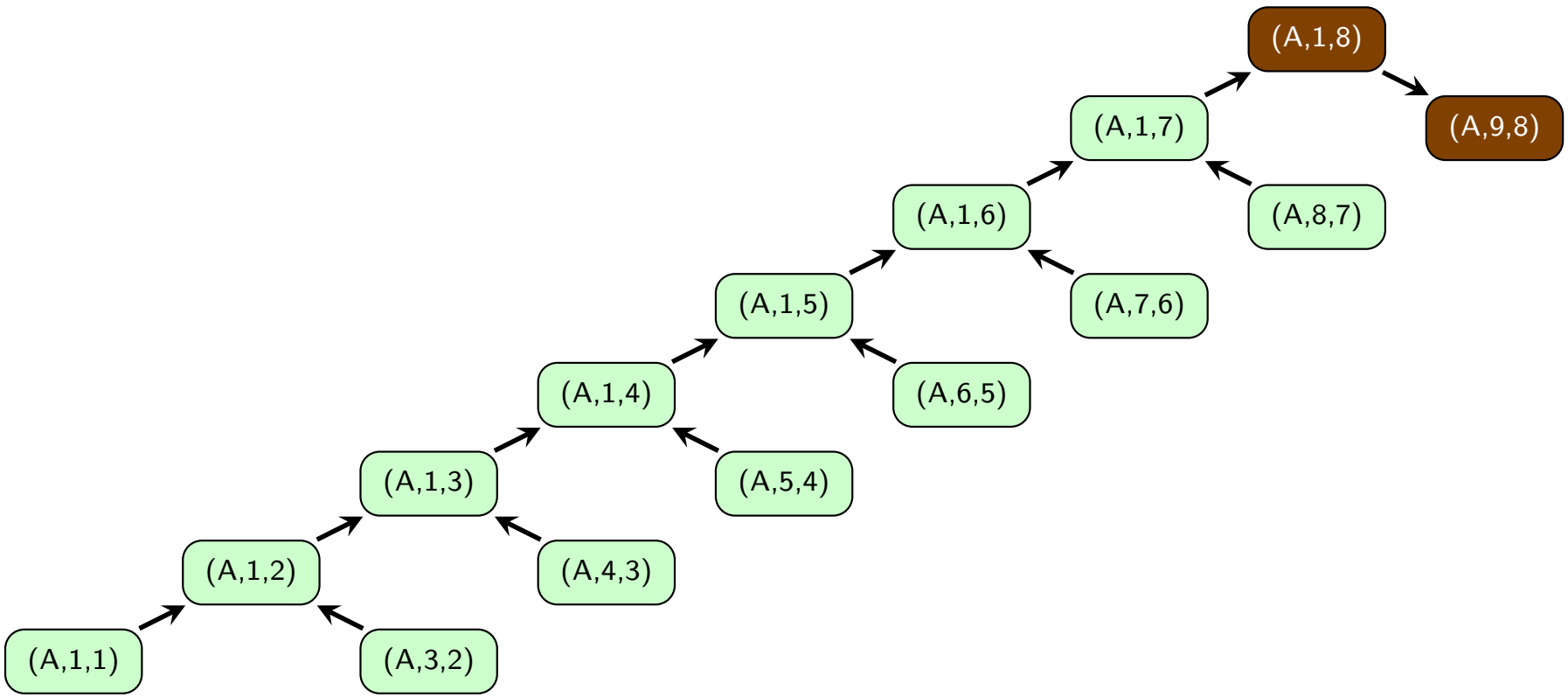




QuickSort: Worst-Case Behavior

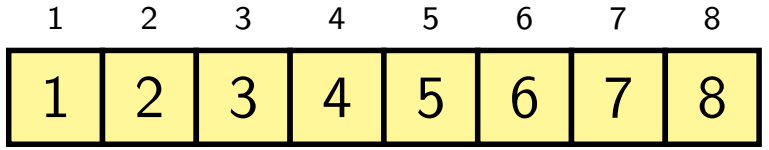
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

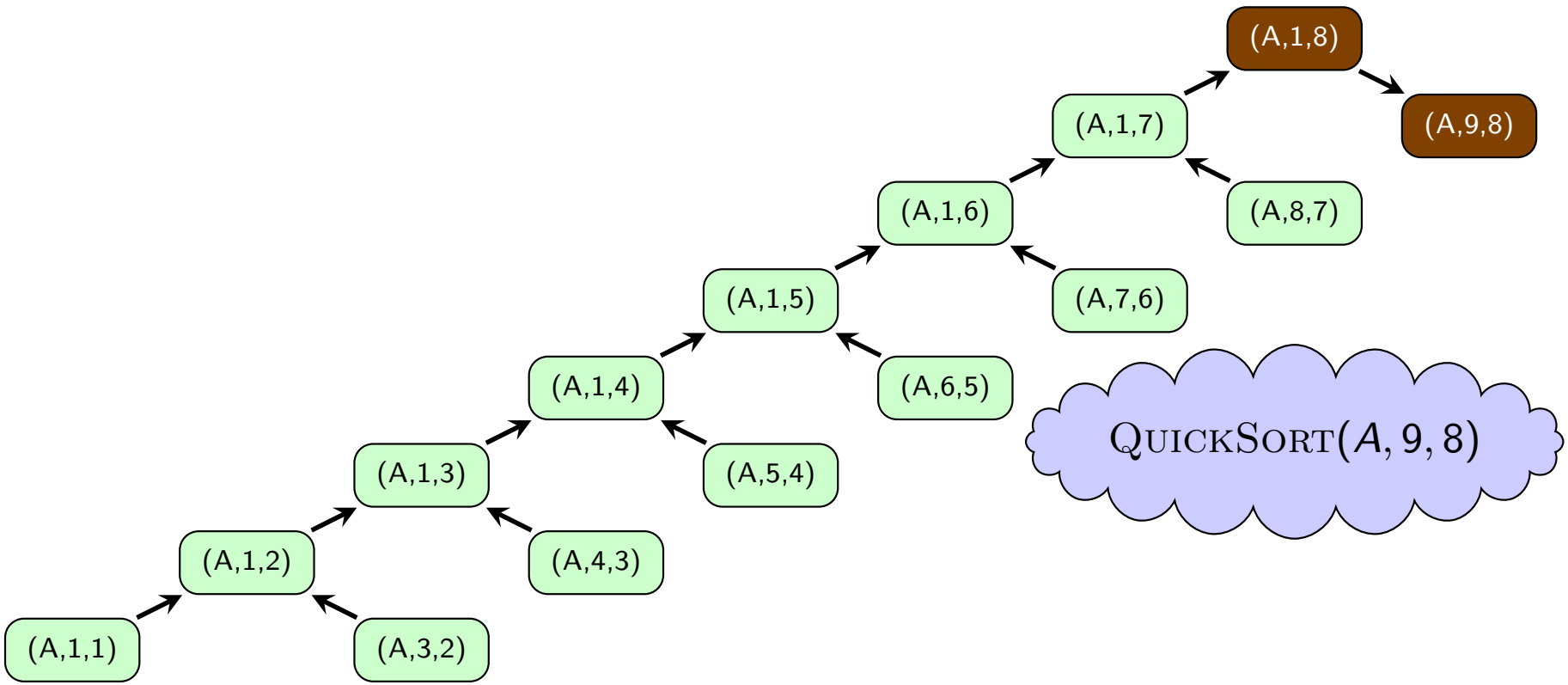
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

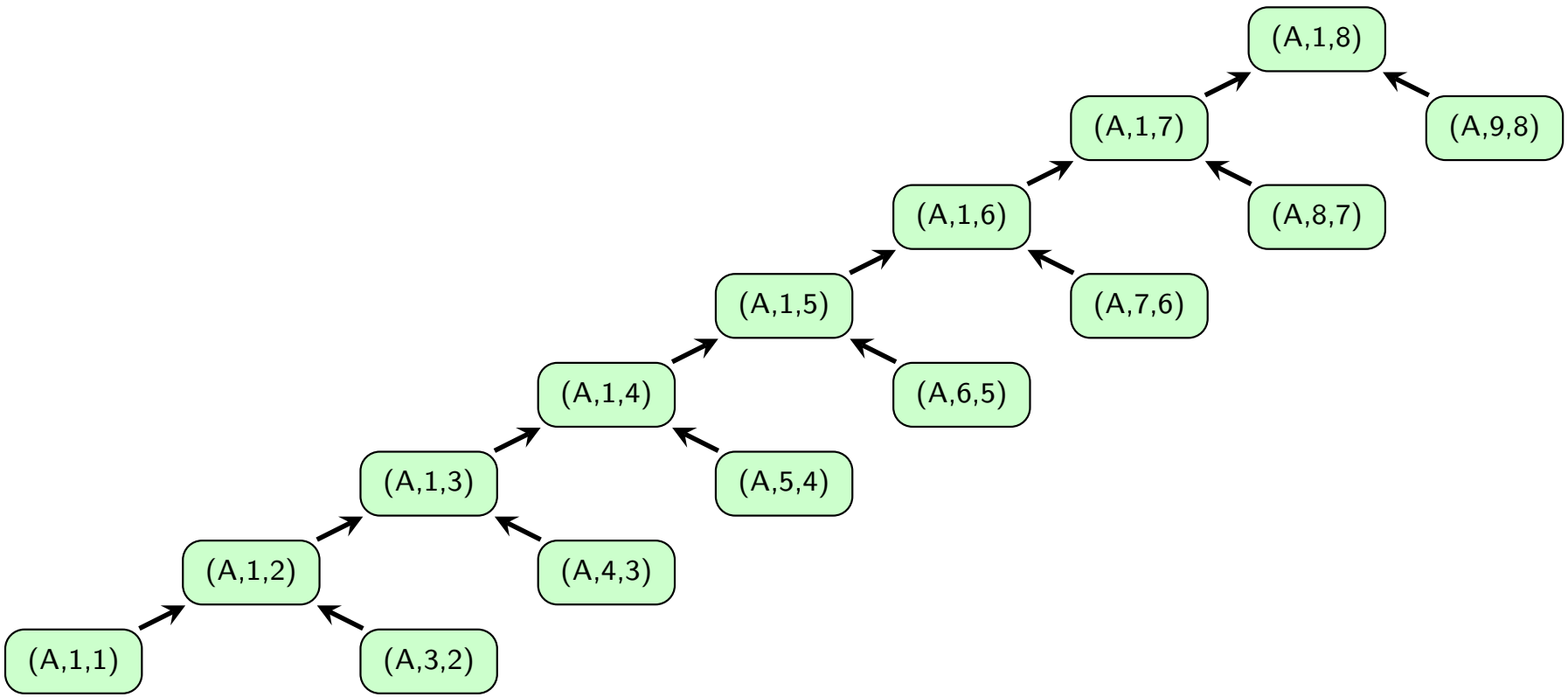




QuickSort: Worst-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8



Worst-Case Complexity

- $a = 2$: number of subproblems.
- The sizes of the two subproblems are $n - 1$ and 0 respectively
- $D(n) = \Theta(n)$: cost of the Partition procedure
- $C(n) = \Theta(1)$: cost of combining the sub-solutions
- Base case $n_0 \leq 1$
- $T(n) =$
$$\begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n-1) + T(0) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

Worst-Case Complexity

- $a = 2$: number of subproblems.
- The sizes of the two subproblems are $n - 1$ and 0 respectively
- $D(n) = \Theta(n)$: cost of the Partition procedure
- $C(n) = \Theta(1)$: cost of combining the sub-solutions
- Base case $n_0 \leq 1$
- $T(n) =$
$$\begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n-1) + T(0) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

$$T(0) = \Theta(1)$$

Worst-Case Complexity

- $a = 2$: number of subproblems.
- The sizes of the two subproblems are $n - 1$ and 0 respectively
- $D(n) = \Theta(n)$: cost of the Partition procedure
- $C(n) = \Theta(1)$: cost of combining the sub-solutions
- Base case $n_0 \leq 1$
- $T(n) =$
$$\begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

$$T(n) = \Theta(n^2)$$

QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

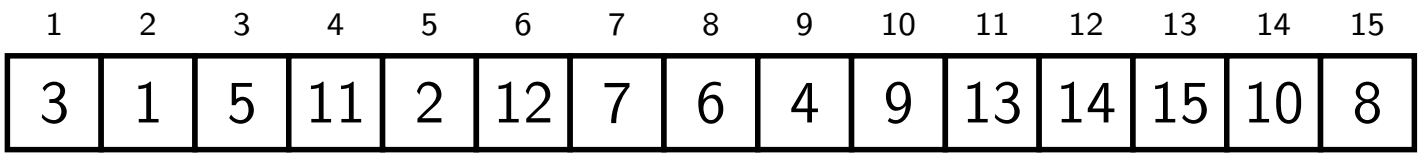
- Best Case:
  - $q = \frac{n}{2}$ (i.e., The partition produces two equal-size subarrays)
  - The size of the two subproblems is $\frac{n}{2}$.

QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

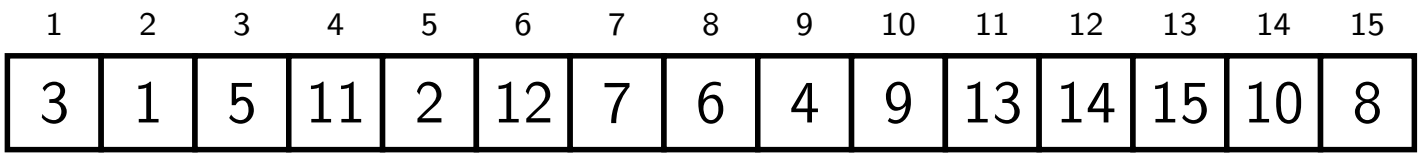
QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



QuickSort: Best-Case Behavior

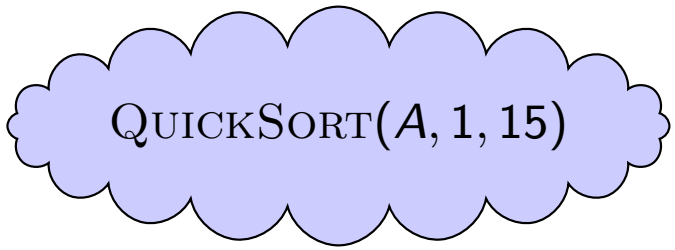
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



(A,1,15)

QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

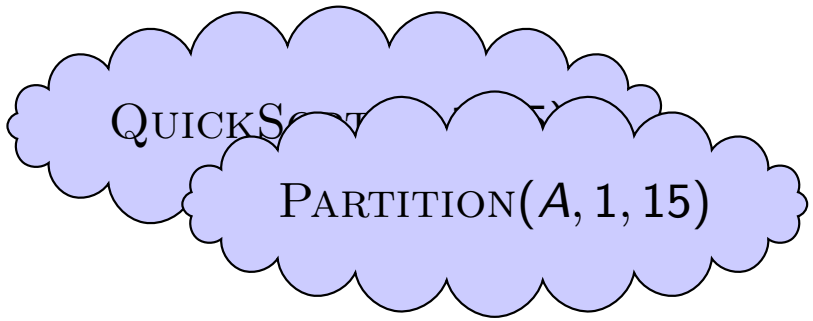


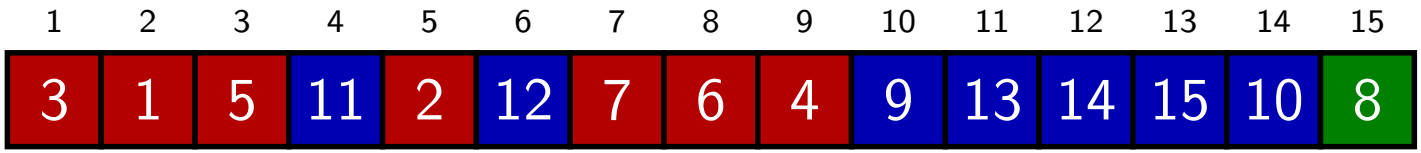
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	1	5	11	2	12	7	6	4	9	13	14	15	10	8

(A,1,15)

QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

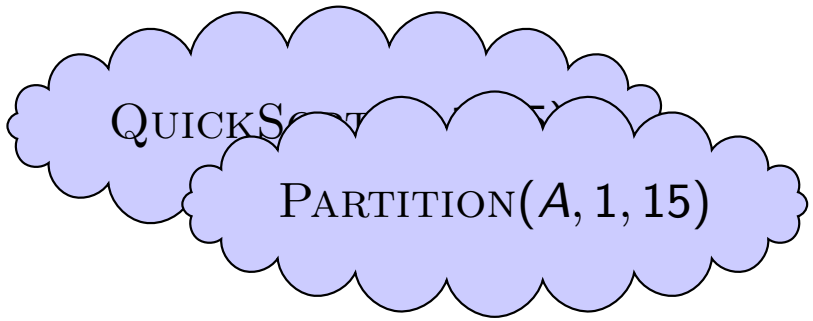


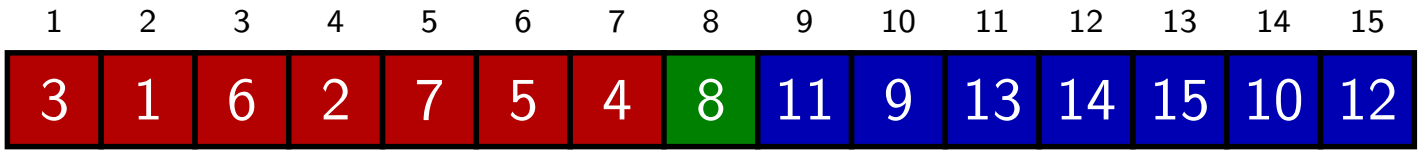


(A,1,15)

QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

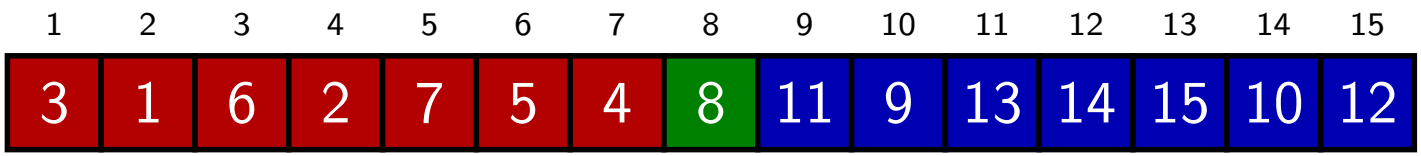


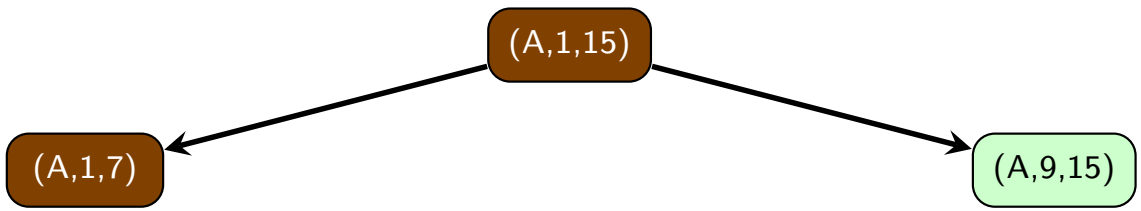


(A,1,15)

QuickSort: Best-Case Behavior

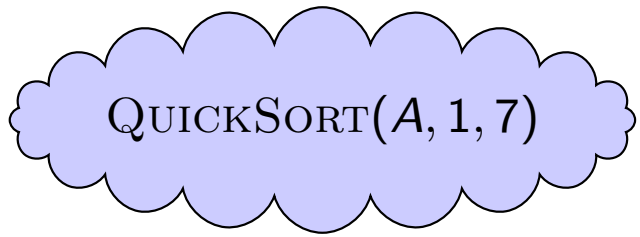
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



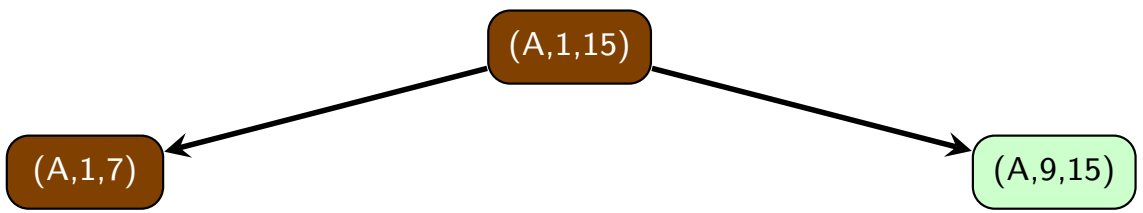


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

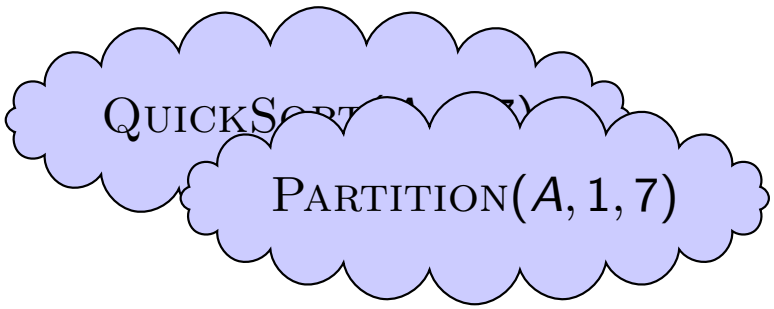


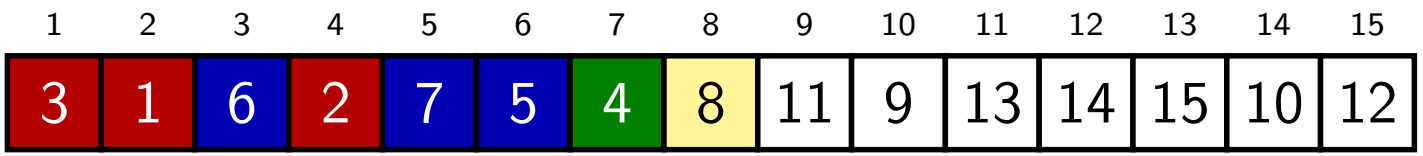
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	1	6	2	7	5	4	8	11	9	13	14	15	10	12

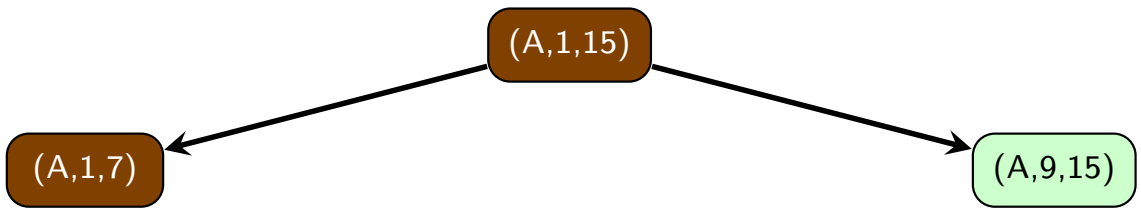


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

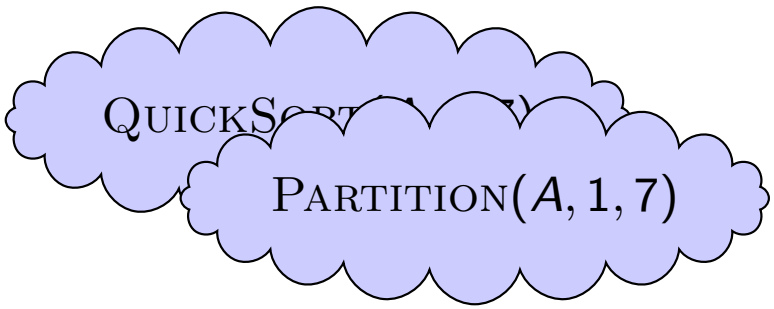


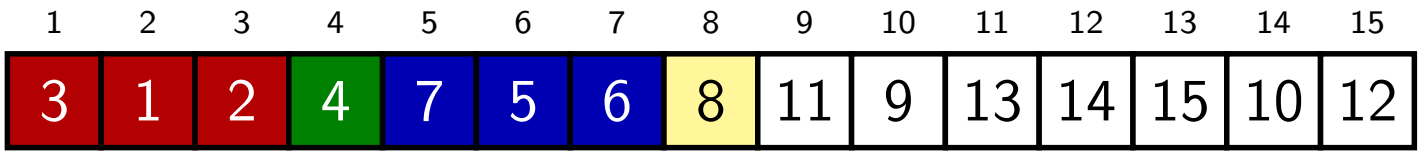


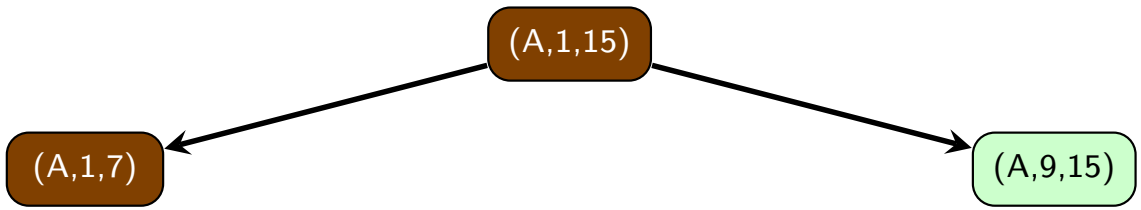


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

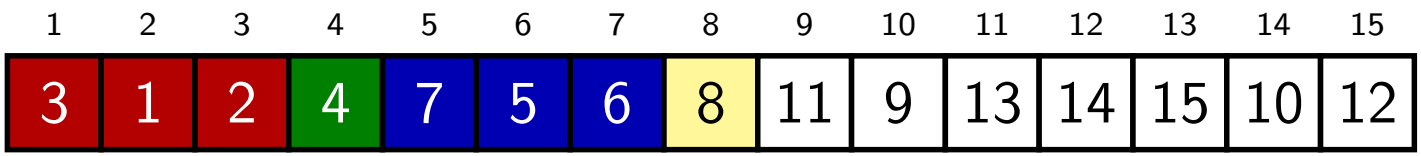


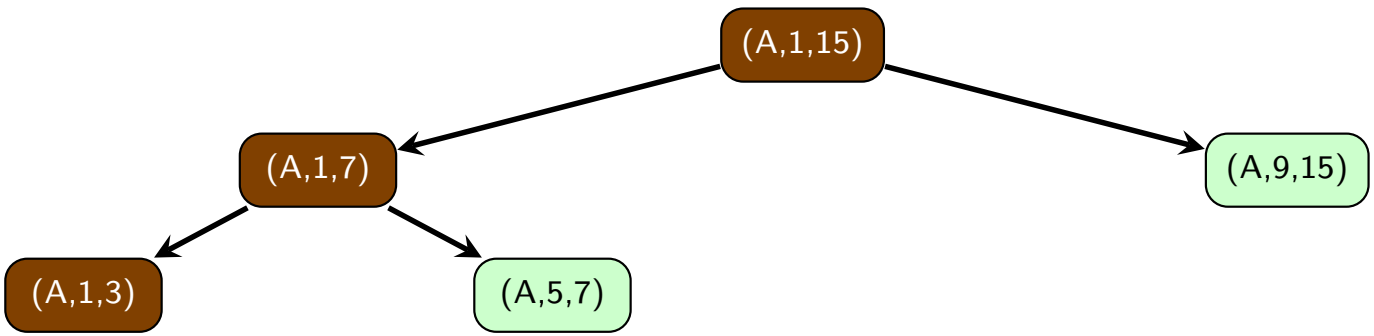




QuickSort: Best-Case Behavior

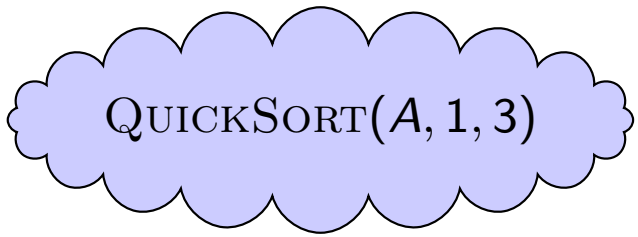
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



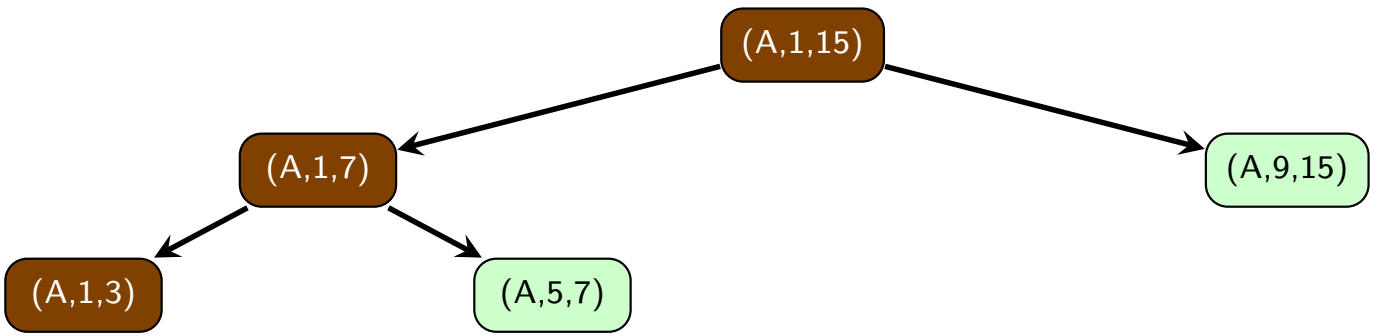


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

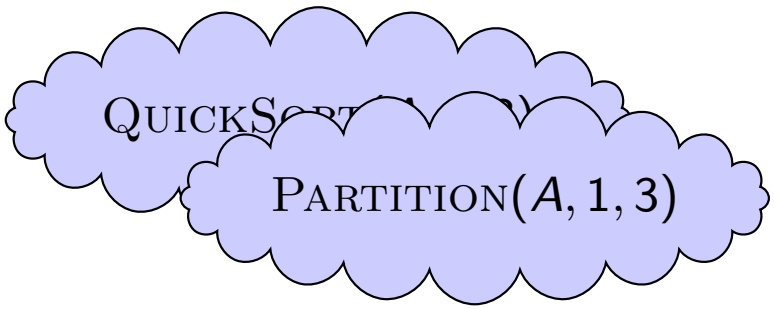


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	1	2	4	7	5	6	8	11	9	13	14	15	10	12

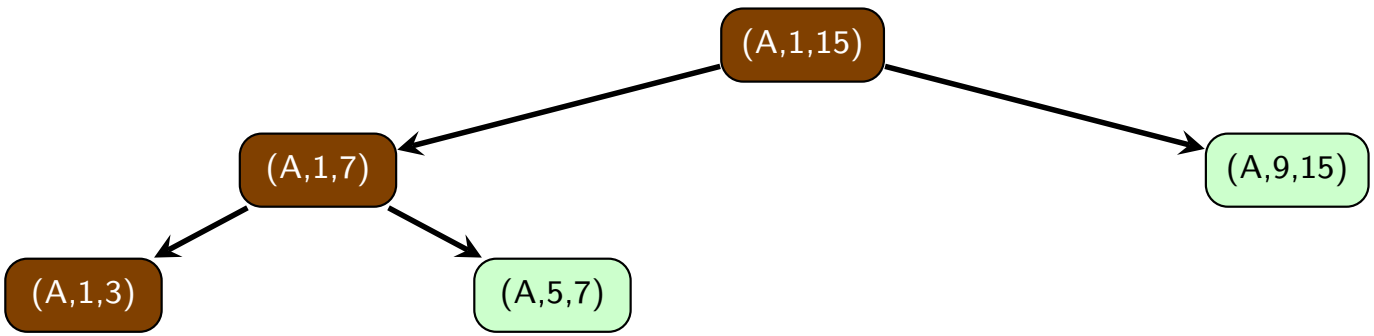


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

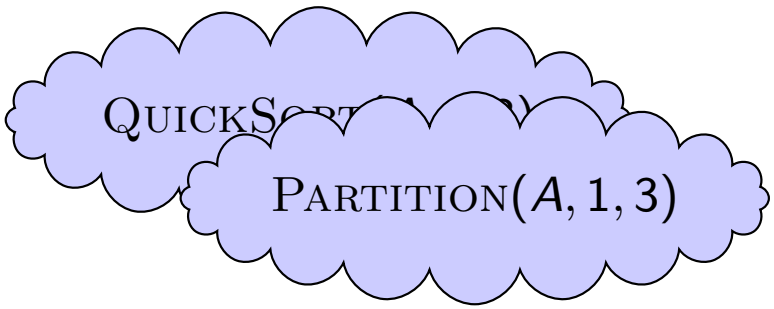


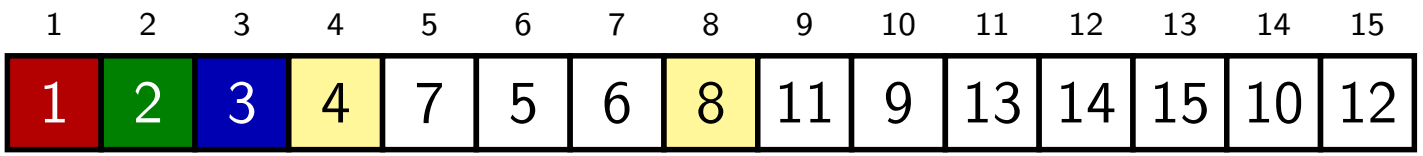
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	1	2	4	7	5	6	8	11	9	13	14	15	10	12

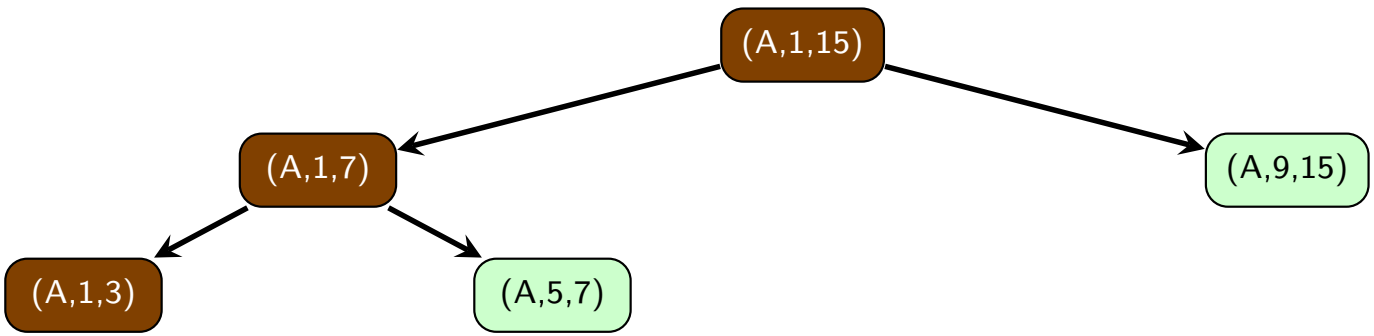


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

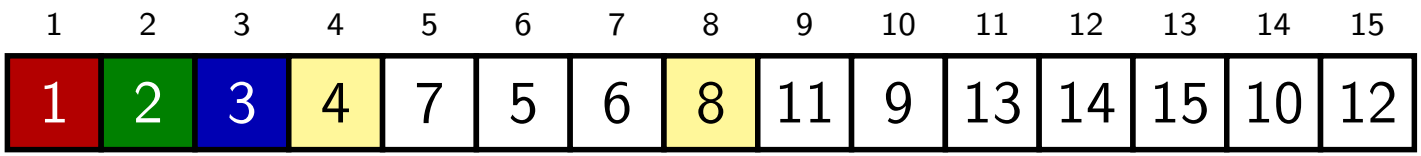


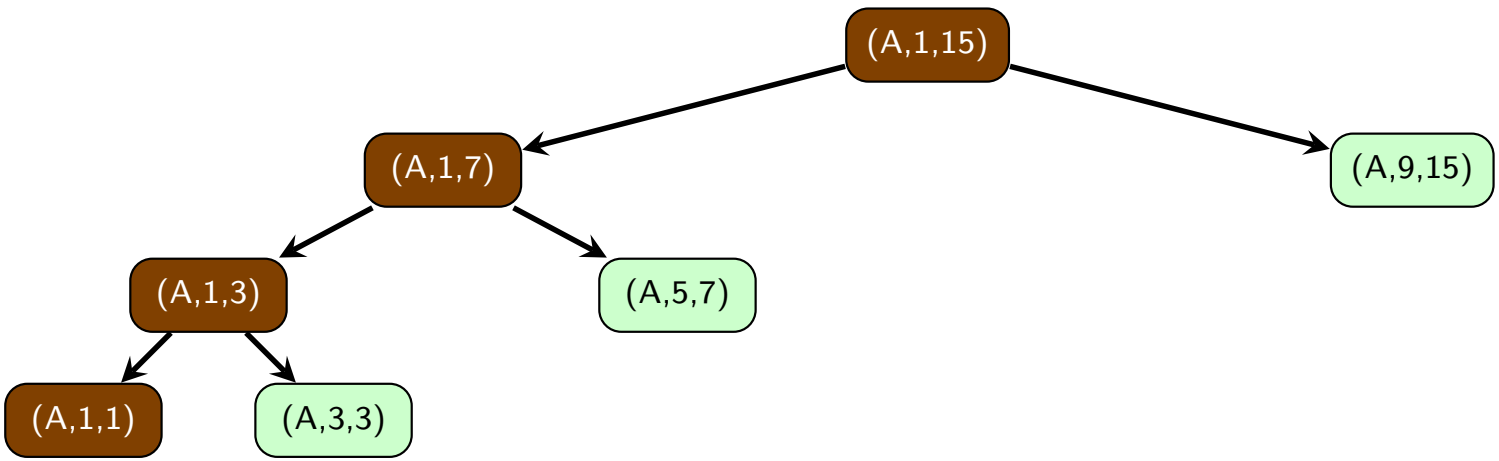




QuickSort: Best-Case Behavior

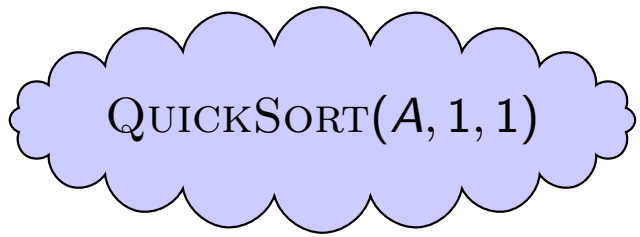
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



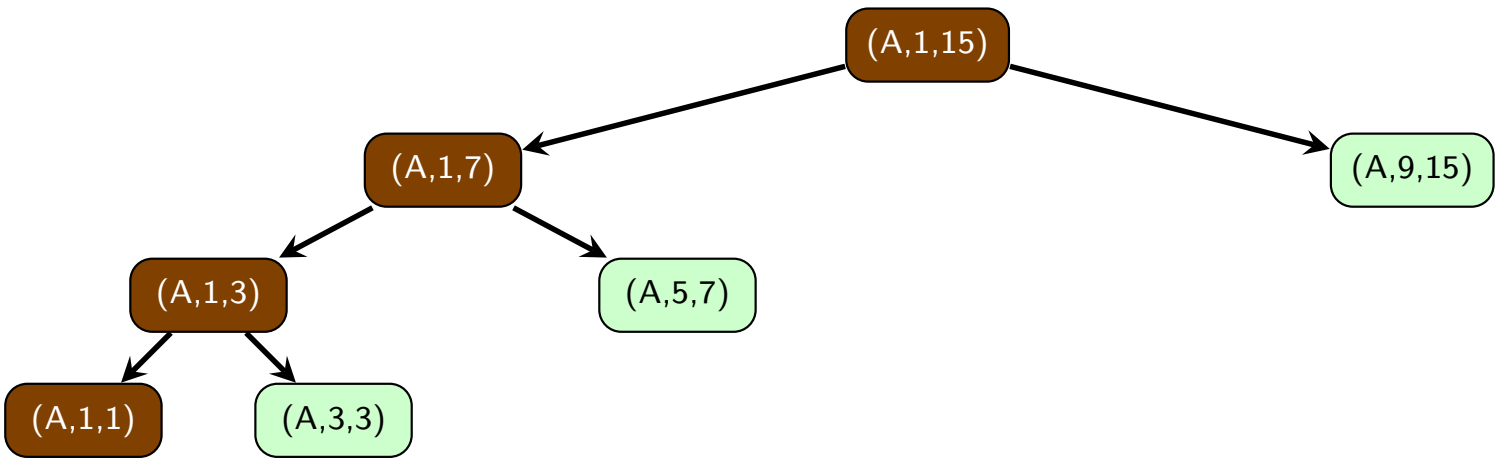


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



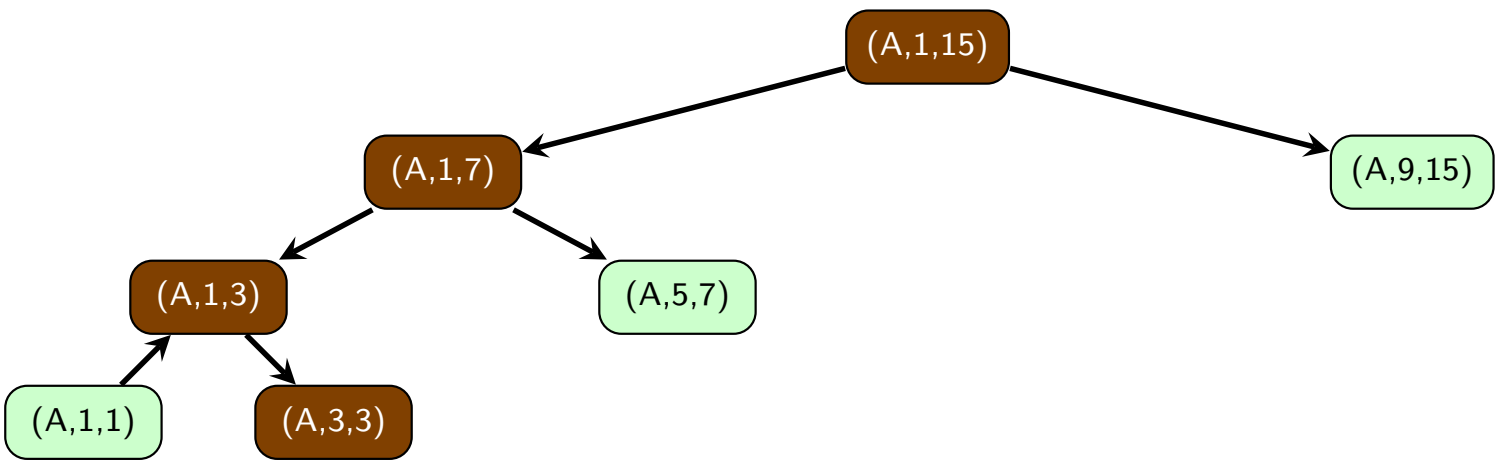
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	7	5	6	8	11	9	13	14	15	10	12



QuickSort: Best-Case Behavior

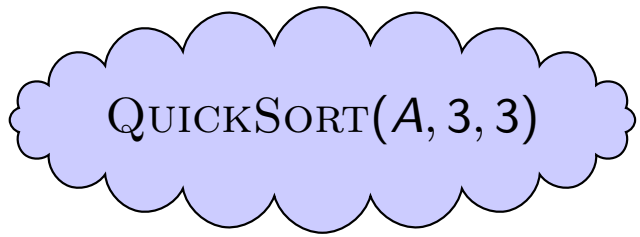
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	7	5	6	8	11	9	13	14	15	10	12

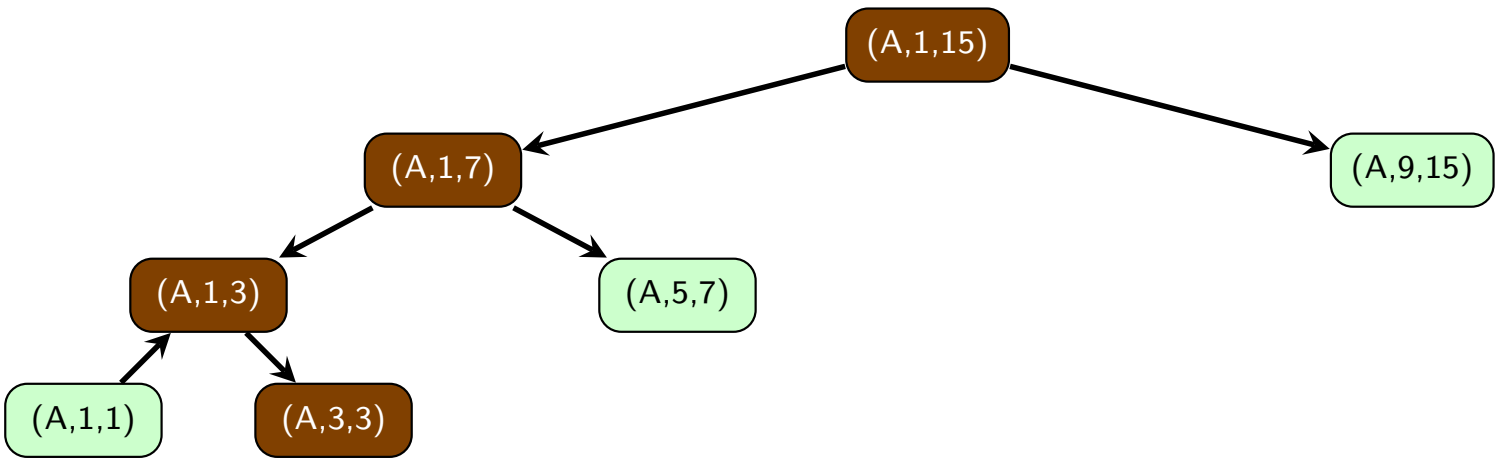


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



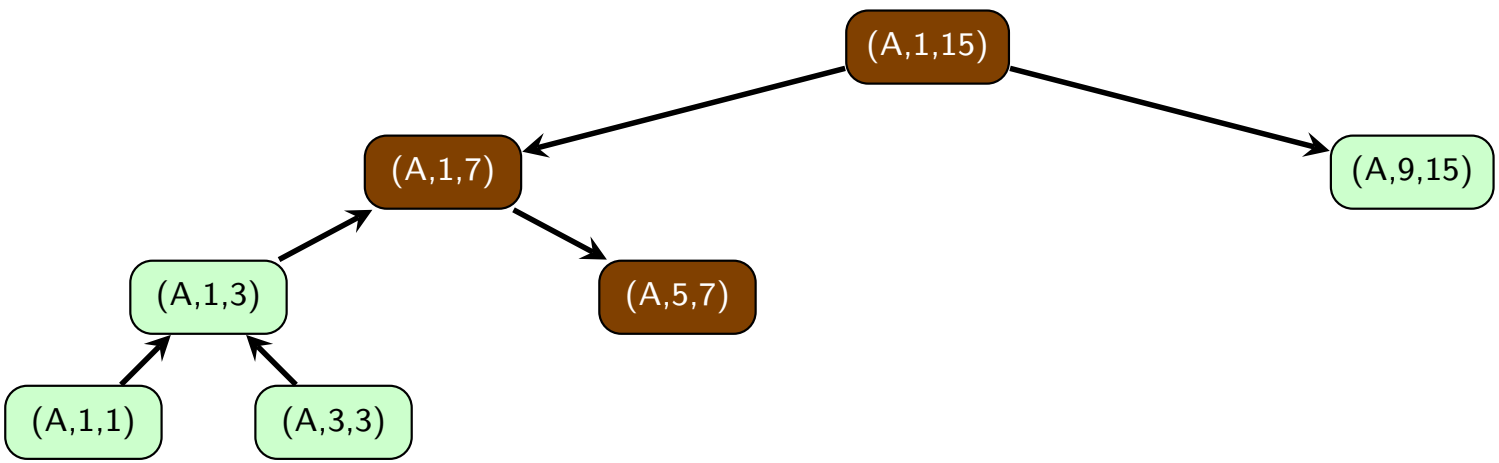
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	7	5	6	8	11	9	13	14	15	10	12



QuickSort: Best-Case Behavior

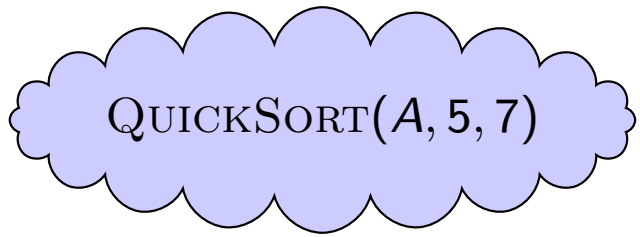
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	7	5	6	8	11	9	13	14	15	10	12

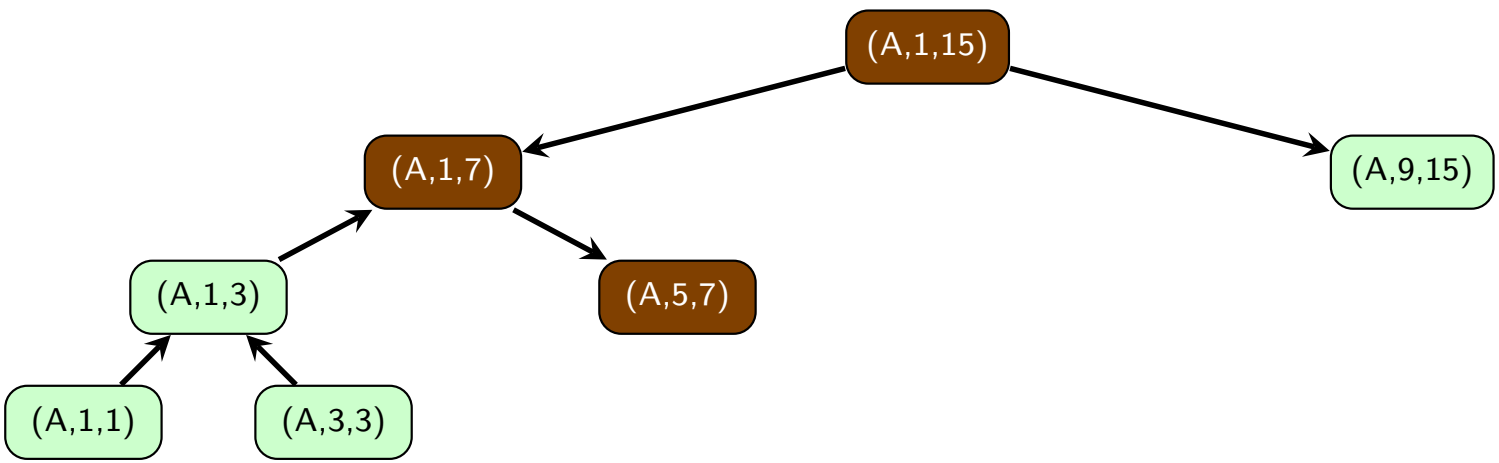


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

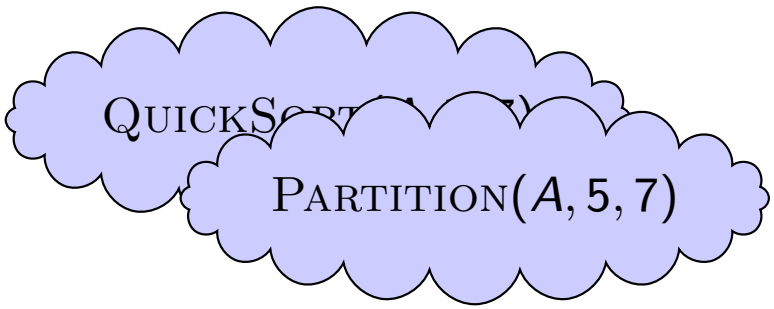


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	7	5	6	8	11	9	13	14	15	10	12

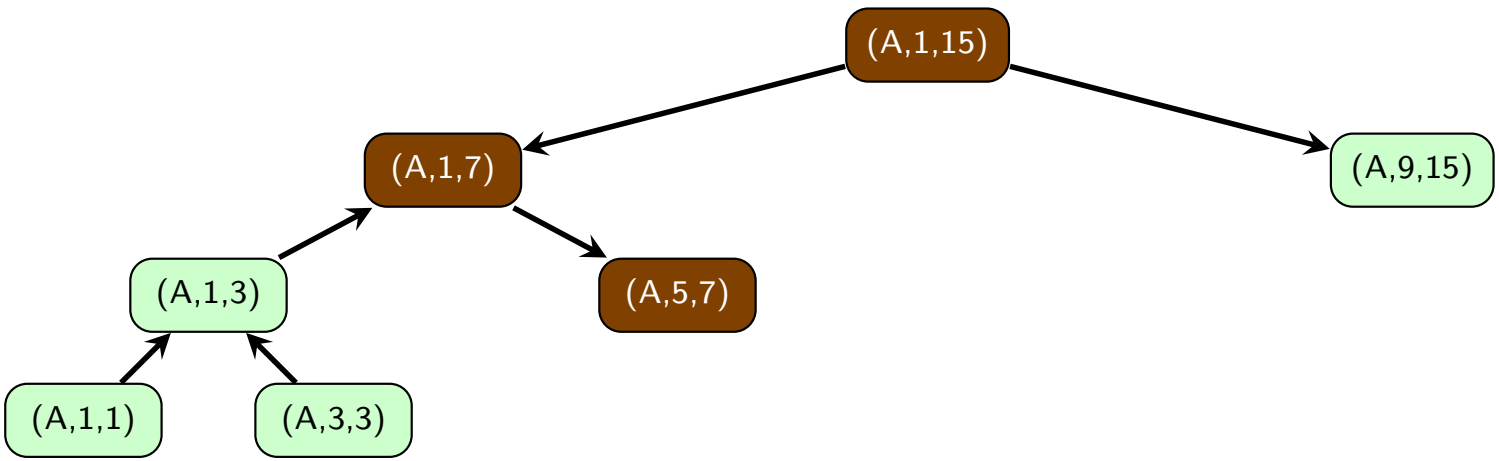


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

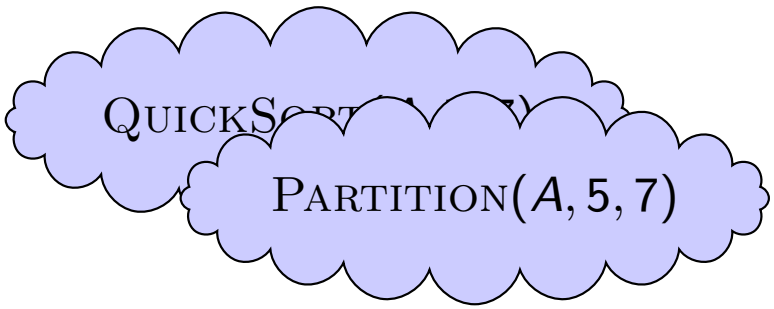


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	7	5	6	8	11	9	13	14	15	10	12

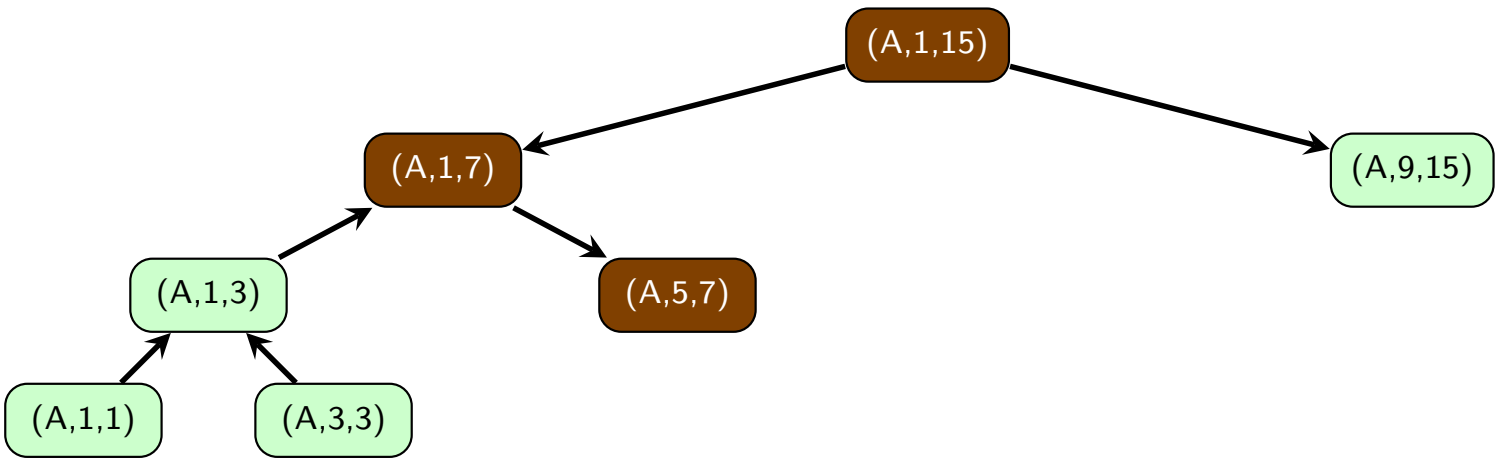


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

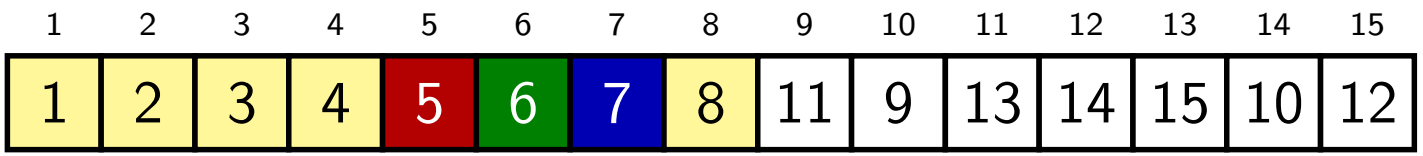


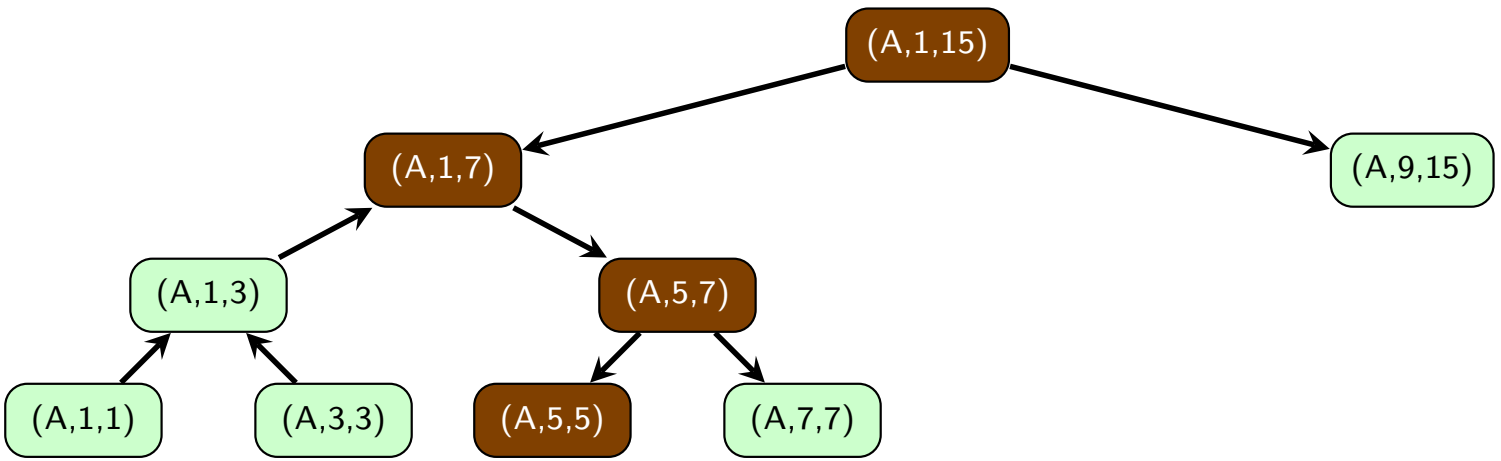
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	11	9	13	14	15	10	12



QuickSort: Best-Case Behavior

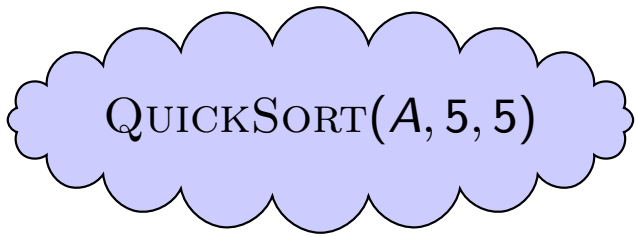
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



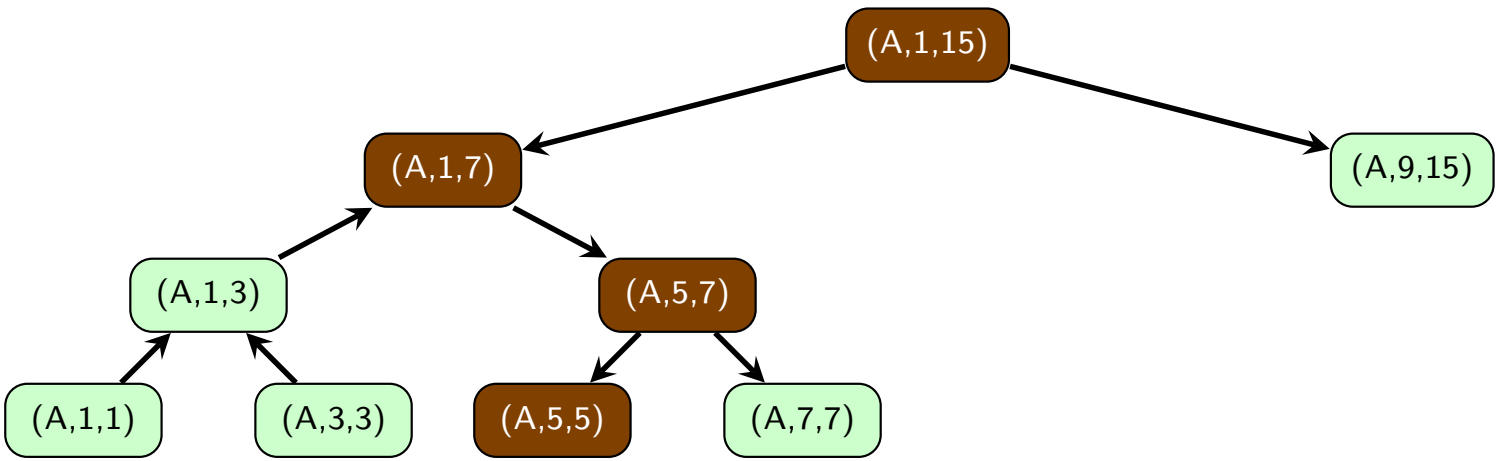


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

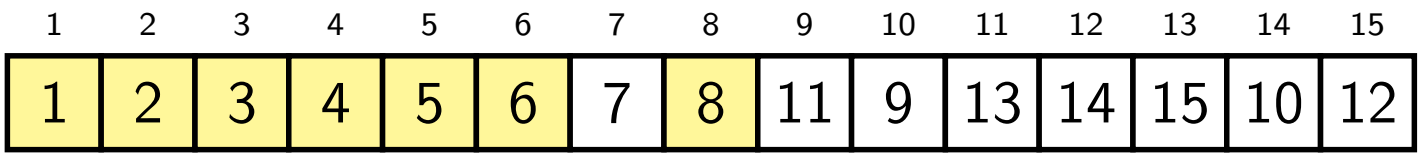


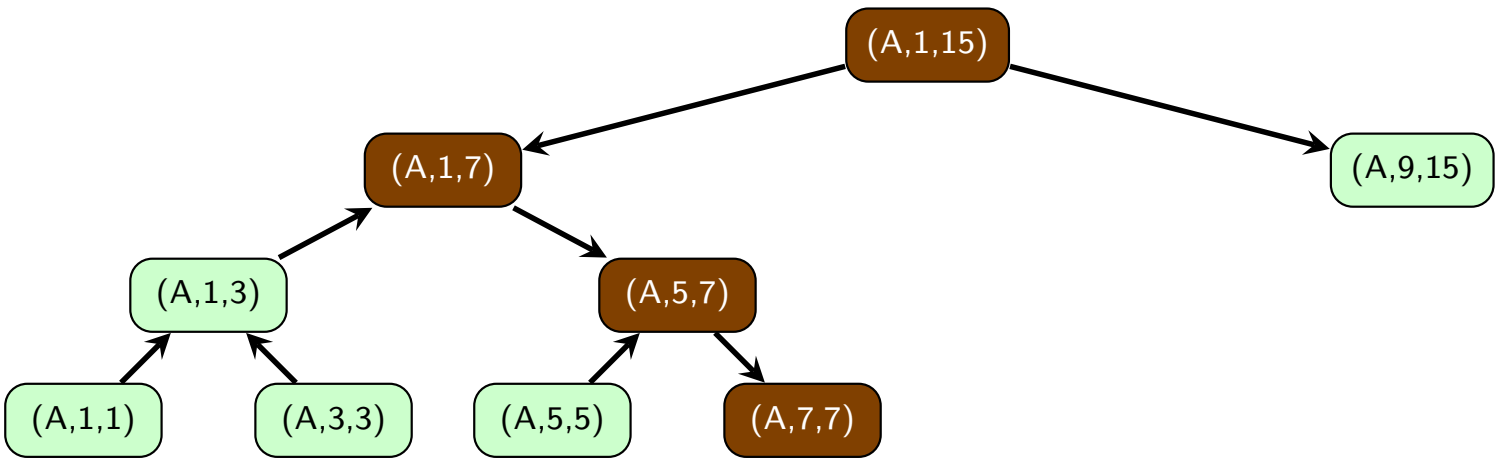
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	11	9	13	14	15	10	12



QuickSort: Best-Case Behavior

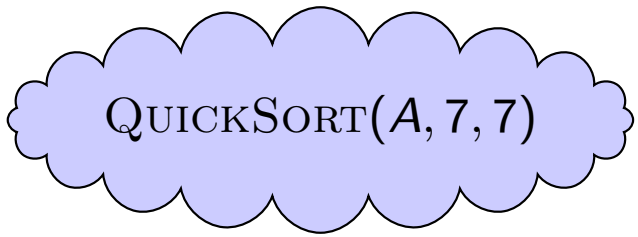
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



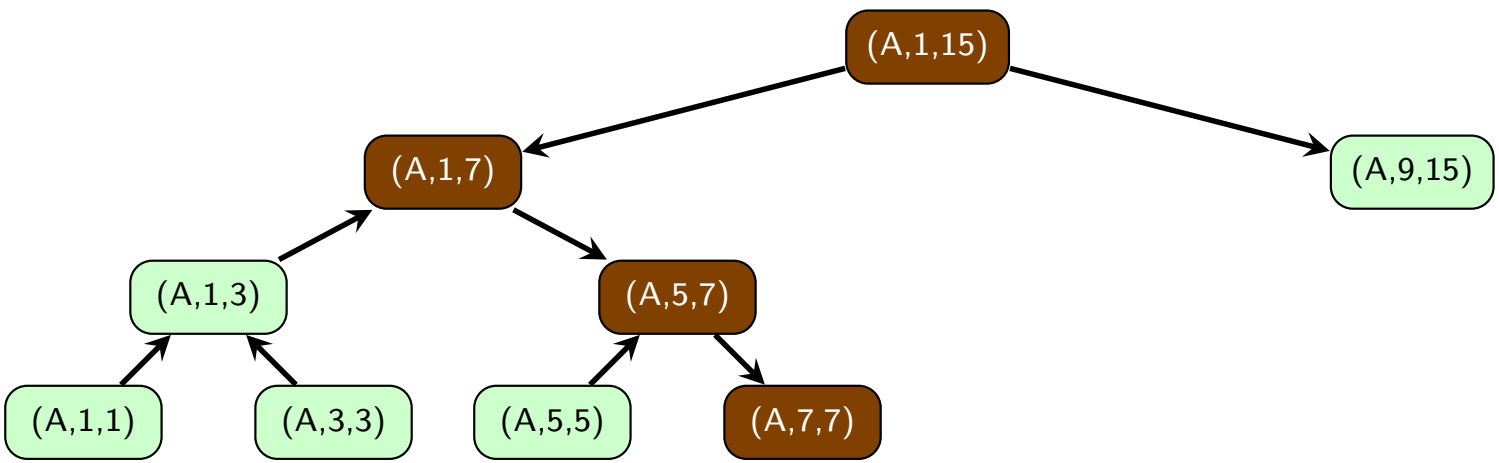


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

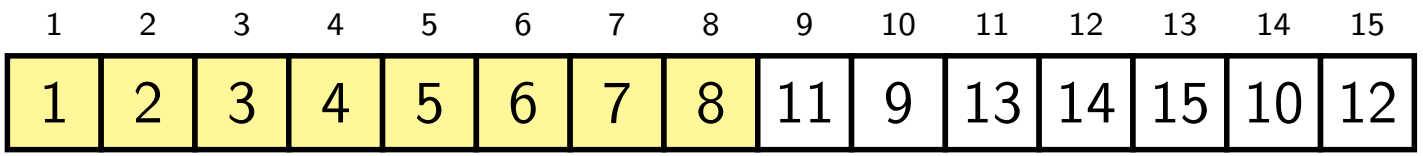


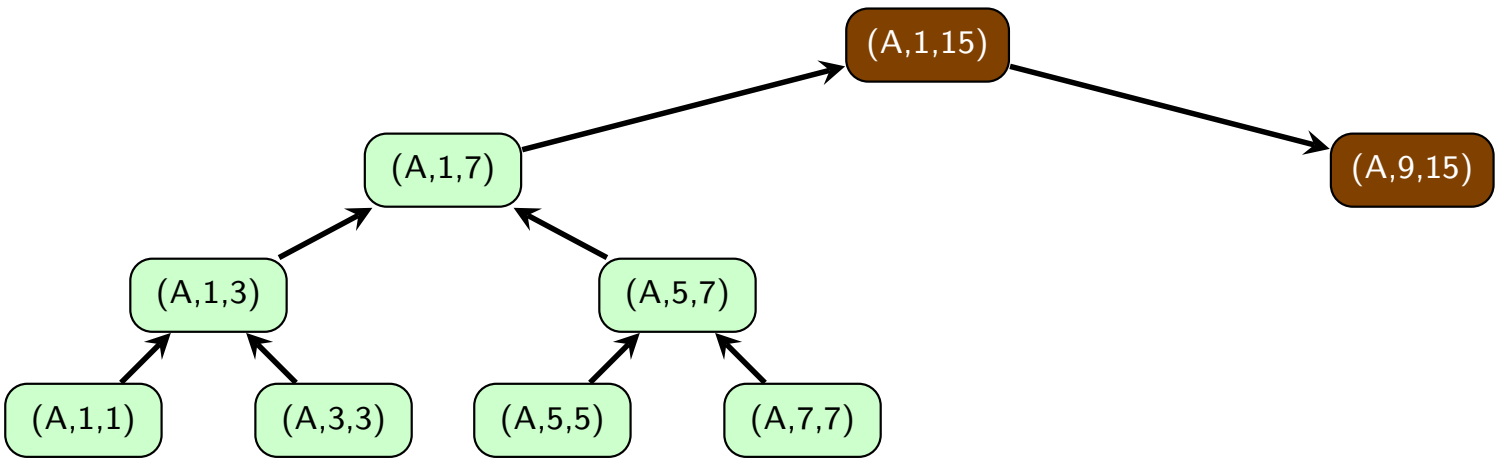
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	11	9	13	14	15	10	12



QuickSort: Best-Case Behavior

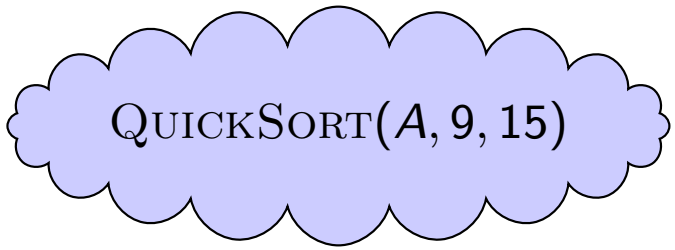
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



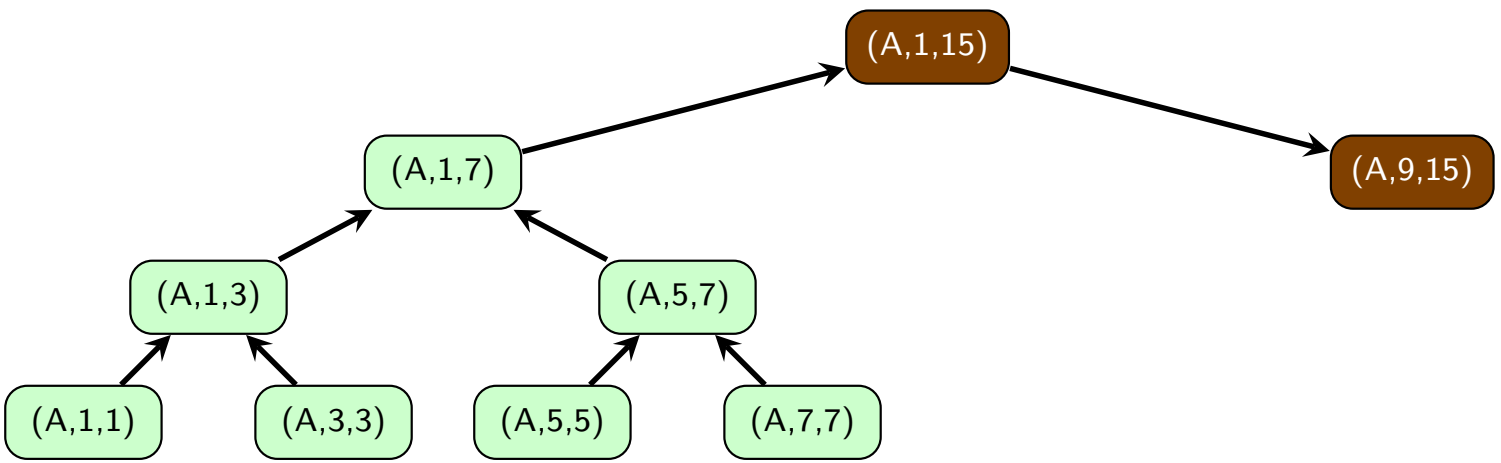


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

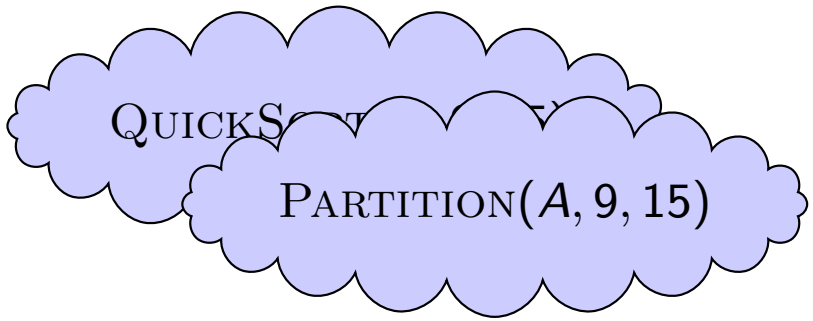


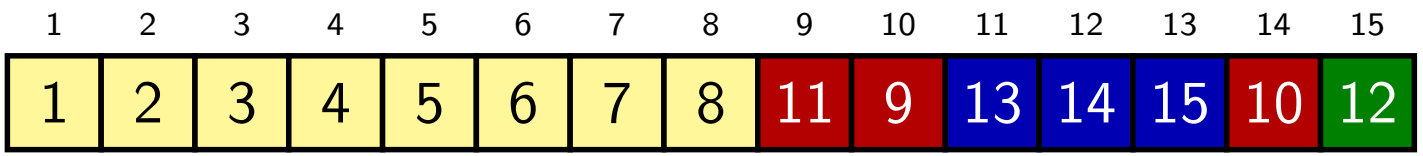
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	11	9	13	14	15	10	12

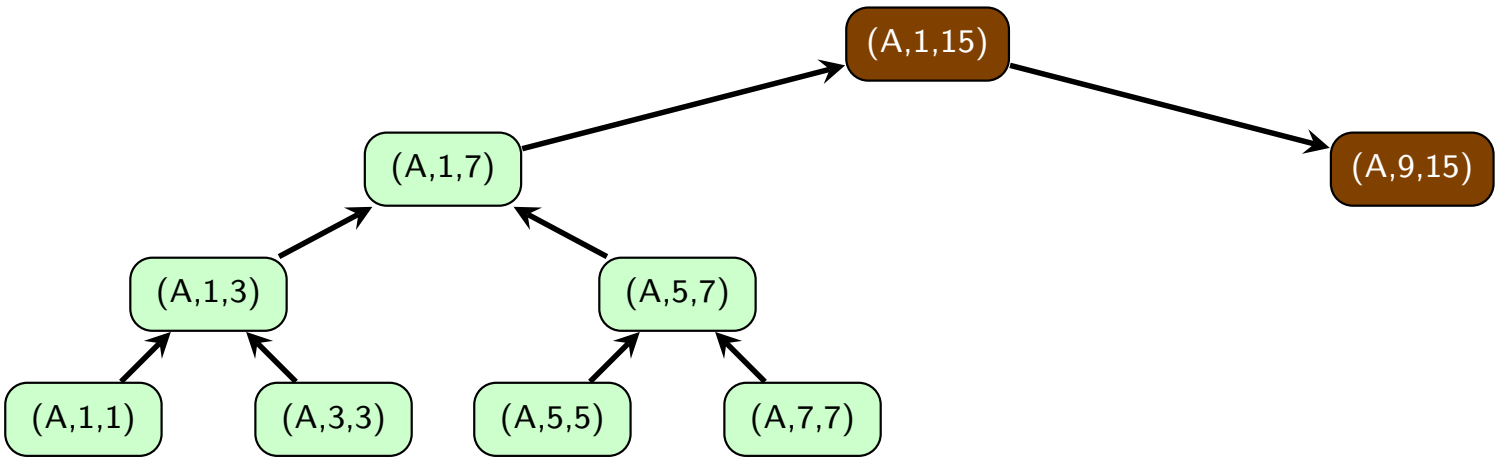


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

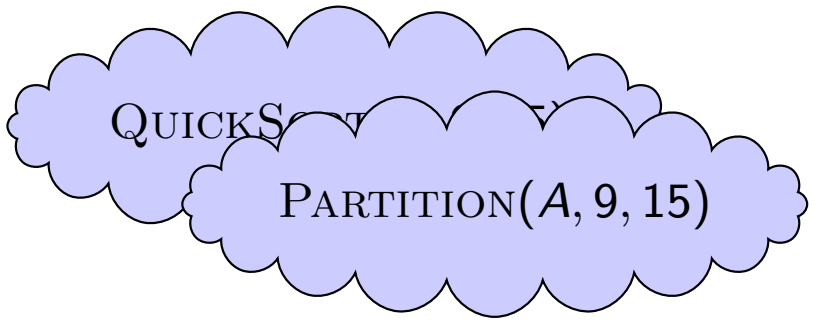


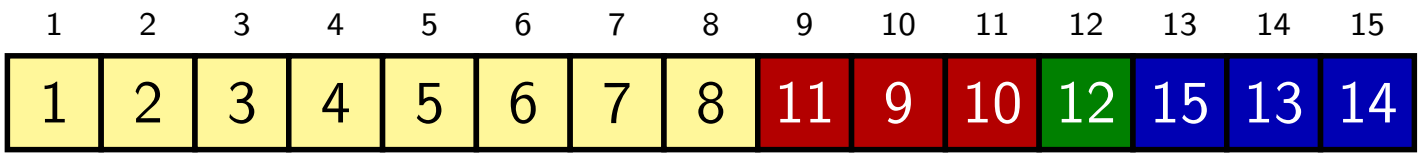


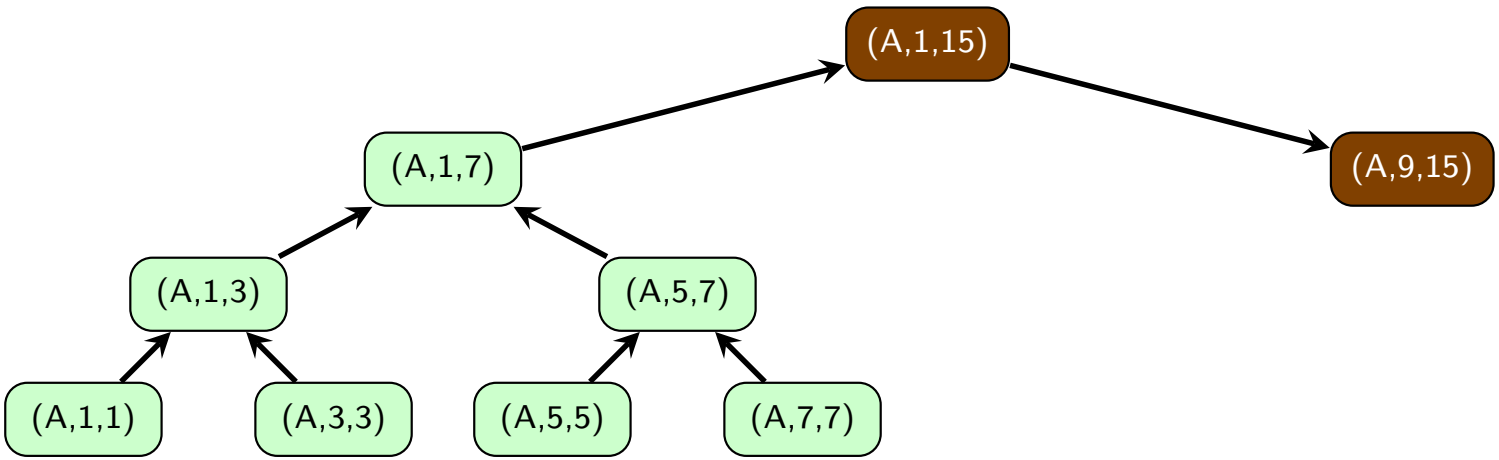


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

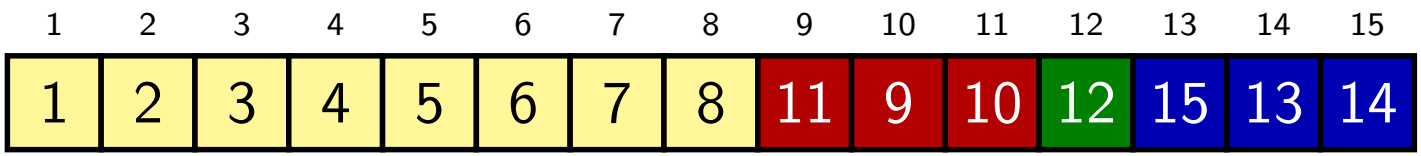


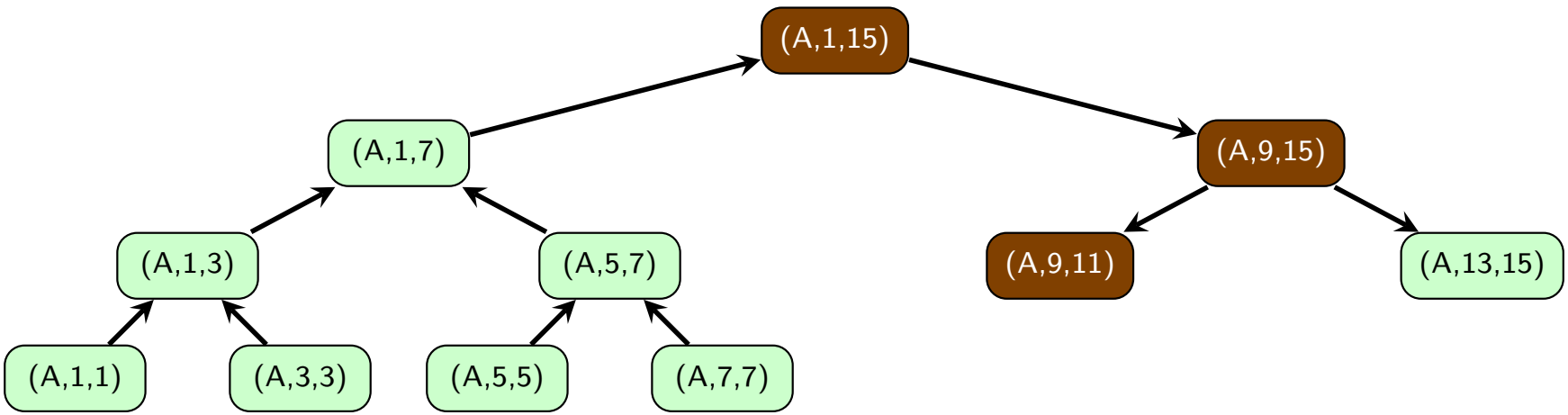




QuickSort: Best-Case Behavior

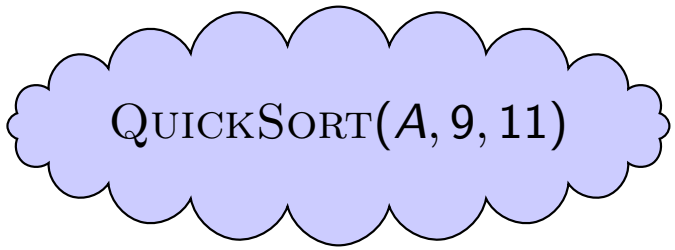
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```



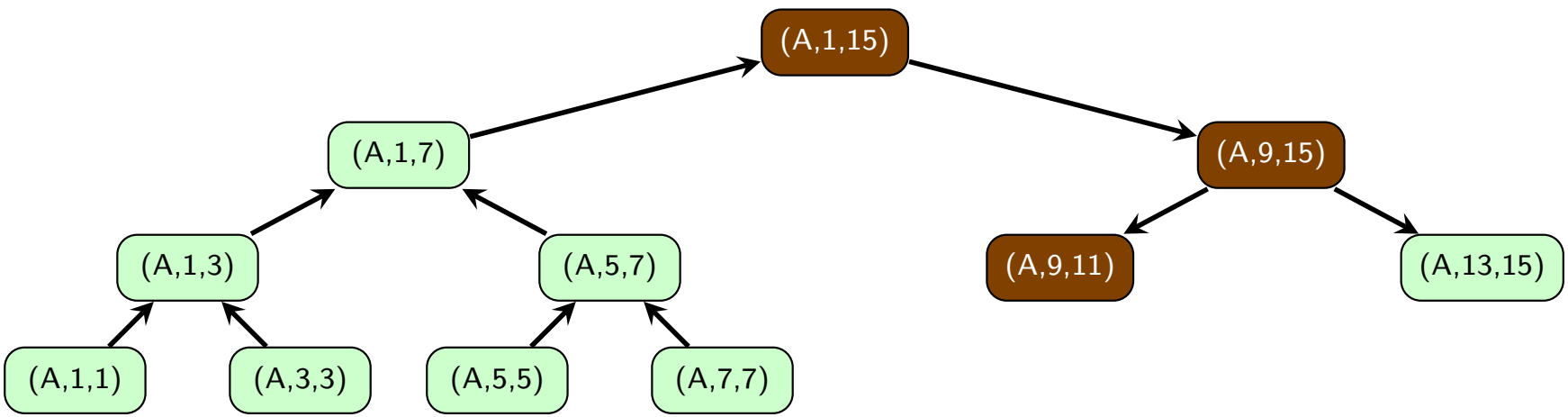


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

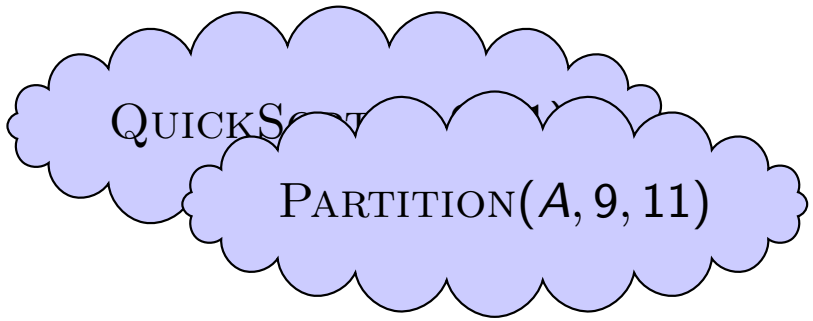


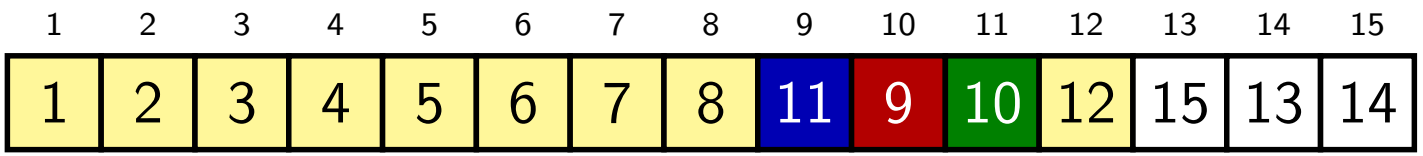
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	11	9	10	12	15	13	14

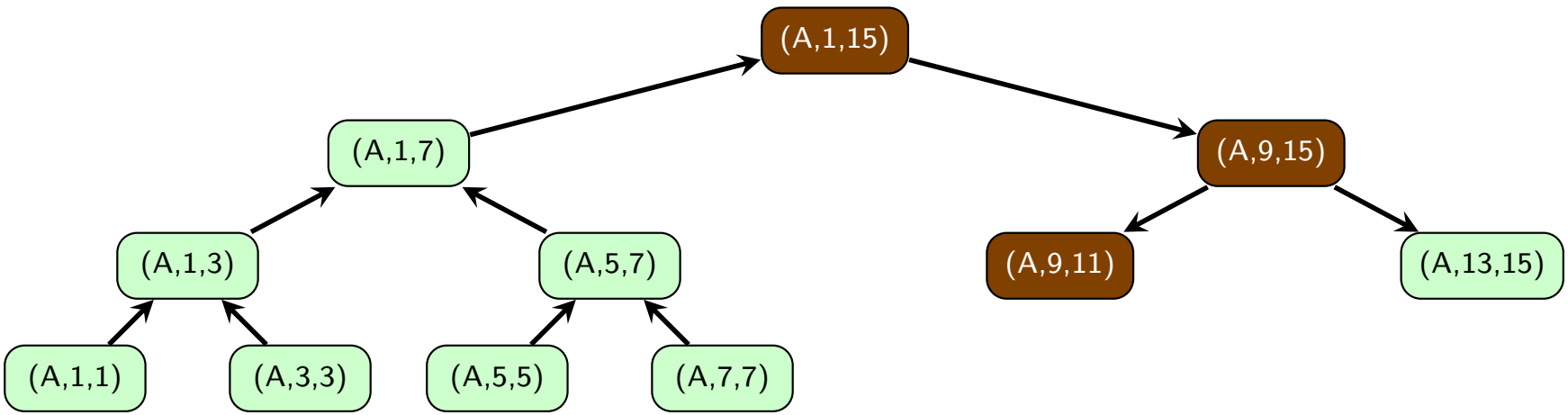


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

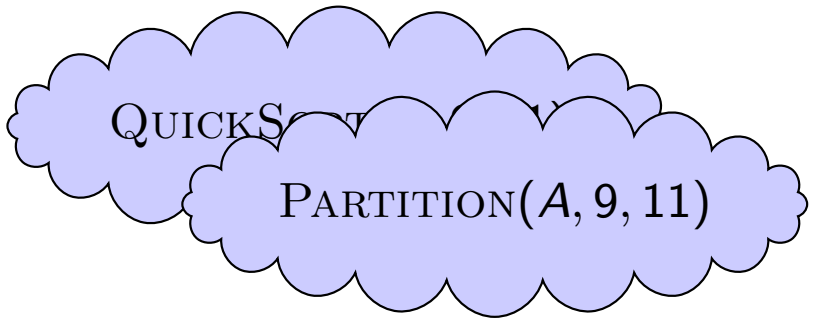


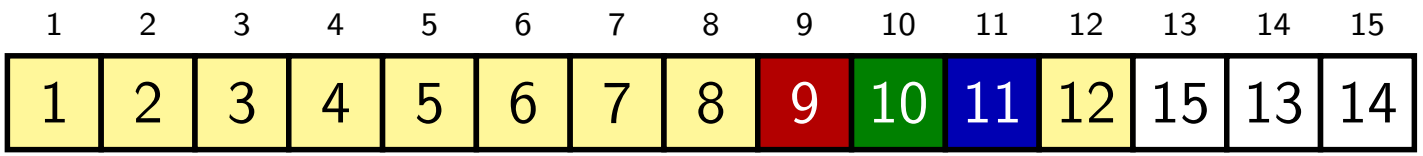


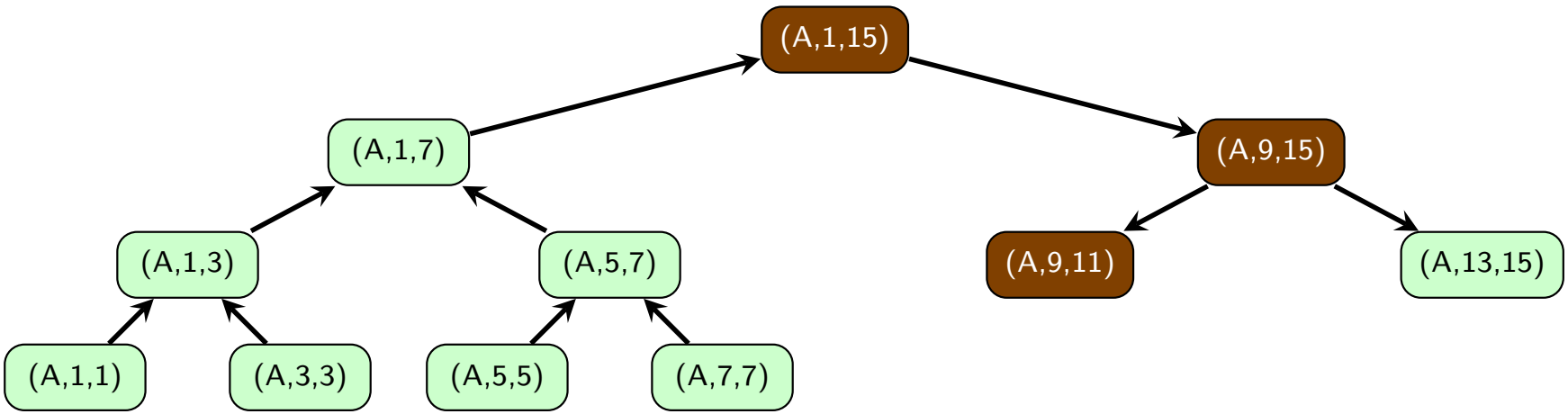


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

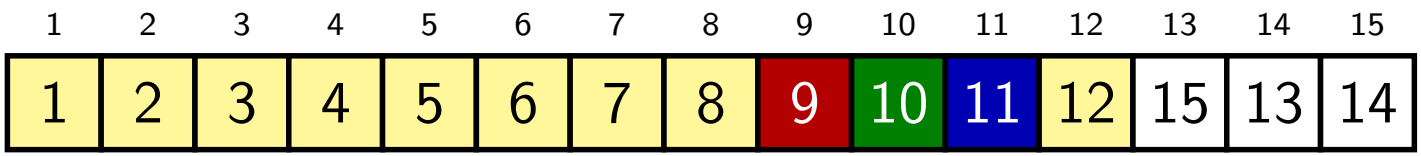


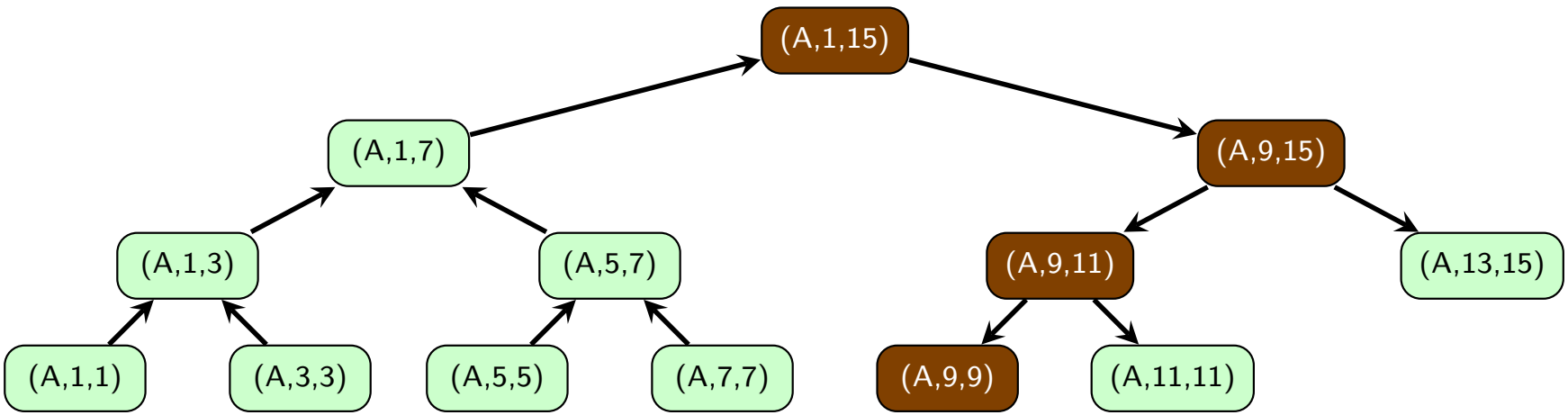




QuickSort: Best-Case Behavior

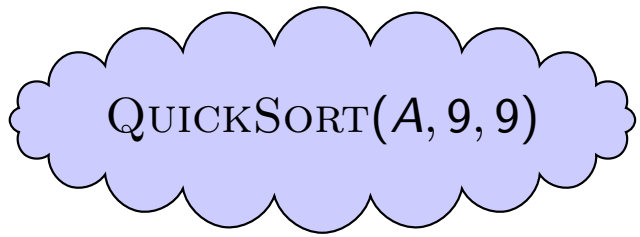
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

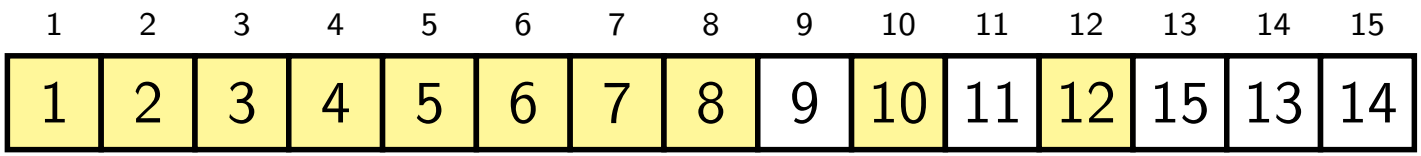


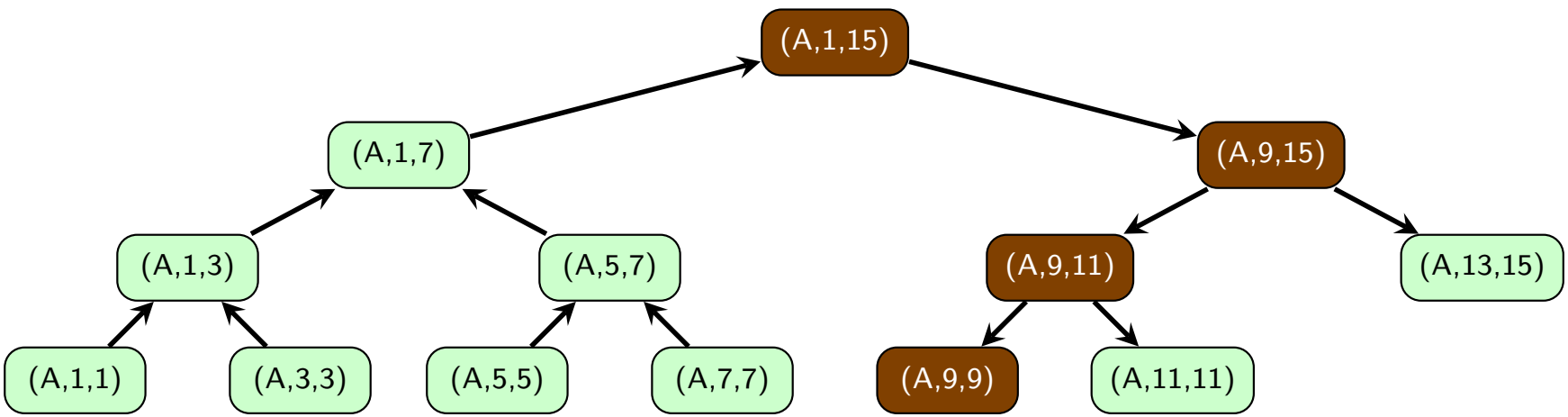


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

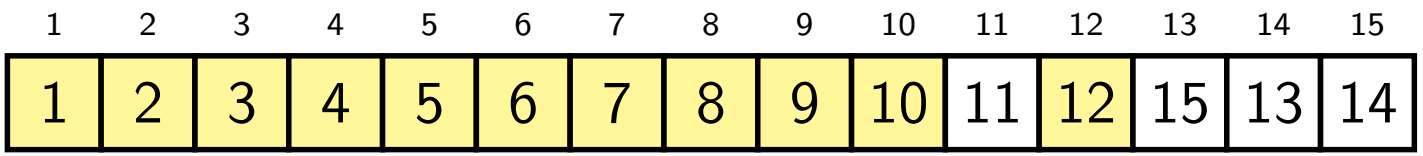


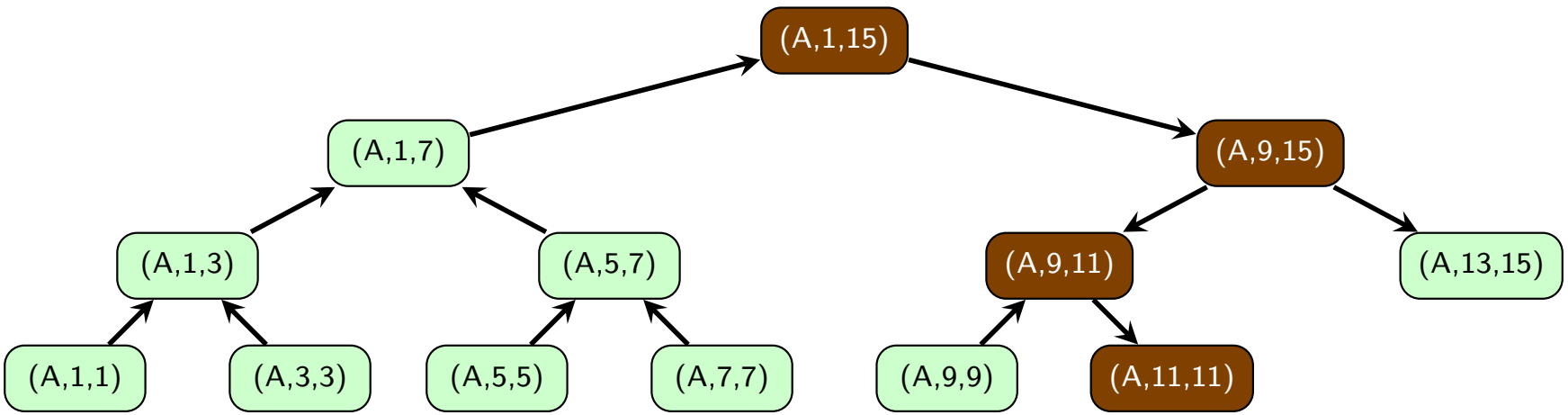




QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

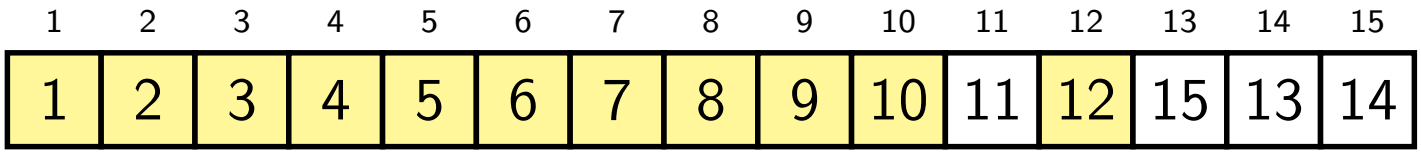


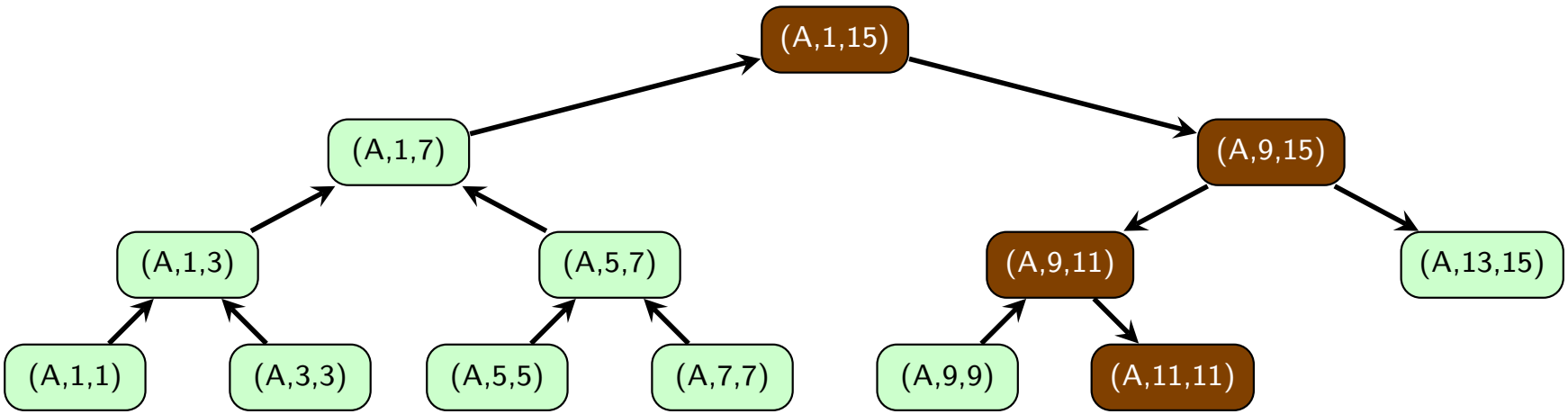


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

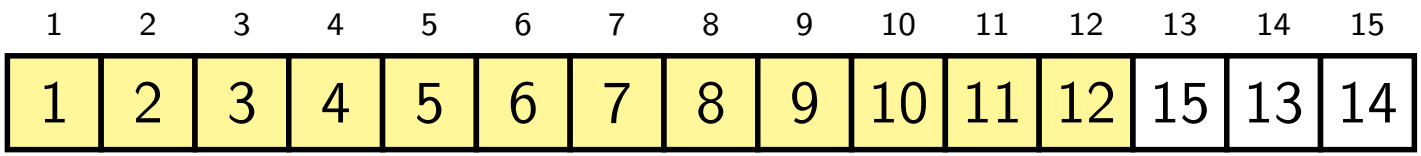
QuickSort$(A, 11, 11)$

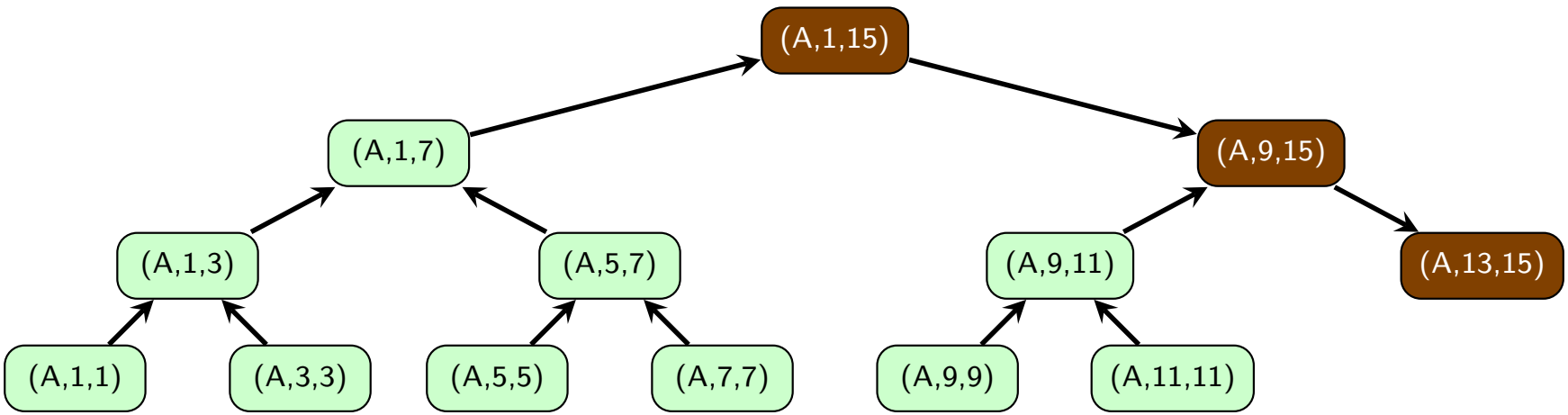




QuickSort: Best-Case Behavior

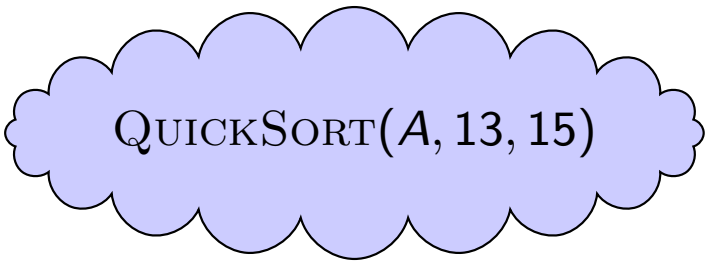
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

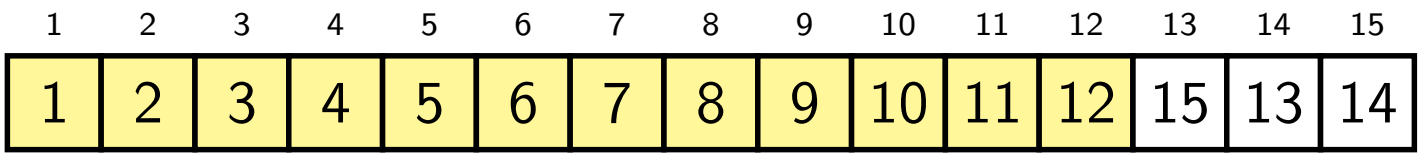


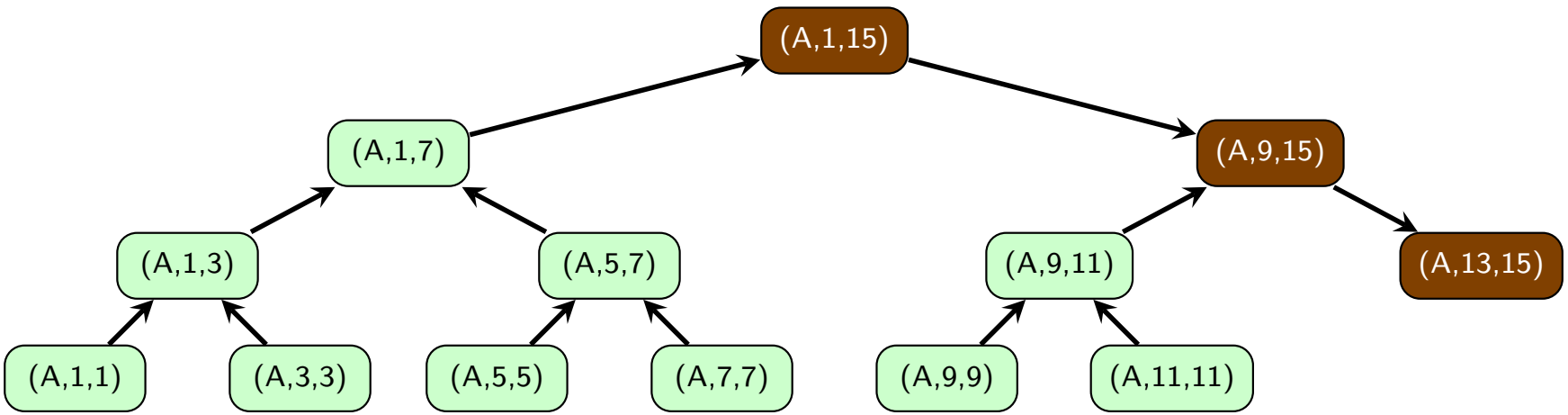


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

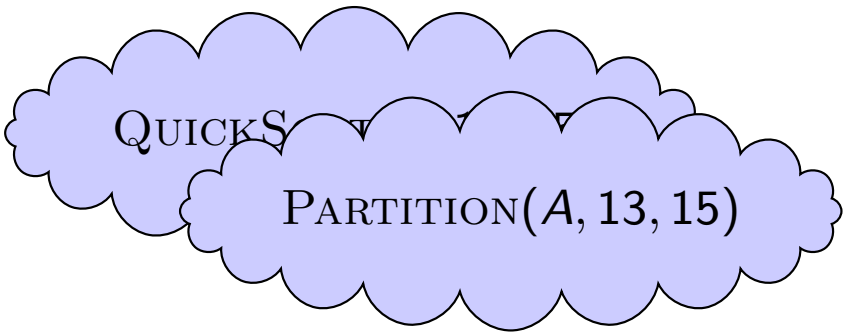


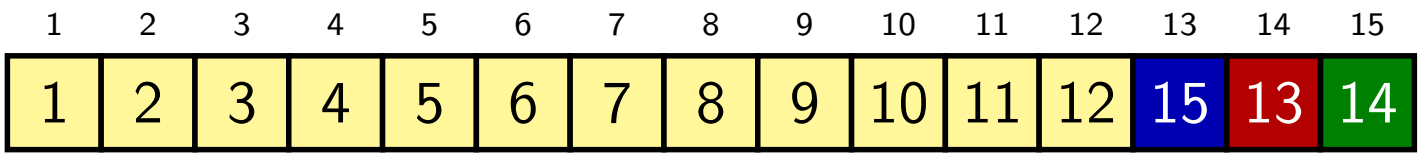


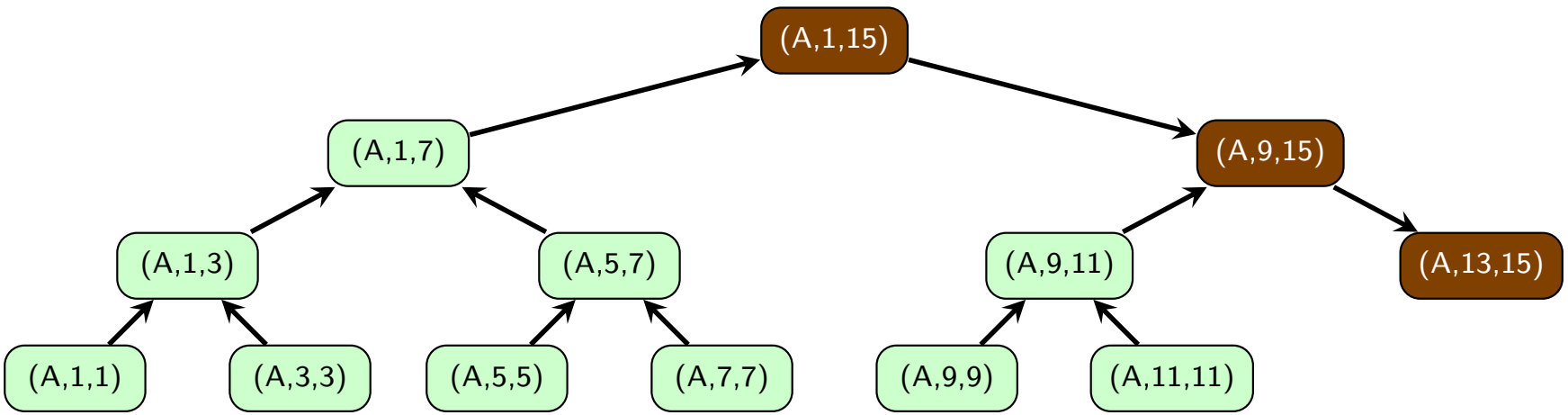


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

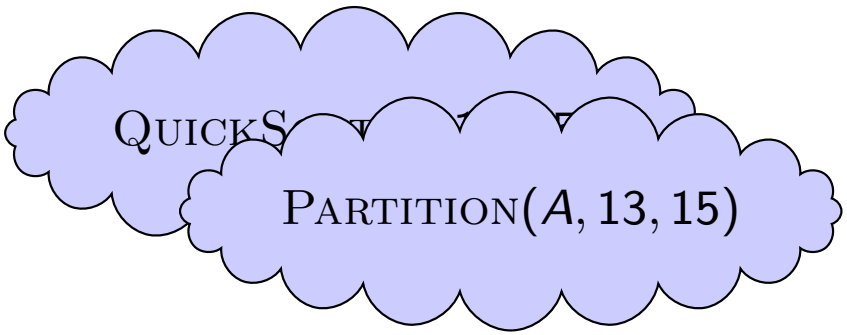


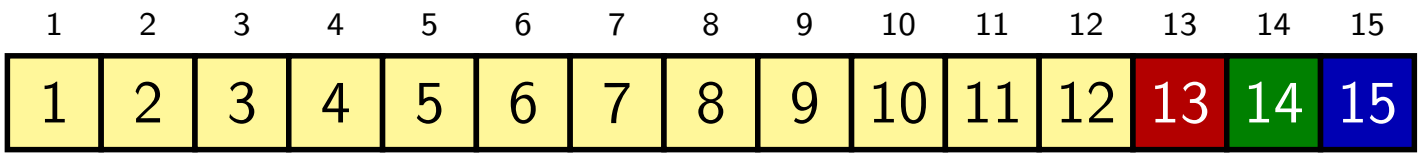


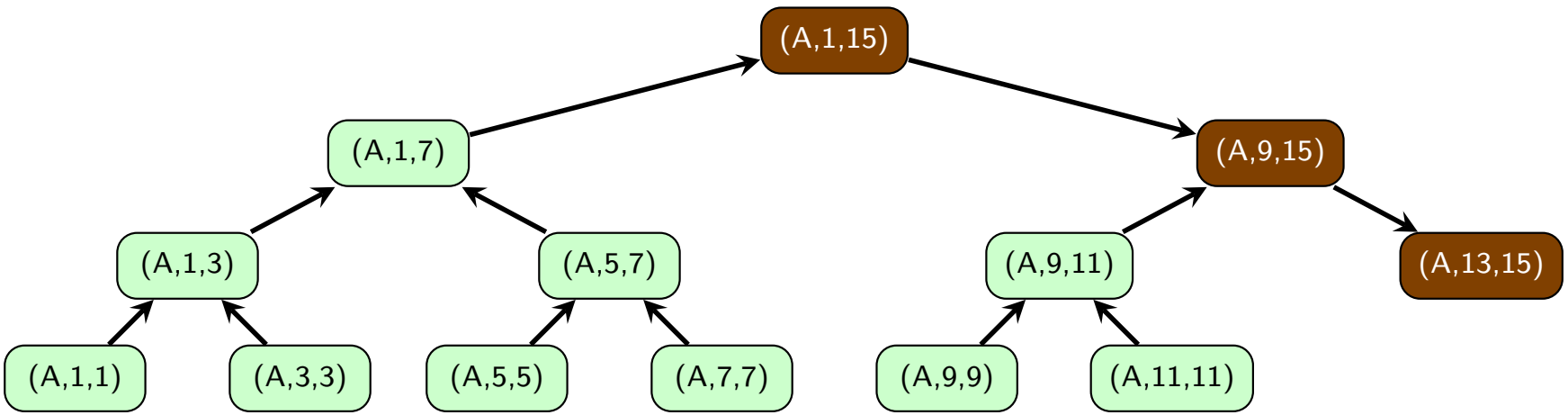


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

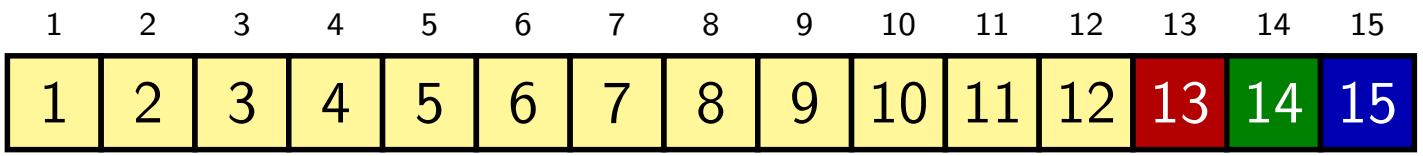


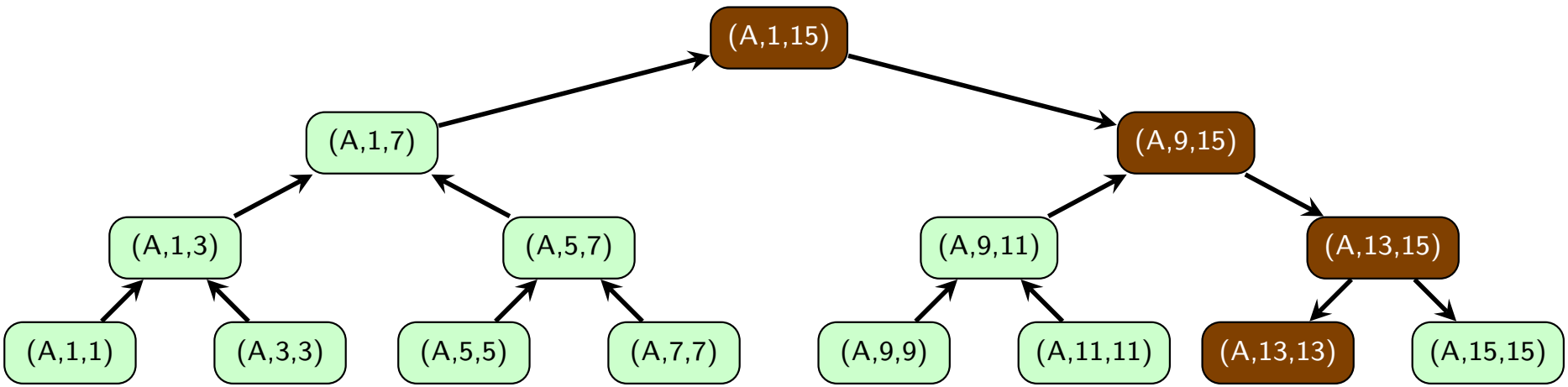




QuickSort: Best-Case Behavior

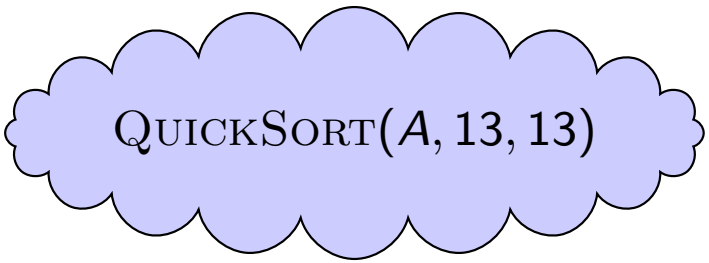
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

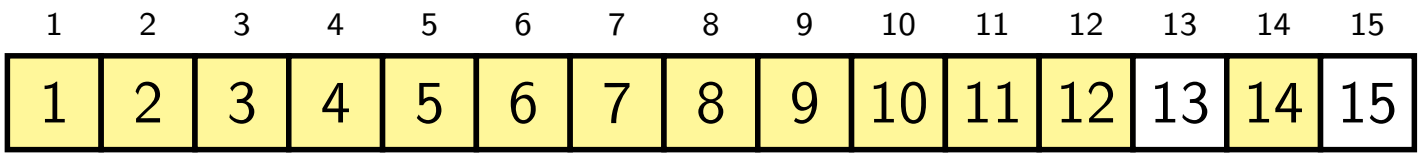


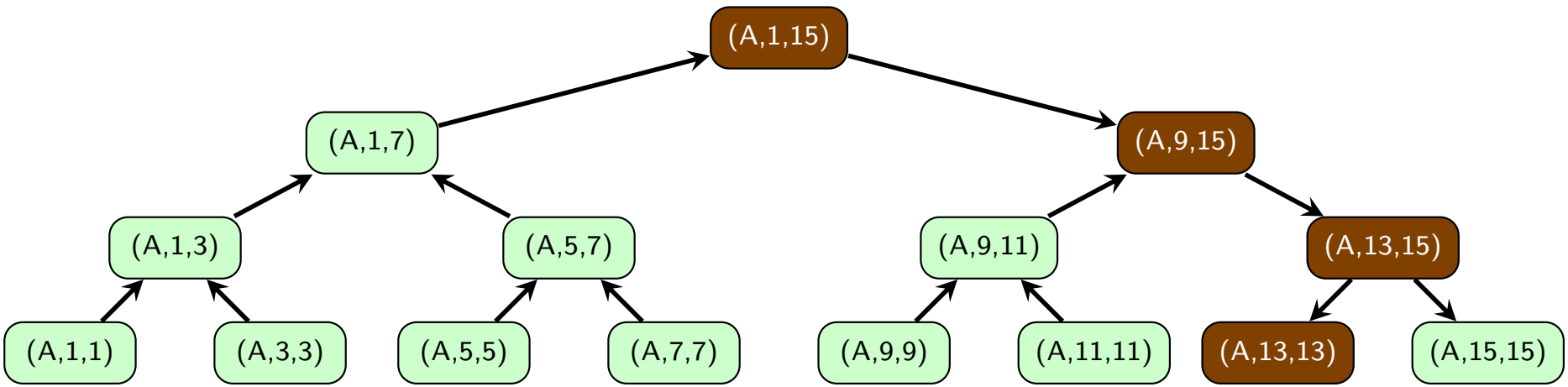


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

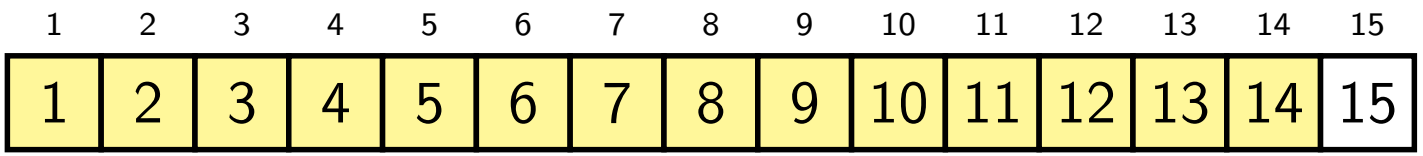


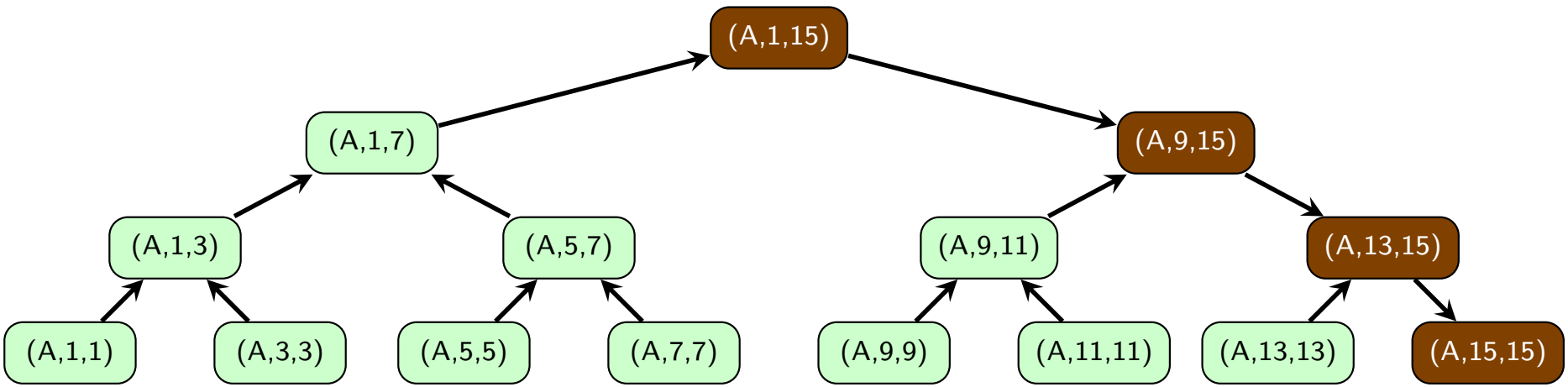




QuickSort: Best-Case Behavior

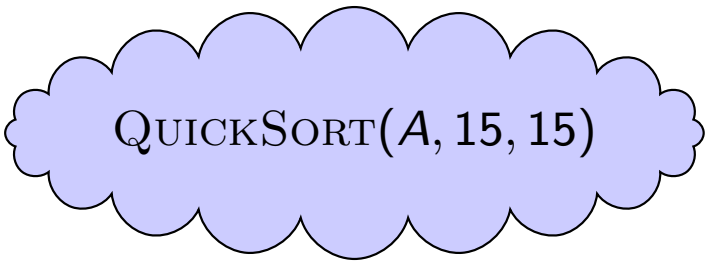
```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

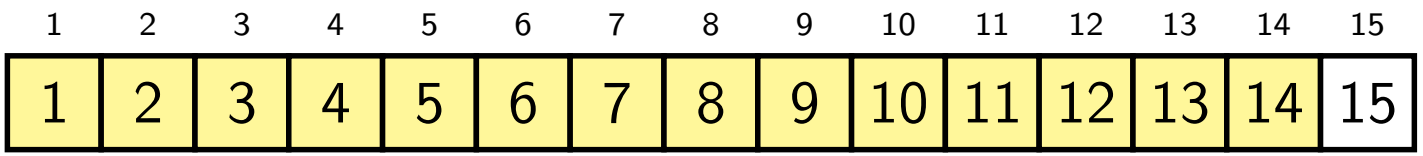


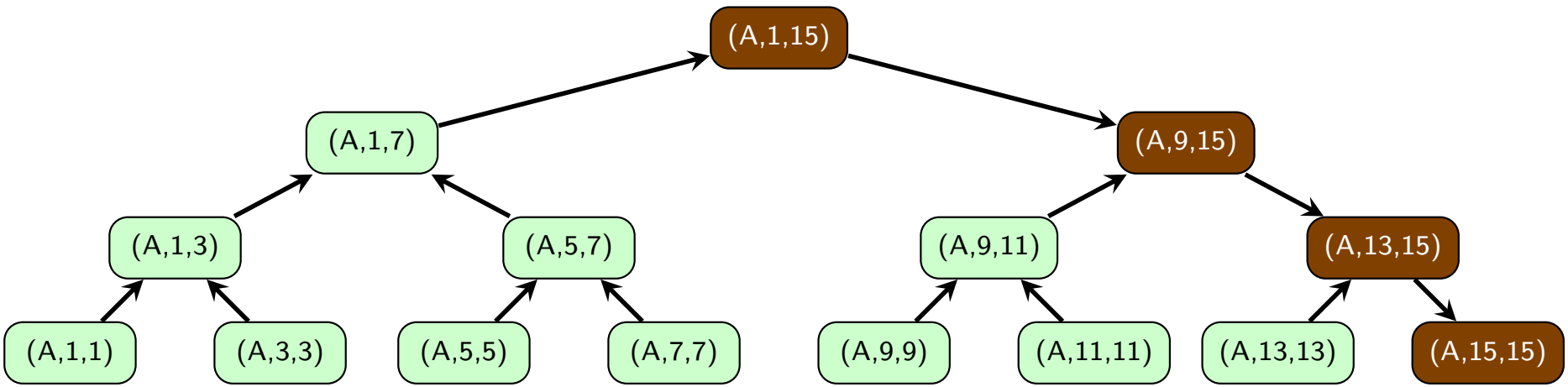


QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

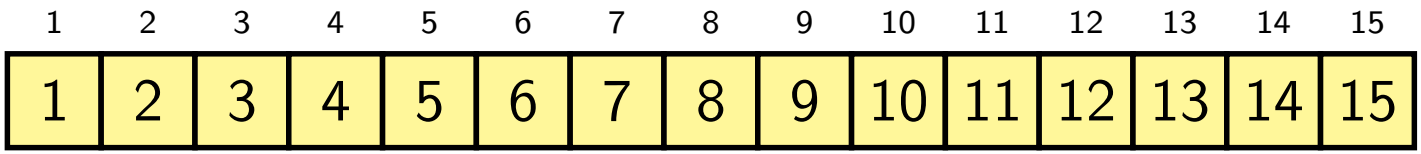


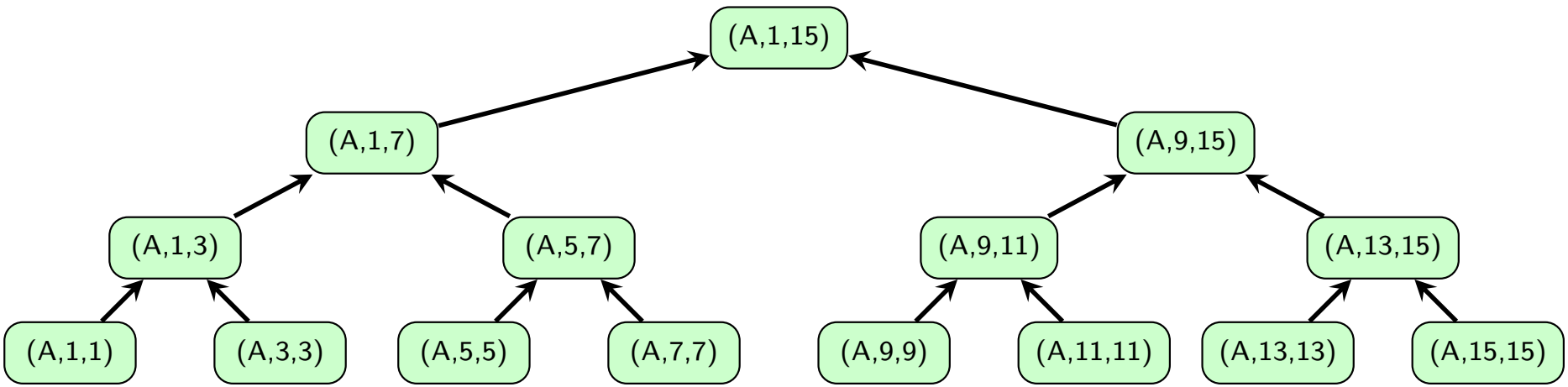




QuickSort: Best-Case Behavior

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```





Best-Case Complexity

- $a = 2$: number of subproblems.
- $\frac{n}{2}$ $(b = 2)$: size of each subproblem.
- $D(n) = \Theta(n)$: cost of the Partition procedure
- $C(n) = \Theta(1)$: cost of combining the sub-solutions
- Base case $n_0 \leq 1$
- $T(n) =$
$$\begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

QuickSort(A, p, r)

```
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

Best-Case Complexity

- $a = 2$: number of subproblems.
- $\frac{n}{2}$ $(b = 2)$: size of each subproblem.
- $D(n) = \Theta(n)$: cost of the Partition procedure
- $C(n) = \Theta(1)$: cost of combining the sub-solutions
- Base case $n_0 \leq 1$
- $T(n) =$
$$\begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

$$T(n) = \Theta(n \cdot log_2(n))$$

Average-Case Complexity

Assume that the partition always produces 9-to-1 splits (which is a poor average behavior), then we obtain the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

Average-Case Complexity

Assume that the partition always produces 9-to-1 splits (which is a poor average behavior), then we obtain the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

$$T(n) = \Theta(n \cdot log_2(n))$$

Average-Case Complexity

Assume that the partition always produces that one half is of the size $d \cdot n$ for some $d : 0 < d < 1$, then we obtain the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(dn) + T((1-d)n) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

Average-Case Complexity

Assume that the partition always produces that one half is of the size $d \cdot n$ for some $d : 0 < d < 1$, then we obtain the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(dn) + T((1-d)n) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

$$T(n) = \Theta(n \cdot log_2(n))$$

Average-Case Complexity

Assume that the partition always produces that one half is of the size $d \cdot n$ for some $d : 0 < d < 1$, then we obtain the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(dn) + T((1-d)n) + \Theta(n) & \text{if } n > 1 \end{cases}$$

```
QuickSort(A, p, r)
1  if p < r
2     then q ← Partition(A, p, r)
3          QuickSort(A, p, q − 1)
4          QuickSort(A, q + 1, r)
```

$$T(n) = \Theta(n \cdot log_2(n))$$

Hence Quicksort takes $\Theta(n \cdot log_2(n))$ time in the average case

Variants of Quicksort

- Pivot Selection:
  - Select a randomly chosen element from the subarray $A[p..r]$ (Instead of always using $A[r]$) as pivot)

```
RandomizedPartition(A, p, r)
1  i ← Random(p, r)
2  swap A[r] ↔ A[i]
3  return Partition(A, p, r)
```

```
RandomizedQuickSort(A, p, r)
1  if p < r
2     then q ← RandomizedPartition(A, p, r)
3          RandomizedQuickSort(A, p, q − 1)
4          RandomizedQuickSort(A, q + 1, r)
```

Variants of Quicksort

- Pivot Selection:
  - Choose the pivot as the median of a set of 3 elements

```
MedianPartition(A, p, r)
1  i ← Median(p, ⌈(p+r)/2⌉, r)
2  swap A[r] ↔ A[i]
3  return Partition(A, p, r)
```

```
MedianQuickSort(A, p, r)
1  if p < r
2     then q ← MedianPartition(A, p, r)
3          MedianQuickSort(A, p, q − 1)
4          MedianQuickSort(A, q + 1, r)
```

Variants of Quicksort

- Small sized subarrays:
  - Quicksort is very heavy for small-sized arrays
  - Use a naive sorting algorithm (e.g., Insertion sort) for arrays of length less than k ($k \approx 20$)

```
QuickSort(A, p, r)
1  if r − p + 1 ≤ Cutoff
2     then InsertionSort(A,p,r)
3     else q ← Partition(A, p, r)
4          QuickSort(A, p, q − 1)
5          QuickSort(A, q + 1, r)
```

Conclusion

- Average time-complexity is $\Theta(n \log(n))$ for many realistic distributions.
- The worst-case complexity is $\Theta(n^2)$, but very rare with a good choice of the pivot element.
- Sort is in place.
- Not stable in the standard implementation.
- In practice QuickSort is usually a bit faster than MergeSort, but this is highly context dependent.