COMP0005 (Algorithms)

Quicksort

Vasileios Lampos
Computer Science, UCL

Slides (with potential **revisions**)
lampos.net/slides/quicksort2019.pdf

@lampos
lampos.net

About this lecture

- **Quicksort** (yet another sorting algorithm)
  - Description
  - Performance analysis

- **Material**
  - Cormen, Leiserson, Rivest and Stein. *Introduction to Algorithms*. MIT Press, 3rd Edition, 2009 (mainly Chapter 7)
  - Alternative slides at https://algs4.cs.princeton.edu/lectures/ (Sedgewick and Wayne)

Quicksort divides & conquers

Quicksort divides & conquers

Given an array A with n elements, $A[1 \ldots n]$:

- **DIVIDE** (step 1)

  Partition, i.e. *re-arrange the elements of*, array $A[1 \ldots n]$ so that for some element $A[q]$:

  1. all elements on the left of $A[q]$, i.e. $A[1 \ldots q-1]$, are less than or equal to $A[q]$, and
  2. all elements on the right of $A[q]$, i.e. $A[q+1 \ldots n]$, are greater than or equal to $A[q]$.

Quicksort divides & conquers

Given an array A with n elements, $A[1 \ldots n]$:

- **DIVIDE** (step 1)
  Partition, i.e. *re-arrange the elements of*, array $A[1 \ldots n]$ so that for some element $A[q]$:
  1. all elements on the left of $A[q]$, i.e. $A[1 \ldots q{-}1]$, are less than or equal to $A[q]$, and
  2. all elements on the right of $A[q]$, i.e. $A[q{+}1 \ldots n]$, are greater than or equal to $A[q]$.

- **CONQUER** (step 2)
  Sort sub-arrays $A[1 \ldots q{-}1]$ and $A[q{+}1 \ldots n]$ by recursive executions of step 1.

Quicksort divides & conquers

Given an array A with n elements, $A[1 \ldots n]$:

- **DIVIDE** (step 1)
  Partition, i.e. *re-arrange the elements of*, array $A[1 \ldots n]$ so that for some element $A[q]$:
  1. all elements on the left of $A[q]$, i.e. $A[1 \ldots q{-}1]$, are less than or equal to $A[q]$, and
  2. all elements on the right of $A[q]$, i.e. $A[q{+}1 \ldots n]$, are greater than or equal to $A[q]$.

- **CONQUER** (step 2)
  Sort sub-arrays $A[1 \ldots q{-}1]$ and $A[q{+}1 \ldots n]$ by recursive executions of step 1.

- **COMBINE** (step 3)
  Just by joining the sorted sub-arrays we obtain a sorted array.

Quicksort divides & conquers

Given an array A with n elements, $A[1 \ldots n]$:

- **DIVIDE** (step 1)

  Partition, i.e. *re-arrange the elements of*, array $A[1 \ldots n]$ so that for some element $A[q]$:

  1. all elements … are less than or equal to $A[$…
  2. all elements … $n]$, are greater than or equ…

- **CONQUER** (s…

  Sort sub-arrays $A[1 \ldots q-1]$ and $A[q+1 \ldots n]$ by recursive executions of step 1.

- **COMBINE** (step 3)

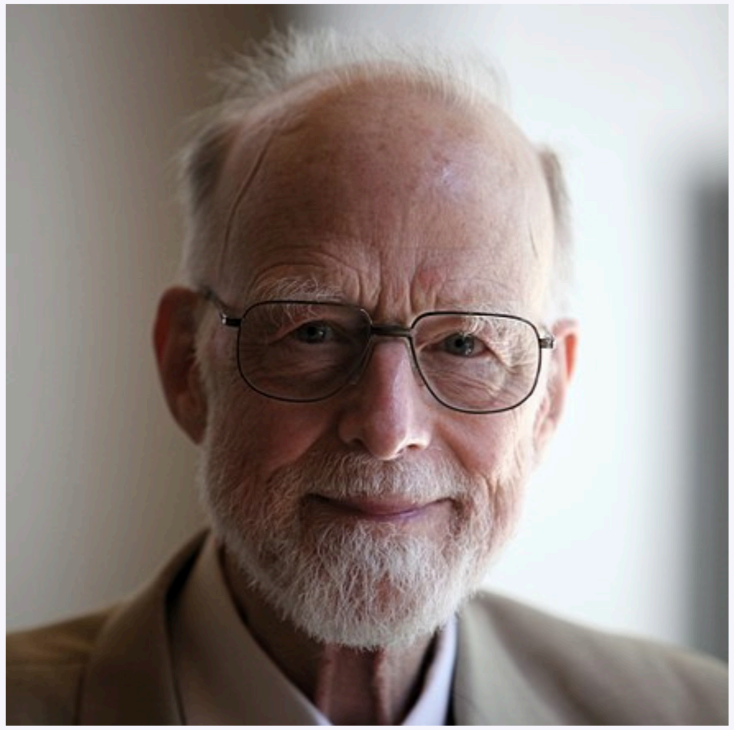
  Just by joining the sorted sub-arrays we obtain a sorted array.

Note:

- We will assume that the elements of A are distinct.
- We will be sorting the elements of A in an ascending order.

Quicksort was...

- invented by Tony Hoare in 1959
- published in 1961 (paper)



Tony Hoare in 2011

Born Charles Antony Richard Hoare
11 January 1934 (age 84)

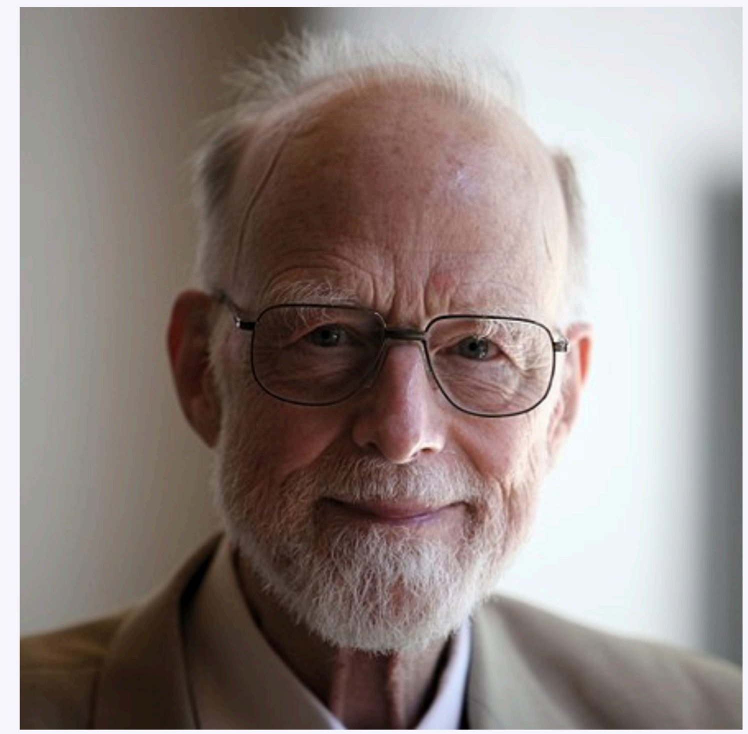
from Wikipedia

Quicksort was...

- invented by Tony Hoare in 1959
- published in 1961 (paper)

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure  quicksort (A,M,N);  value M,N;
              array A;  integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin      integer I,J;
           if M < N then begin partition (A,M,N,I,J);
                               quicksort (A,M,J);
                               quicksort (A, I, N)
                         end
end        quicksort
```



Tony Hoare in 2011

Born	Charles Antony Richard Hoare 11 January 1934 (age 84)

from Wikipedia

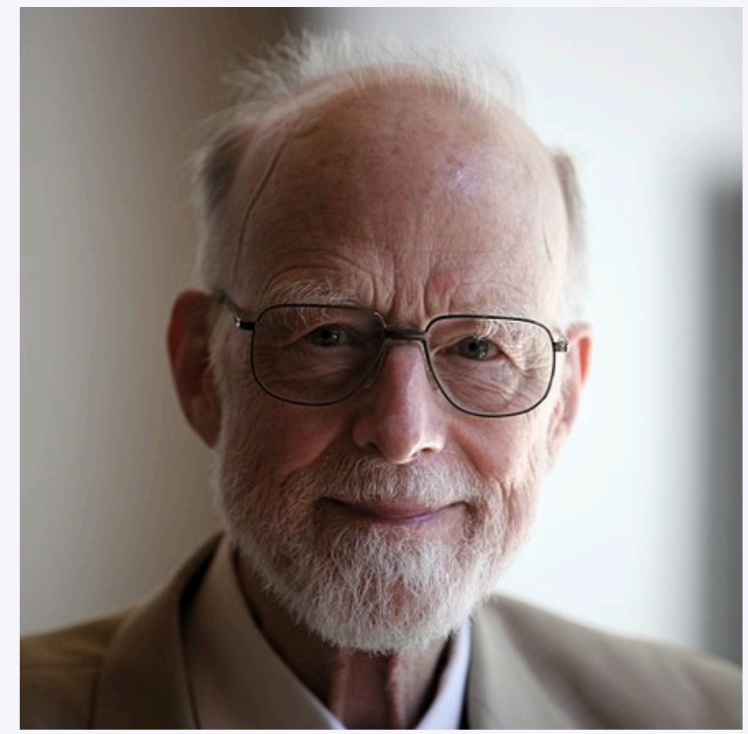
Quicksort was...

- invented by Tony Hoare in 1959
- published in 1961 (paper)

ALGORITHM 64
QUICKSORT
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure   quicksort (A,M,N);  value M,N;
              array A;  integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin       integer I,J;
            if M < N then begin partition (A,M,N,I,J);
                                quicksort (A,M,J);
                                quicksort (A, I, N)
                          end
end         quicksort
```

that was the entire paper!



Tony Hoare in 2011

Born	Charles Antony Richard Hoare 11 January 1934 (age 84)

from Wikipedia

Quicksort was...

- invented by Tony Hoare in 1959
- published in 1961 (paper)

ALGORITHM 64
QUICKSORT
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure   quicksort (A,M,N);  value M,N;
               array A;  integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin        integer I,J;
             if M < N then begin partition (A,M,N,I,J);
                                 quicksort (A,M,J);
                                 quicksort (A, I, N)
                             end
end          quicksort
```

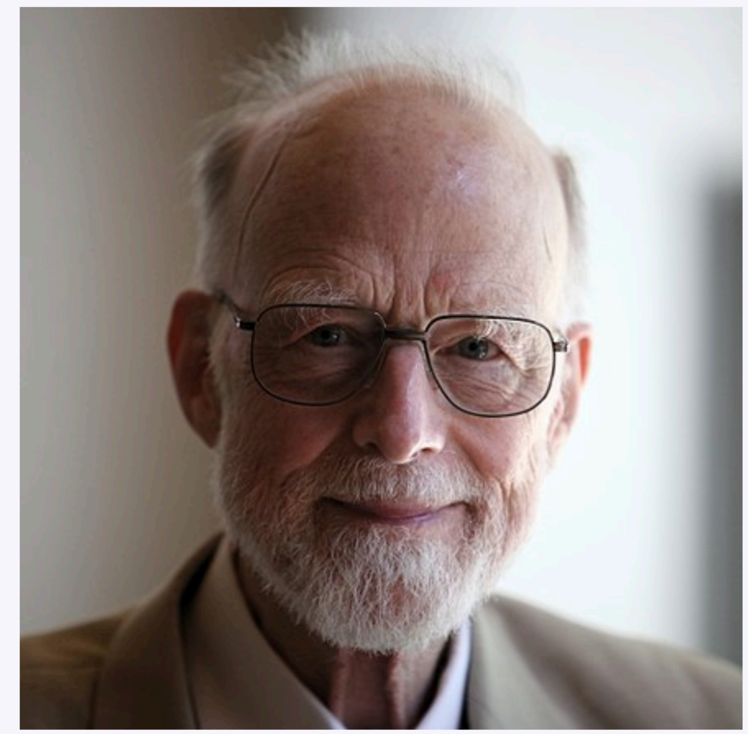
that was the entire paper!

- published with an analysis in 1962 (paper)

Table 1

NUMBER OF ITEMS	MERGE SORT	QUICKSORT
500	2 min 8 sec	1 min 21 sec
1,000	4 min 48 sec	3 min 8 sec
1,500	8 min 15 sec*	5 min 6 sec
2,000	11 min 0 sec*	6 min 47 sec

* These figures were computed by formula, since they cannot be achieved on the 405 owing to limited store size.



Tony Hoare in 2011

Born	Charles Antony Richard Hoare 11 January 1934 (age 84)

from Wikipedia

Quicksort was...

- invented by Tony Hoare in 1959
- published in 1961 (paper)

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure   quicksort (A,M,N);  value M,N;
              array A;  integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin        integer I,J;
             if M < N then begin partition (A,M,N,I,J);
                                 quicksort (A,M,J);
                                 quicksort (A, I, N)
                           end
end          quicksort
```

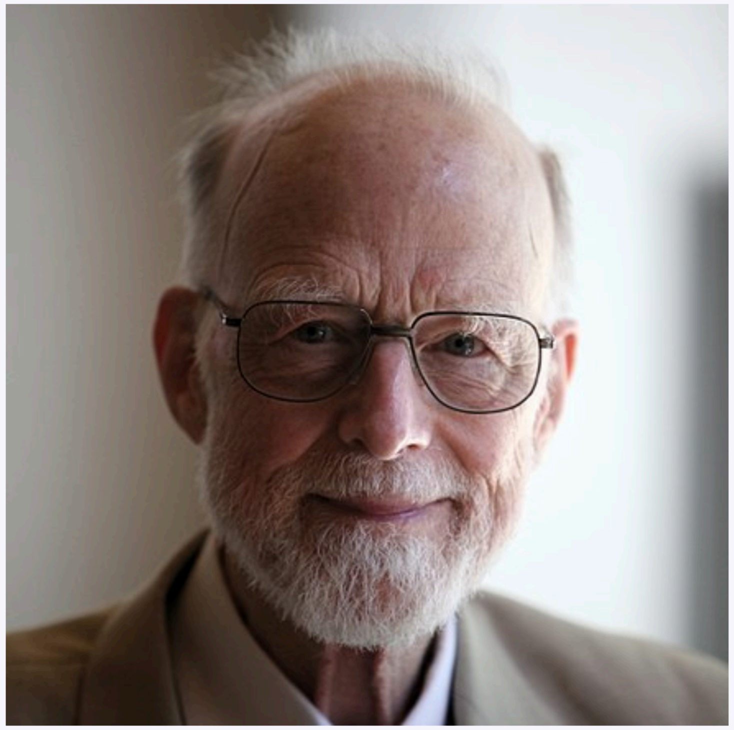
that was the entire paper!

- published with an analysis in 1962 (paper)

Table 1

NUMBER OF ITEMS	MERGE SORT	QUICKSORT
500	2 min 8 sec	1 min 21 sec
1,000	4 min 48 sec	3 min 8 sec
1,500	8 min 15 sec*	5 min 6 sec
2,000	11 min 0 sec*	6 min 47 sec

* These figures were computed by formula, since they cannot be achieved on the 405 owing to limited store size.



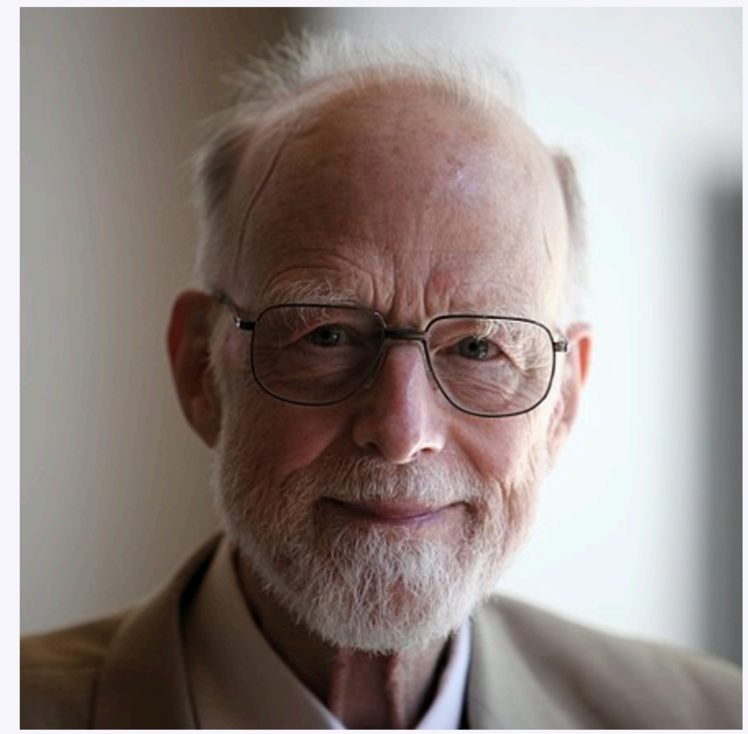
Tony Hoare in 2011

Born Charles Antony Richard Hoare
11 January 1934 (age 84)

from Wikipedia

Quicksort…

- is still being used (in principle, i.e. its optimised versions)



Tony Hoare in 2011

Born Charles Antony Richard Hoare
11 January 1934 (age 84)

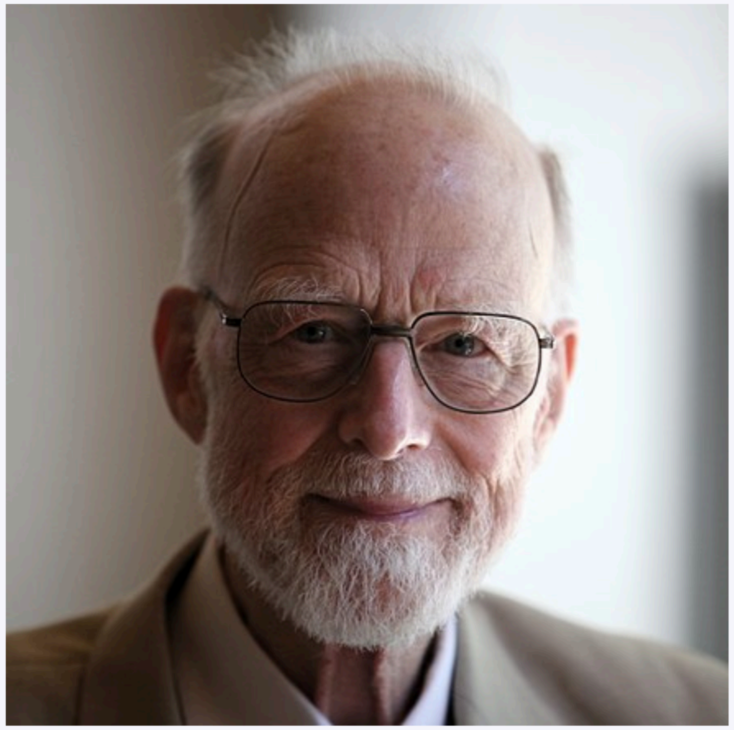
from Wikipedia

Quicksort...

- is still being used (in principle, i.e. its optimised versions)

- is efficient
  — $O(n \log n)$ on average
  — $\Theta(n \log n)$ best case
  — $\Theta(n^2)$ worst case

  (for an array with n elements)



Tony Hoare in 2011

Born Charles Antony Richard Hoare
11 January 1934 (age 84)

from Wikipedia

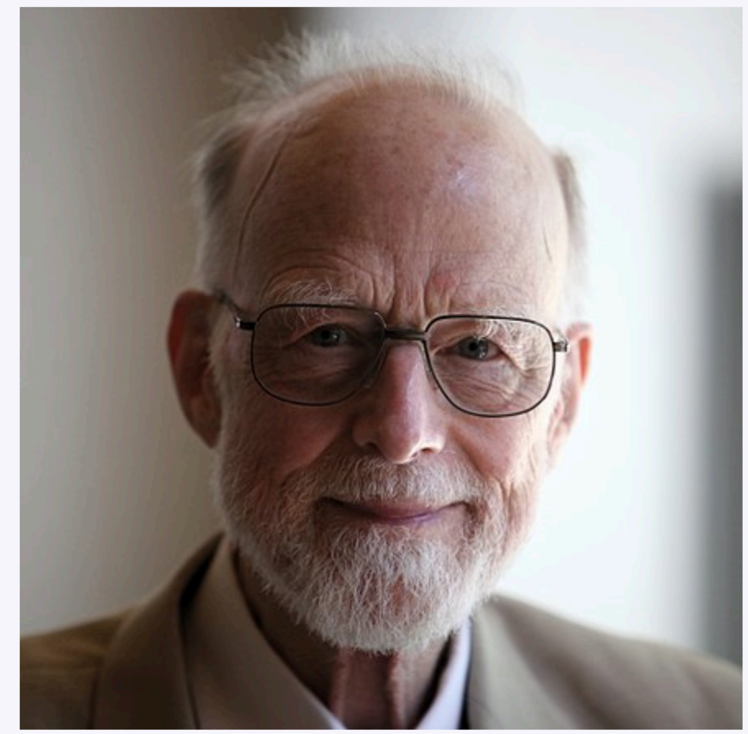
Quicksort…

- is still being used (in principle, i.e. its optimised versions)

- is efficient
  — $O(n \log n)$ on average
  — $\Theta(n \log n)$ best case
  — $\Theta(n^2)$ worst case

  (for an array with n elements)

- requires a small amount of memory (*in-place* algorithm)



Tony Hoare in 2011

Born Charles Antony Richard Hoare
11 January 1934 (age 84)

from Wikipedia

Quicksort

both p, r are array indices

	p						r	
…	6	5	1	…	22	9	2	…

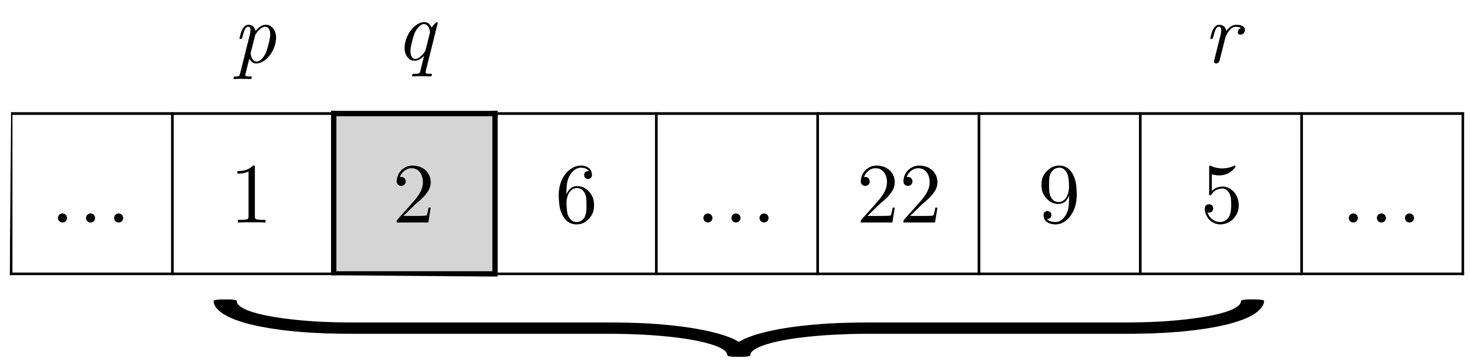
sub-array $A[p \ldots r]$

QUICKSORT(A, p, r)

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

Quicksort

p, r, q are array indices

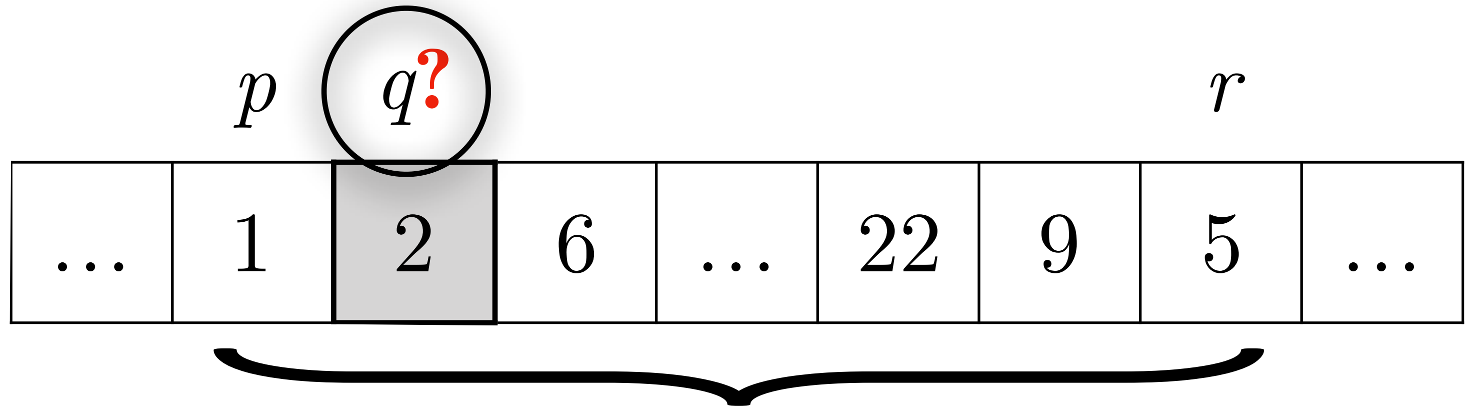


sub-array $A[p \ldots r]$

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

Quicksort — Partition

p, r, q are array indices



sub-array $A[p \ldots r]$

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Partition is the central sorting operation of quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

Step-by-step example

i	p,j					r,x
	6	5	1	3	2	4

QUICKSORT(A, p, r)

if $p < r$
  $q = $ PARTITION(A, p, r)
  QUICKSORT$(A, p, q - 1)$
  QUICKSORT$(A, q + 1, r)$

PARTITION(A, p, r)

$x = A[r]$
$i = p - 1$
for $j = p$ **to** $r - 1$
  if $A[j] \leq x$
    $i = i + 1$
    exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$

Step-by-step example

i	p,j					r,x
	6	5	1	3	2	4

pivot element

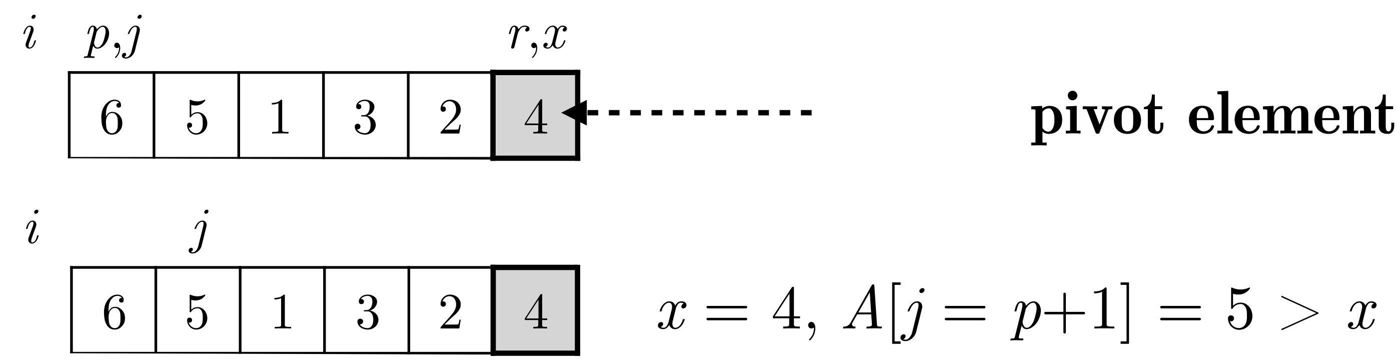
QUICKSORT(A, p, r)

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

PARTITION(A, p, r)

```
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Step-by-step example

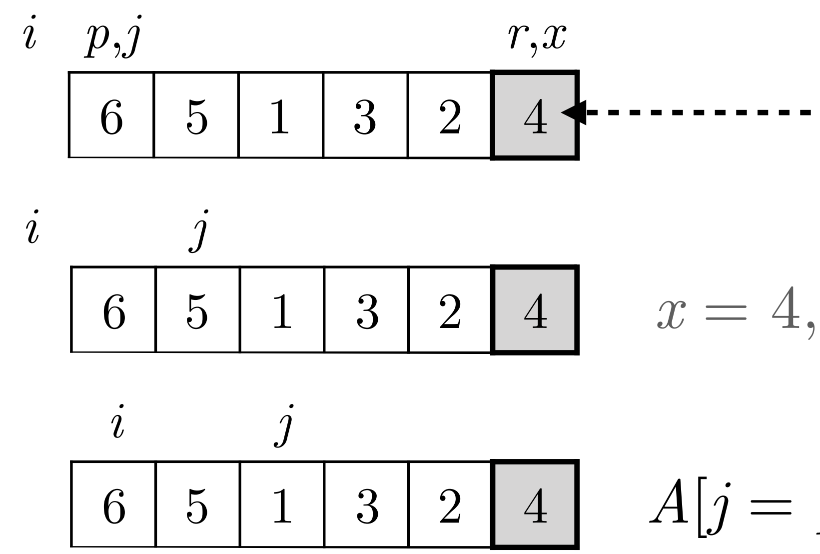


$x = 4,\ A[j = p{+}1] = 5 > x$

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Step-by-step example



pivot element

$x = 4,\ A[j = p{+}1] = 5 > x$

$A[j = p{+}2] = 1 < x,\ i = i{+}1$

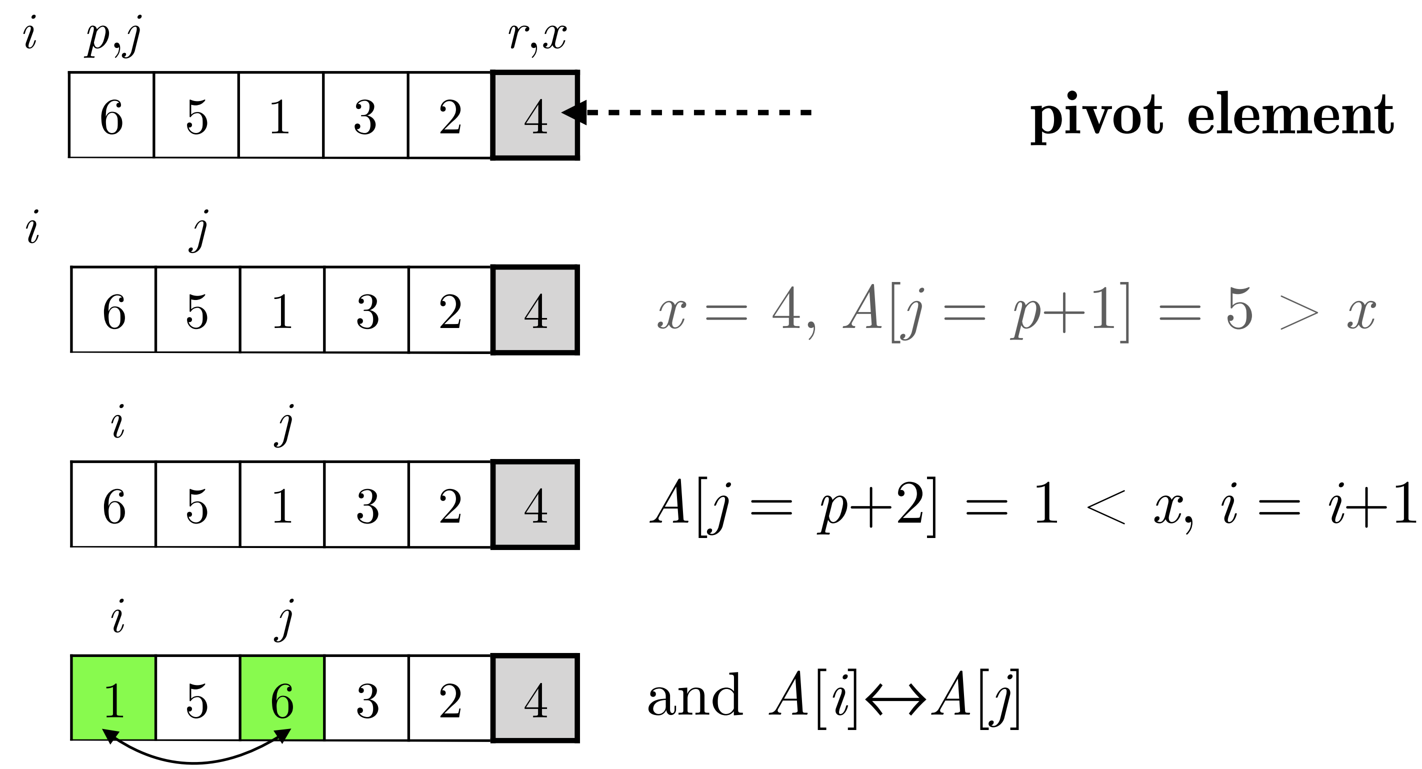
QUICKSORT(A, p, r)

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

PARTITION(A, p, r)

```
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Step-by-step example




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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

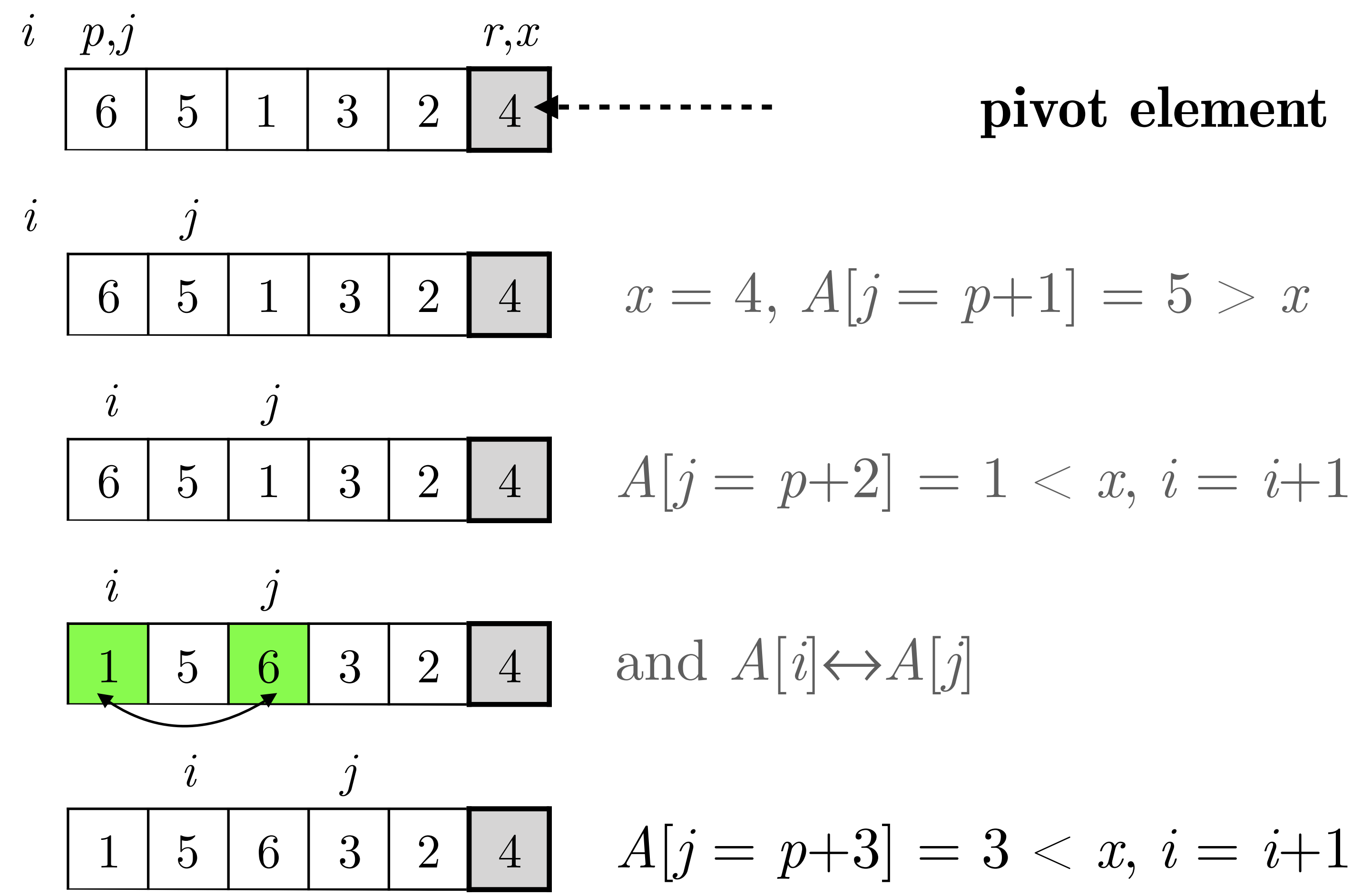
Step-by-step example

QUICKSORT(A, p, r)

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

PARTITION(A, p, r)

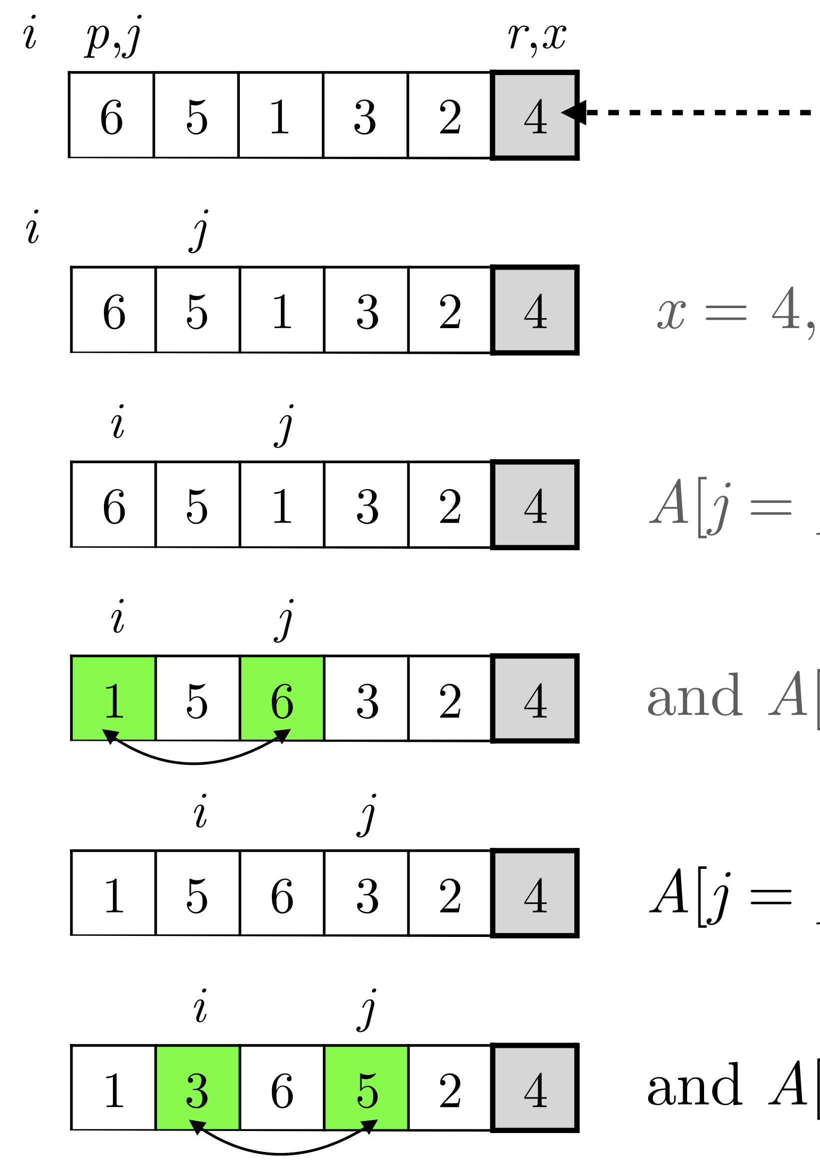
```
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



Step-by-step example

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

```
PARTITION(A, p, r)
    x = A[r]
    i = p − 1
    for j = p to r − 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```



pivot element

$x = 4,\ A[j = p{+}1] = 5 > x$

$A[j = p{+}2] = 1 < x,\ i = i{+}1$

and $A[i] \leftrightarrow A[j]$

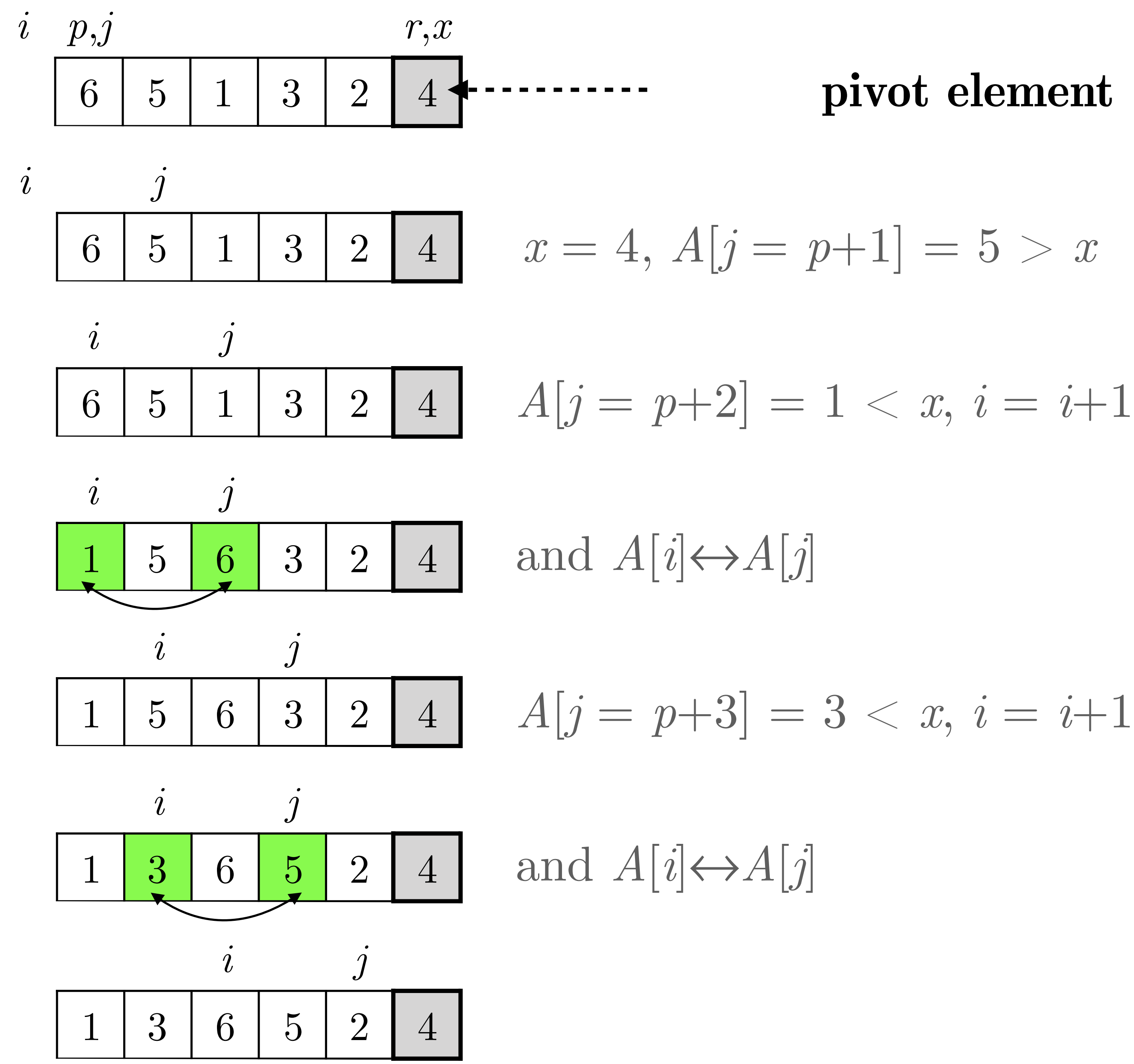
$A[j = p{+}3] = 3 < x,\ i = i{+}1$

and $A[i] \leftrightarrow A[j]$

Step-by-step example

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



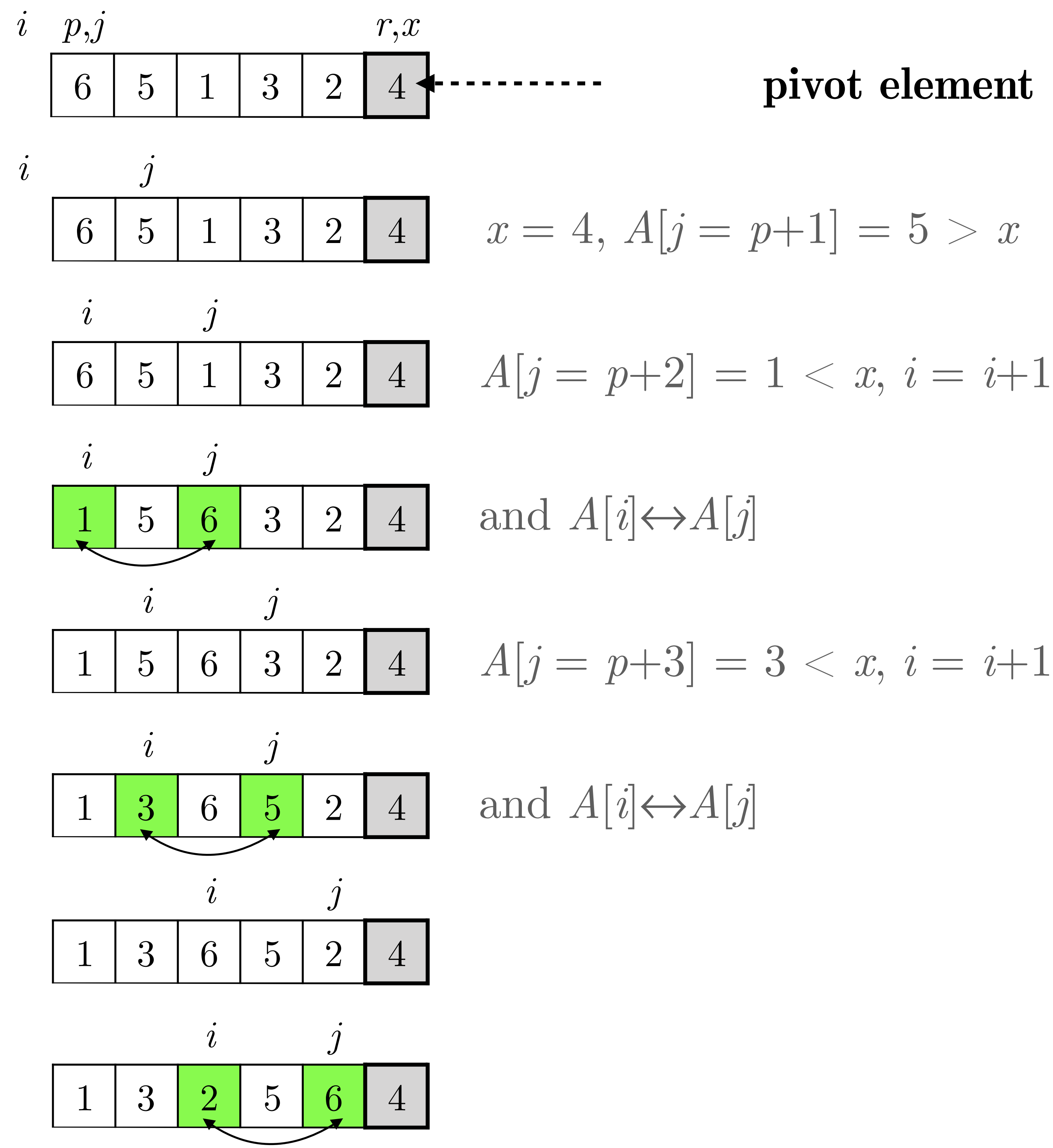
Step-by-step example

QUICKSORT(A, p, r)

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

PARTITION(A, p, r)

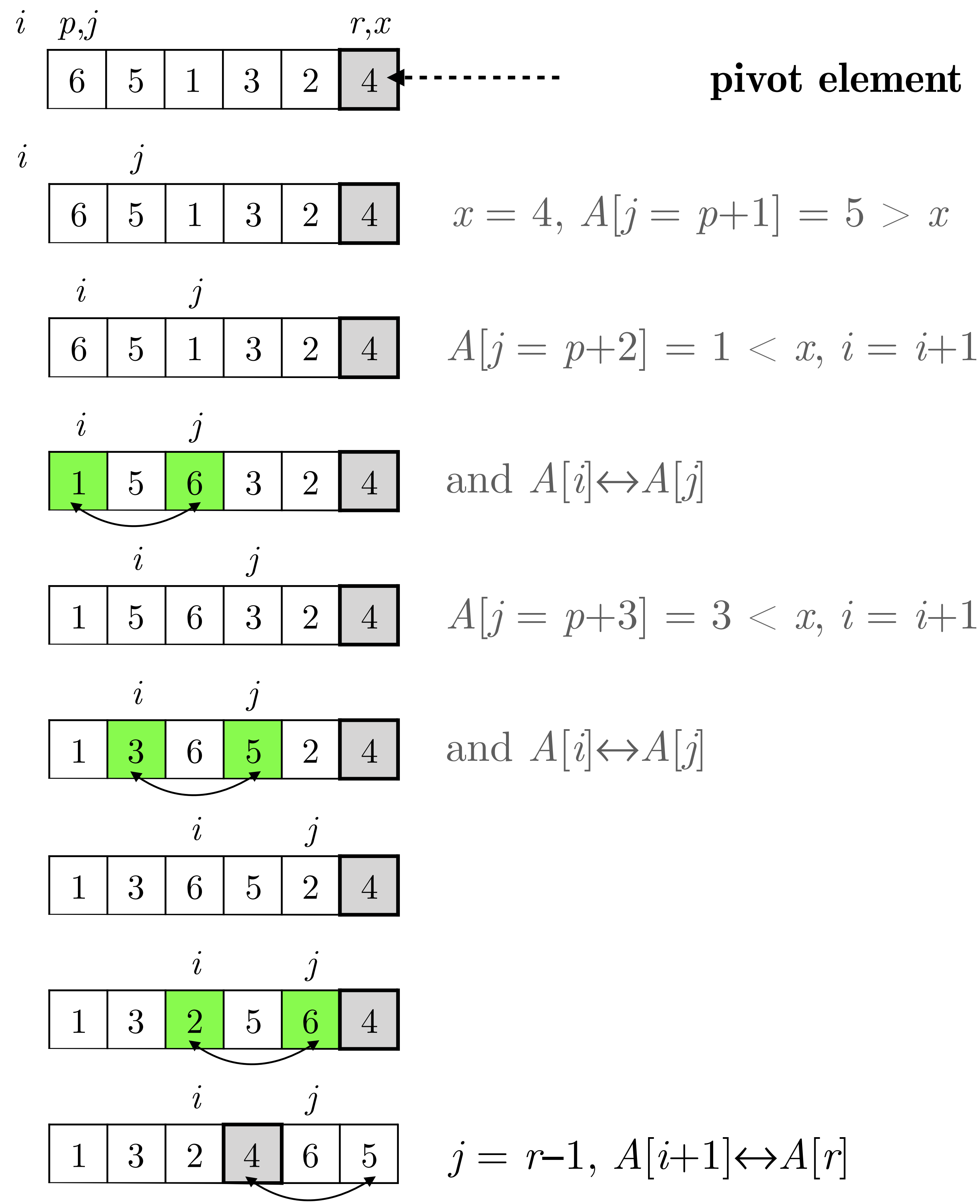
```
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



Step-by-step example

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

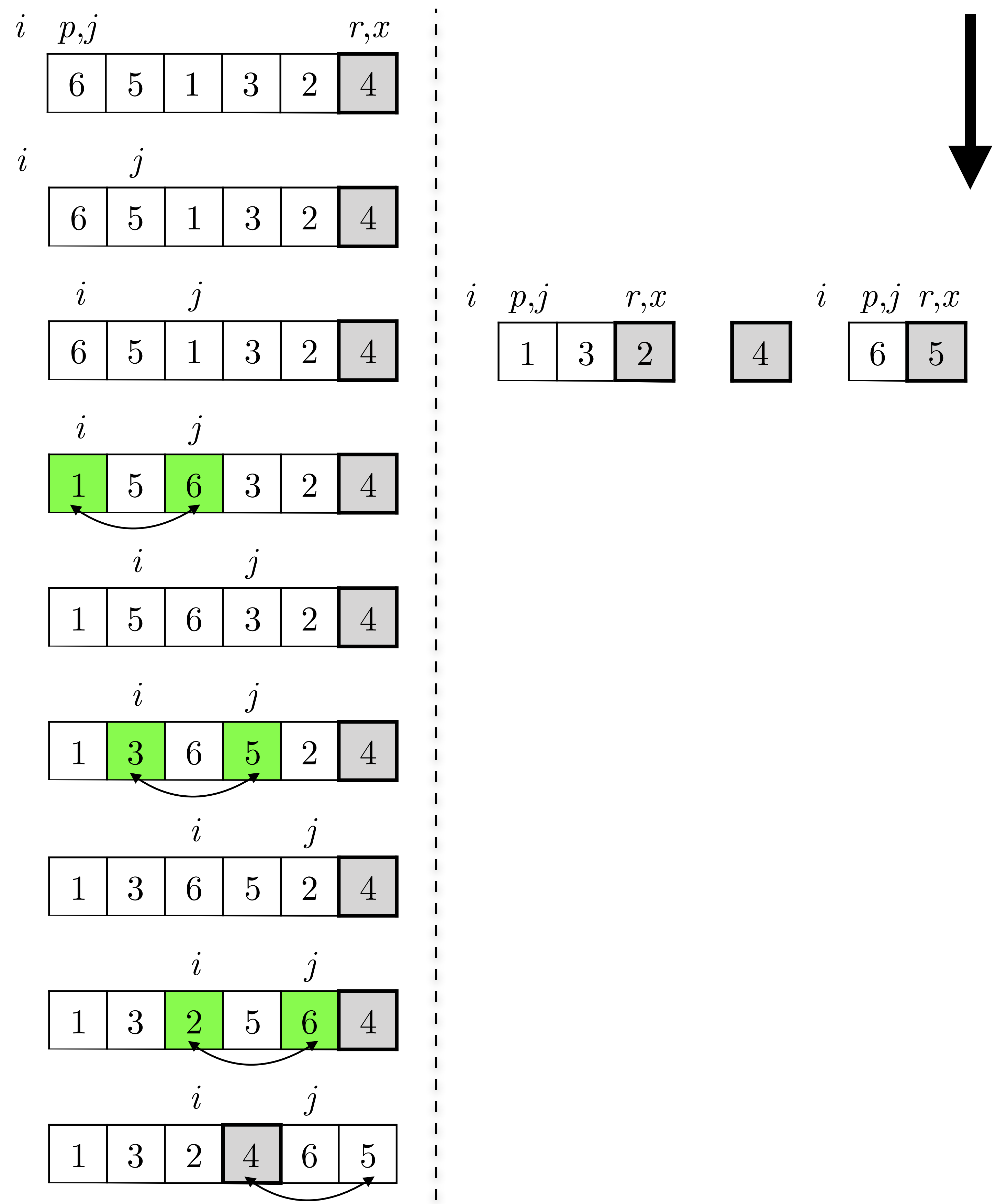
```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



Step-by-step example

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



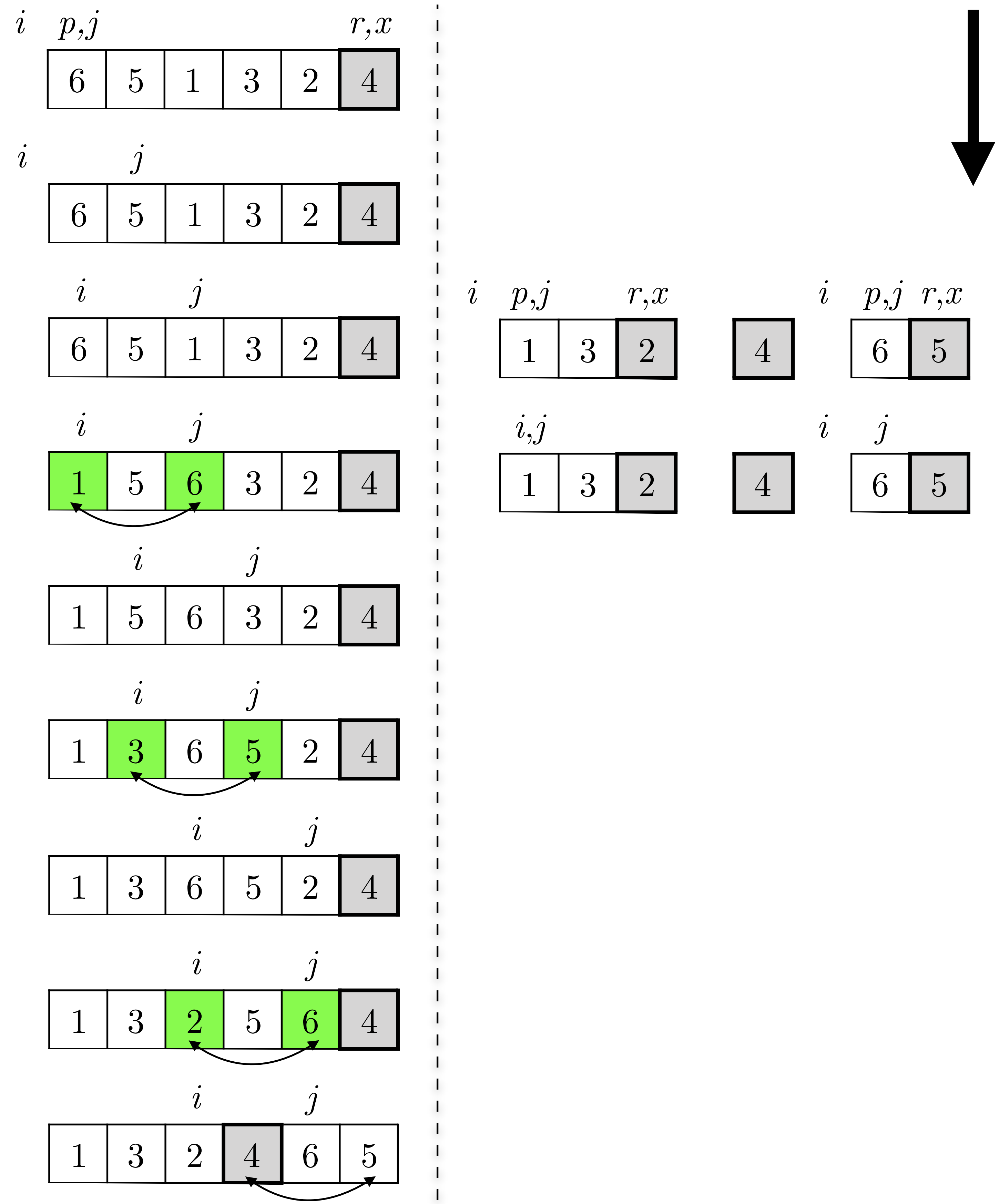
Step-by-step example

QUICKSORT(A, p, r)

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

PARTITION(A, p, r)

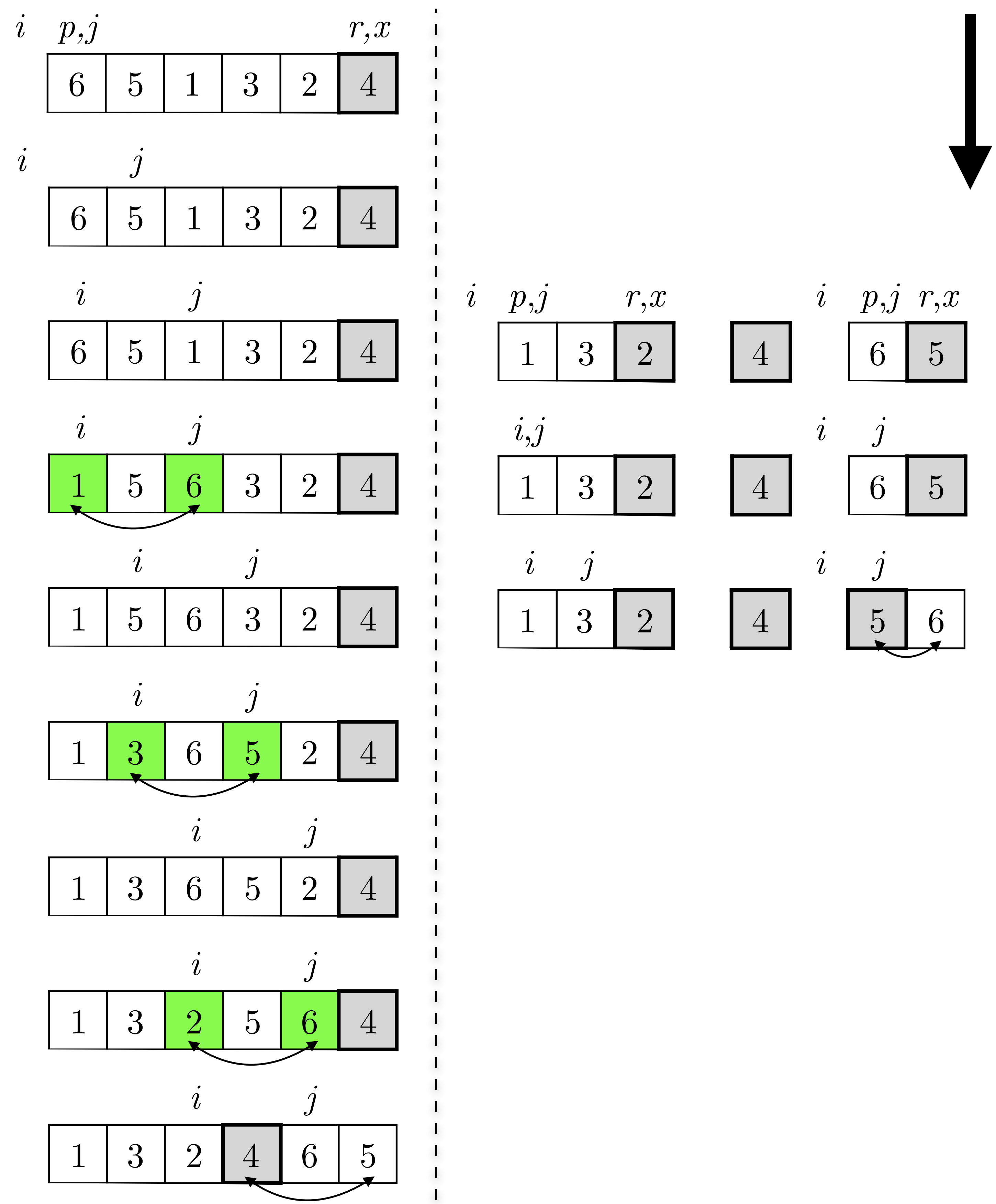
```
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



Step-by-step example

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

```
PARTITION(A, p, r)
    x = A[r]
    i = p − 1
    for j = p to r − 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```



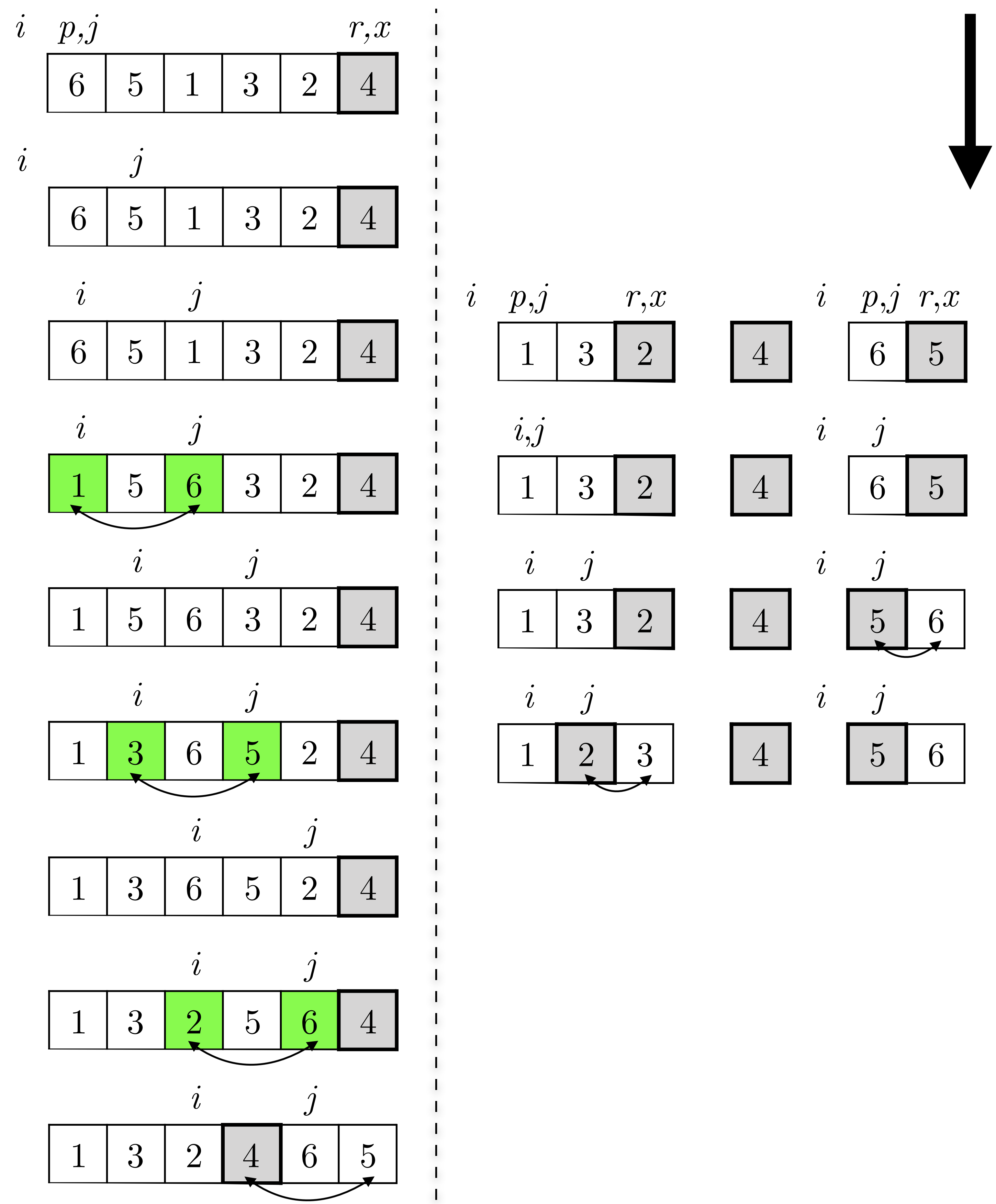
Step-by-step example

QUICKSORT(A, p, r)

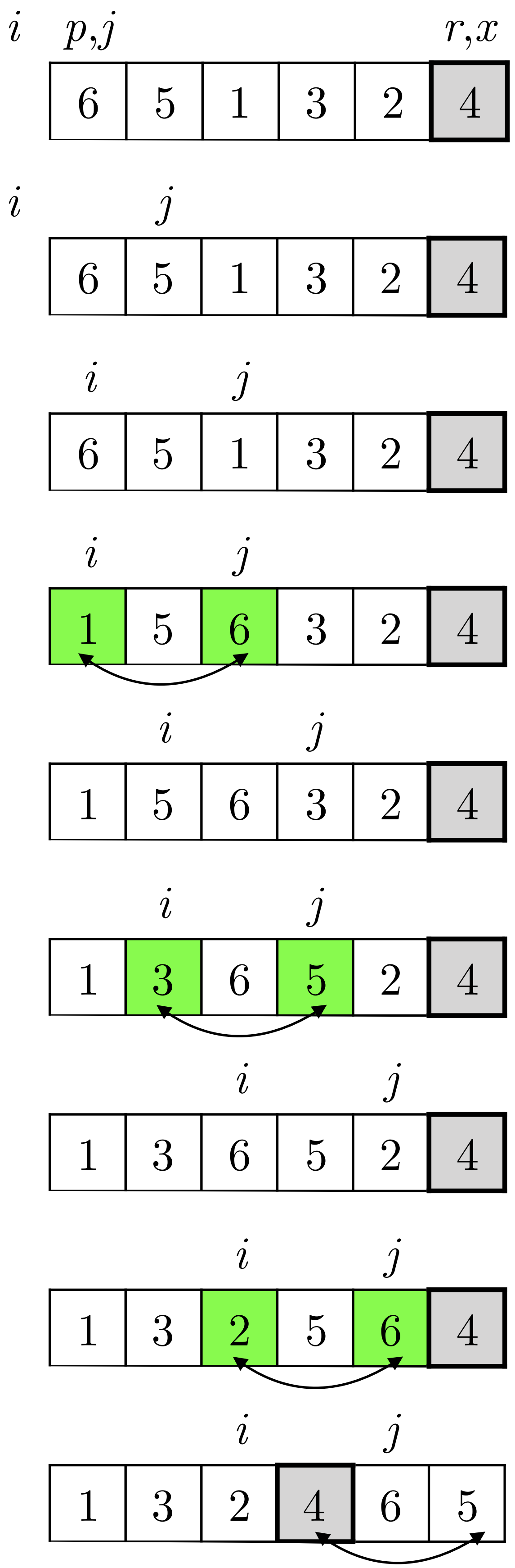
if $p < r$
 $q = $ PARTITION(A, p, r)
 QUICKSORT$(A, p, q - 1)$
 QUICKSORT$(A, q + 1, r)$

PARTITION(A, p, r)

$x = A[r]$
$i = p - 1$
for $j = p$ **to** $r - 1$
 if $A[j] \leq x$
  $i = i + 1$
  exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$



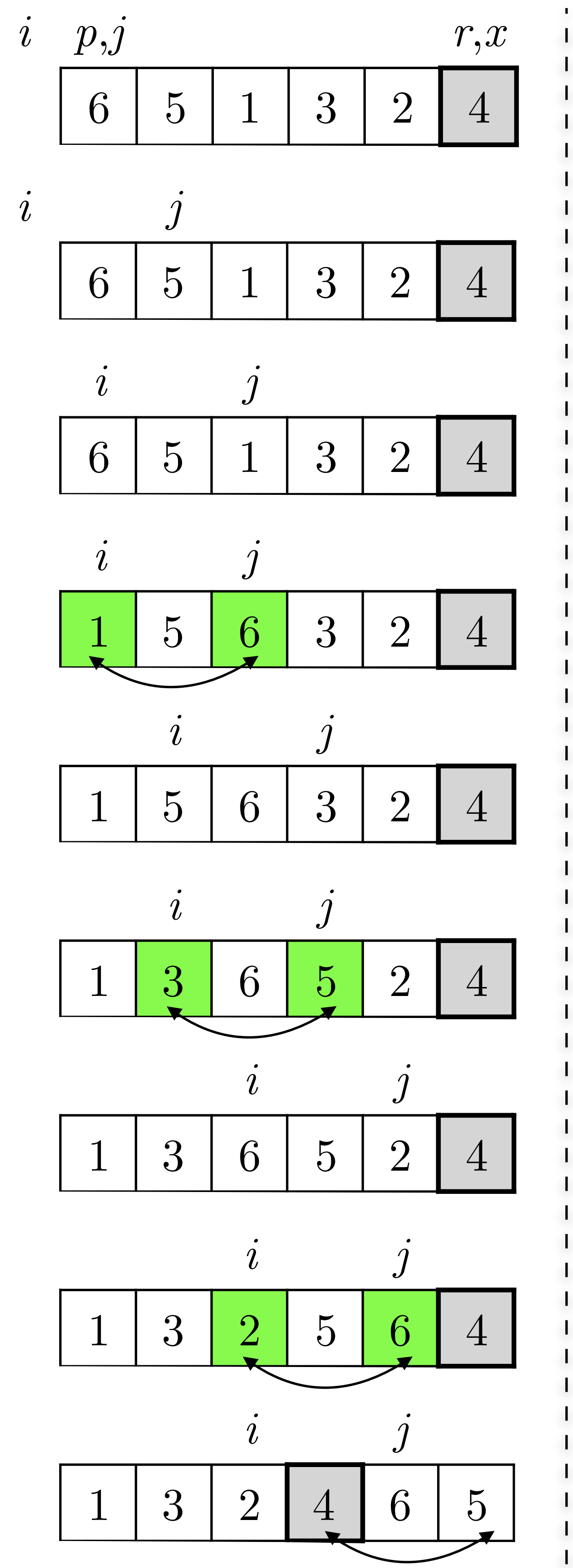
Why does quicksort work?



- As i goes through the array from left to right, no element greater than the pivot element ($= 4$) is left behind it. When such element is identified, it is swapped.

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Why does quicksort work?



- As i goes through the array from left to right, no element greater than the pivot element ($= 4$) is left behind it. When such element is identified, it is swapped.
- Elements i+1 to j–1 are always greater than the pivot element.

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort's performance

Quicksort's performance

- The performance is affected by the **choice of the pivot element** during partitioning: balanced vs. unbalanced outcome

Quicksort's performance

- The performance is affected by the **choice of the pivot element** during partitioning: balanced vs. unbalanced outcome

- **Worst case:** $\Theta(n^2)$
  - when partitioning is always **completely unbalanced**, i.e. the choice of pivot generates sub-arrays that always have n–1 and 0 elements, respectively
  - when the array is already sorted

Quicksort's performance

- The performance is affected by the **choice of the pivot element** during partitioning: balanced vs. unbalanced outcome

- **Worst case:** $\Theta(n^2)$
  - — when partitioning is always **completely unbalanced**, i.e. the choice of pivot generates sub-arrays that always have n–1 and 0 elements, respectively
  - — when the array is already sorted

- **Best case:** $\Theta(n \log n)$
  - — when partitioning is always **fairly balanced**, i.e. the choice of pivot generates sub-arrays that always have $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil - 1$ elements, respectively

Step-by-step example for worst case

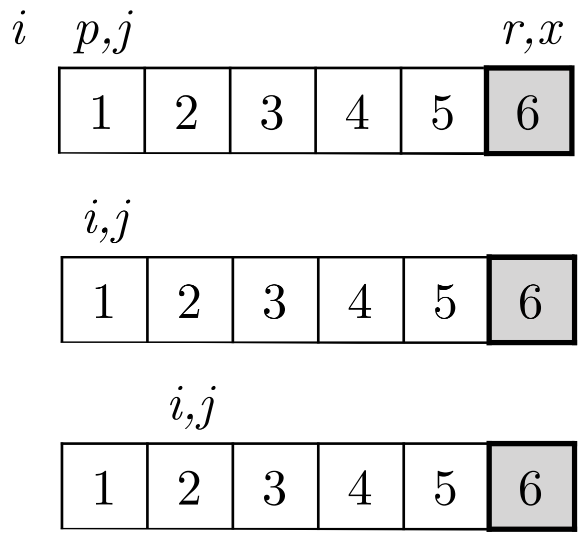
i	p,j					r,x
	1	2	3	4	5	6

Step-by-step example for worst case

i	p,j					r,x
	1	2	3	4	5	6

i,j					
1	2	3	4	5	6

Step-by-step example for worst case



Step-by-step example for worst case

i p, j r, x

1	2	3	4	5	6

i, j

1	2	3	4	5	6

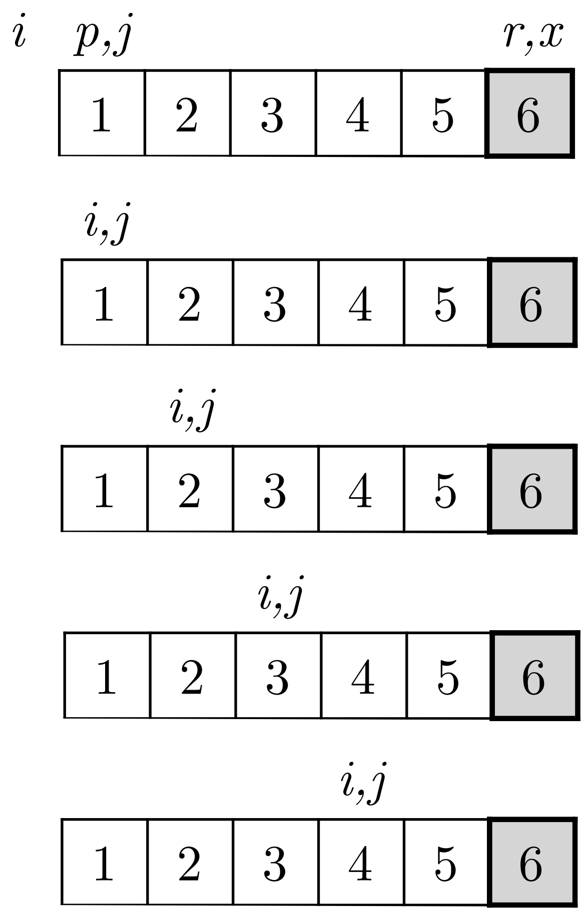
i, j

1	2	3	4	5	6

i, j

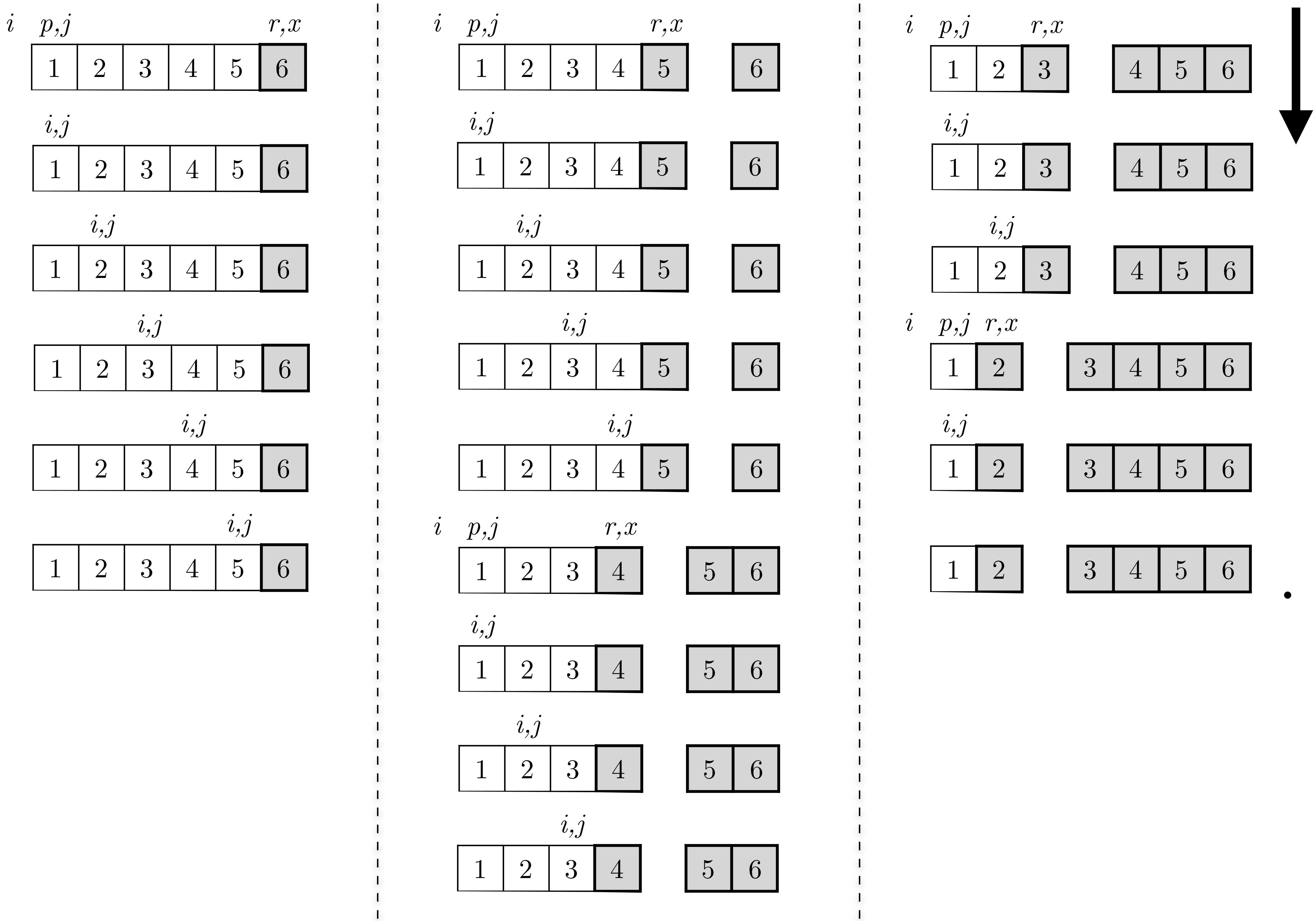
1	2	3	4	5	6

Step-by-step example for worst case

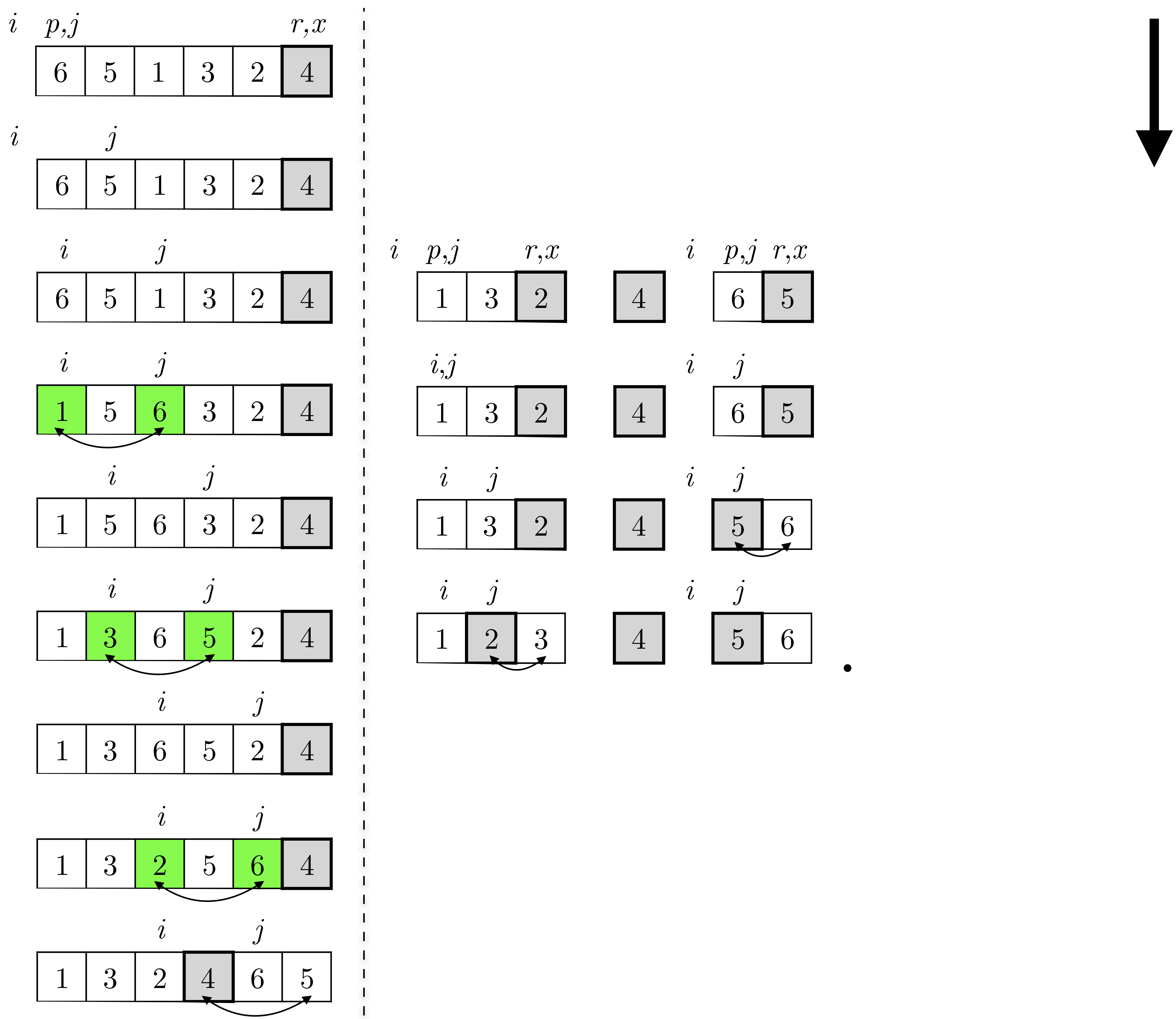


Step-by-step example for worst case

Step-by-step example for worst case



Recall previous example (average case)



Cost estimation (running time)

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- Cost is mainly affected by the partition operation, and especially by the for-loop in it that performs n−1 comparisons

- The cost for a single partition operation is: $\Theta(n)$, where $n = r - p + 1$

Worst case cost estimation example ($n = 8$)

$n = 8$ n-dimensional array, cost: $c * 8$

Worst case cost estimation example ($n = 8$)

$$\begin{tikzcd} & n = 8 \arrow[dl] \arrow[dr] & \\ n - 1 = 7 & P & 0 \end{tikzcd}$$

n-dimensional array, cost: $c * 8$

P: pivot element, cost: $c * (7+1)$

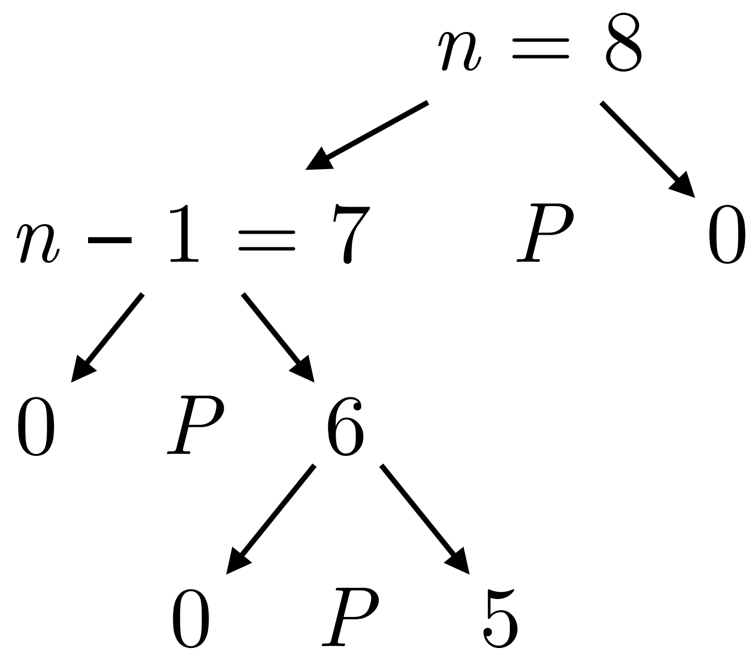
Worst case cost estimation example ($n = 8$)

$n = 8$ — n-dimensional array, cost: $c * 8$

$n - 1 = 7$ $\quad P \quad 0$ — P: pivot element, cost: $c * (7+1)$

$0 \quad P \quad 6$ — cost: $c * 7$

Worst case cost estimation example ($n = 8$)



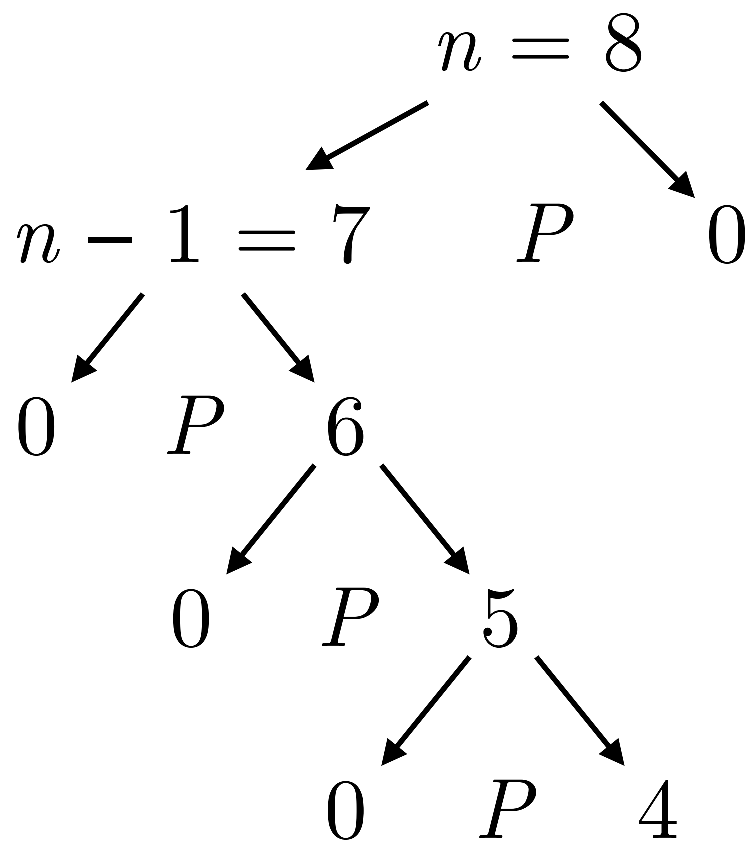
n-dimensional array, cost: $c * 8$

P: pivot element, cost: $c * (7{+}1)$

cost: $c * 7$

cost: $c * 6$

Worst case cost estimation example ($n = 8$)



n-dimensional array, cost: $c * 8$

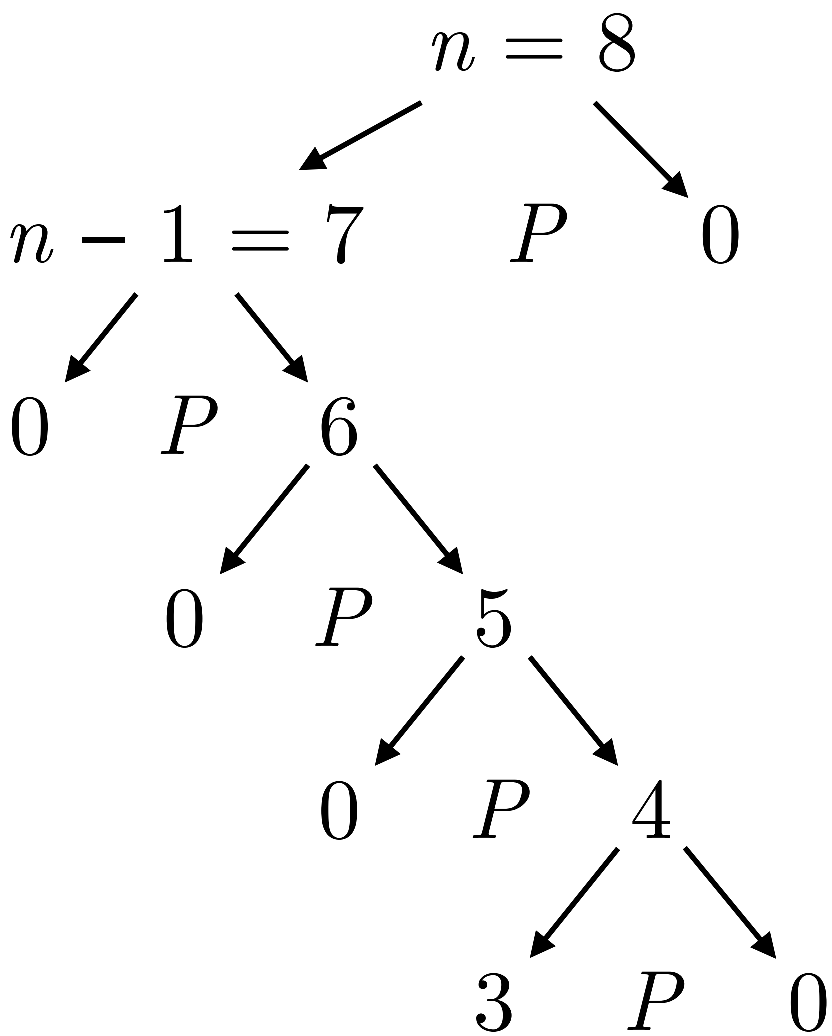
P: pivot element, cost: $c * (7+1)$

cost: $c * 7$

cost: $c * 6$

cost: $c * 5$

Worst case cost estimation example ($n = 8$)



n-dimensional array, cost: $c * 8$

P: pivot element, cost: $c * (7{+}1)$

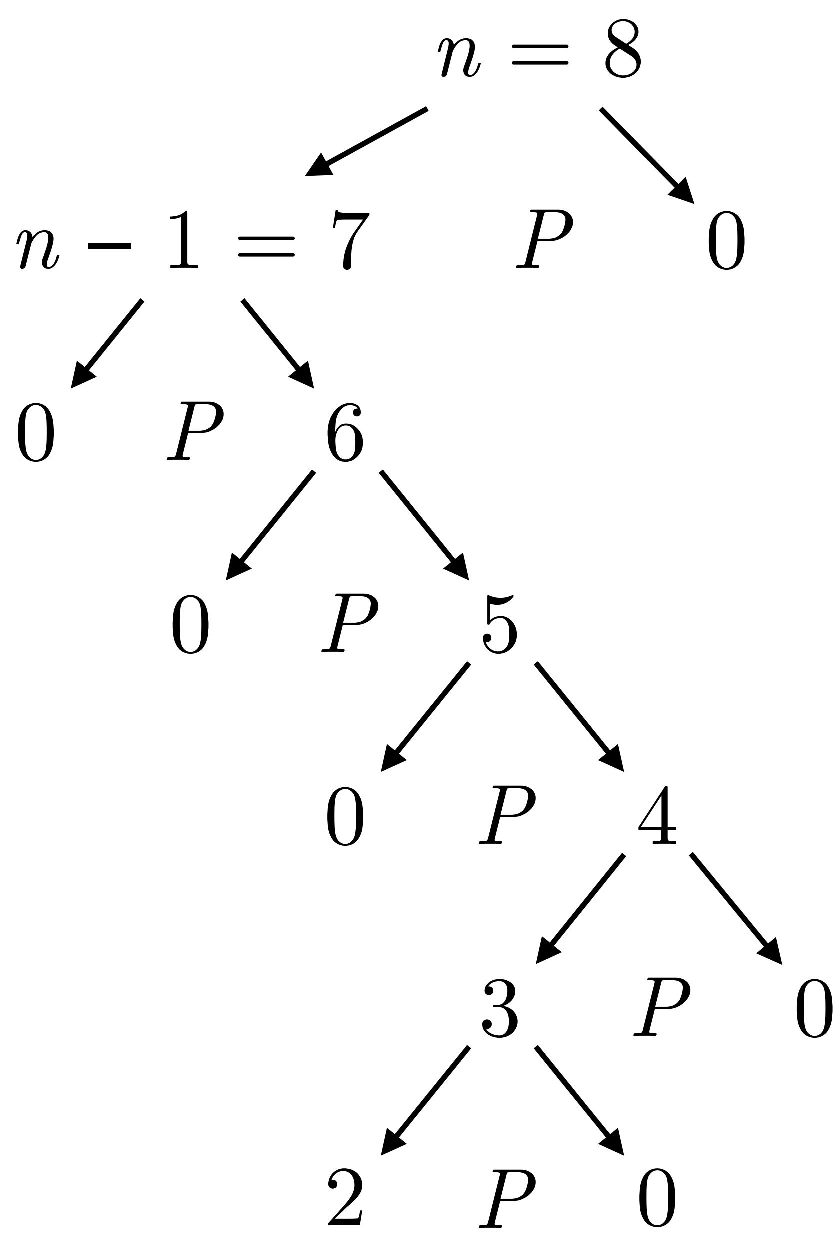
cost: $c * 7$

cost: $c * 6$

cost: $c * 5$

cost: $c * 4$

Worst case cost estimation example ($n = 8$)



n-dimensional array, cost: $c * 8$

P: pivot element, cost: $c * (7{+}1)$

cost: $c * 7$

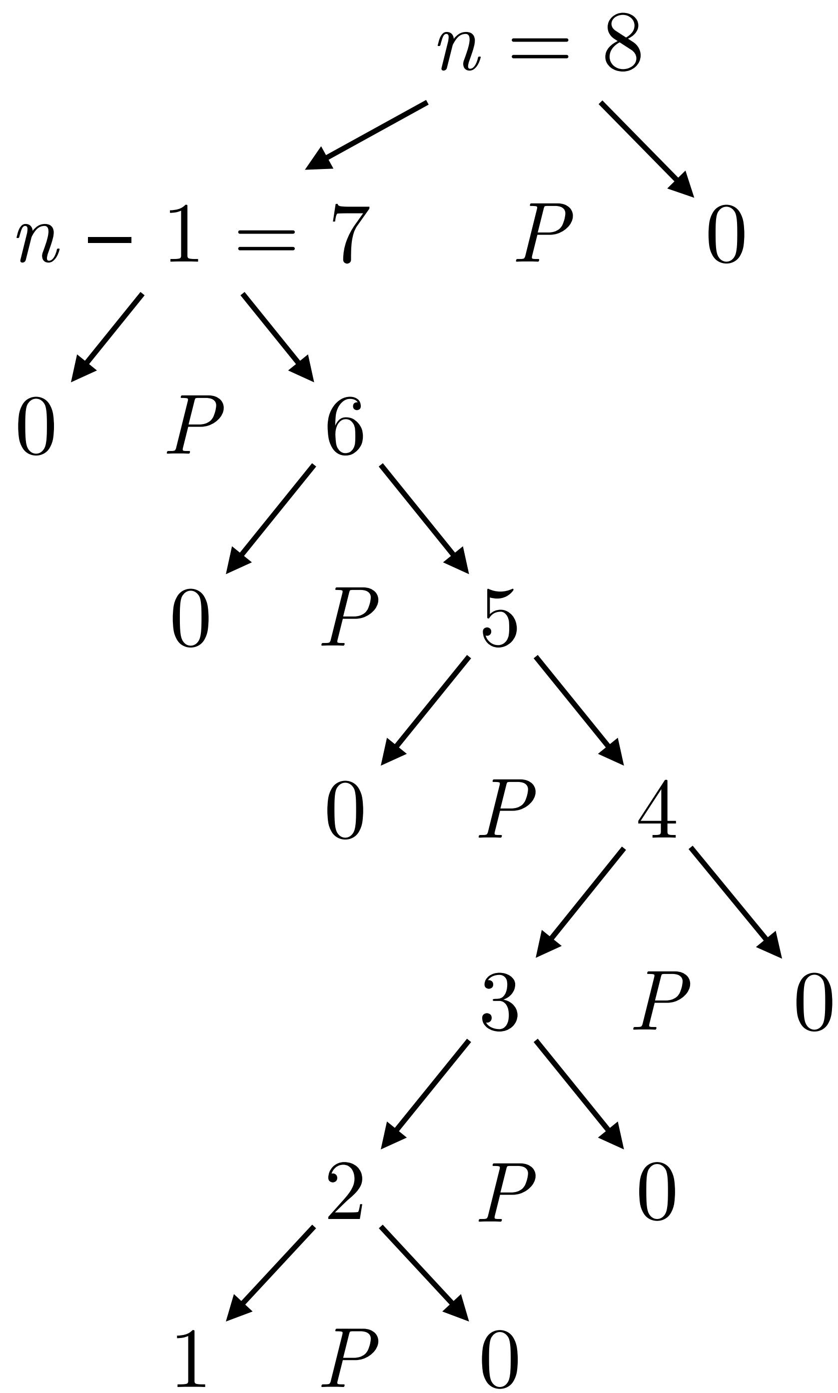
cost: $c * 6$

cost: $c * 5$

cost: $c * 4$

cost: $c * 3$

Worst case cost estimation example ($n = 8$)



n-dimensional array, cost: $c * 8$

P: pivot element, cost: $c * (7+1)$

cost: $c * 7$

cost: $c * 6$

cost: $c * 5$

cost: $c * 4$

cost: $c * 3$

cost: $c * 2$

total cost: $c * (8+8\ldots+2) = c * 43 \approx \Theta(n^2)$

Worst case cost analysis (1/3)

- Performing partition once on an n-dimensional array, generates 2 sub-arrays with q and $n{-}q{-}1$ elements

Worst case cost analysis (1/3)

- Performing partition once on an n-dimensional array, generates 2 sub-arrays with q and $n{-}q{-}1$ elements
- The cost of partitioning an array with n elements is $\Theta(n)$

Worst case cost analysis (1/3)

- Performing partition once on an n-dimensional array, generates 2 sub-arrays with q and $n - q - 1$ elements
- The cost of partitioning an array with n elements is $\Theta(n)$

So, the cost of quicksort is: $\mathrm{T}(n) = \big(\mathrm{T}(q) + \mathrm{T}(n - q - 1)\big) + \Theta(n)$

Worst case cost analysis (1/3)

- Performing partition once on an n-dimensional array, generates 2 sub-arrays with q and $n-q-1$ elements
- The cost of partitioning an array with n elements is $\Theta(n)$

So, the cost of quicksort is: $\mathrm{T}(n) = \big(\mathrm{T}(q) + \mathrm{T}(n-q-1)\big) + \Theta(n)$

In the worst case, this cost will be maximised, i.e.

$$\mathrm{T}(n) = \max_{0 \le q \le n-1} \big(\mathrm{T}(q) + \mathrm{T}(n-q-1)\big) + \Theta(n) \qquad (1)$$

Worst case cost analysis (1/3)

- Performing partition once on an n-dimensional array, generates 2 sub-arrays with q and $n\!-\!q\!-\!1$ elements
- The cost of partitioning an array with n elements is $\Theta(n)$

So, the cost of quicksort is: $\mathrm{T}(n) = \big(\mathrm{T}(q) + \mathrm{T}(n-q-1)\big) + \Theta(n)$

In the worst case, this cost will be maximised, i.e.

$$\mathrm{T}(n) = \max_{0 \le q \le n-1} \big(\mathrm{T}(q) + \mathrm{T}(n-q-1)\big) + \Theta(n) \qquad (1)$$

Let's now assume that: $\mathrm{T}(n) = \mathrm{O}(n^2) \le cn^2$, where $c > 0$ (2)

Worst case cost analysis (1/3)

- Performing partition once on an n-dimensional array, generates 2 sub-arrays with q and $n-q-1$ elements
- The cost of partitioning an array with n elements is $\Theta(n)$

So, the cost of quicksort is: $\mathrm{T}(n) = \big(\mathrm{T}(q) + \mathrm{T}(n-q-1)\big) + \Theta(n)$

In the worst case, this cost will be maximised, i.e.

$$\mathrm{T}(n) = \max_{0 \le q \le n-1} \big(\mathrm{T}(q) + \mathrm{T}(n-q-1)\big) + \Theta(n) \qquad (1)$$

Let's now assume that: $\mathrm{T}(n) = \mathrm{O}(n^2) \le cn^2$, where $c > 0$ (2)

and substitute (1) in (2): $\mathrm{T}(n) \le \max_{0 \le q \le n-1} \big(cq^2 + c(n-q-1)^2\big) + \Theta(n)$

Worst case cost analysis

$$\mathrm{T}(n) \leq \max_{0\leq q\leq n-1} \left(cq^2 + c(n-q-1)^2\right) + \Theta(n)$$

$$= c \max_{0\leq q\leq n-1} \left(q^2 + (n-q-1)^2\right) + \Theta(n)$$

Worst case cost analysis (2/3)

$$\mathrm{T}(n) \leq \max_{0 \leq q \leq n-1} \left(cq^2 + c(n-q-1)^2 \right) + \Theta(n)$$

$$= c \max_{0 \leq q \leq n-1} \underbrace{\boxed{\left(q^2 + (n-q-1)^2 \right)}}_{g(q)} + \Theta(n)$$

So, we want to maximise $\quad g(q) = q^2 + (n-q-1)^2$

Worst case cost analysis (2/3)

$$\mathrm{T}(n) \leq \max_{0 \leq q \leq n-1} \left(cq^2 + c(n-q-1)^2\right) + \Theta(n)$$

$$= c \max_{0 \leq q \leq n-1} \underbrace{\left(q^2 + (n-q-1)^2\right)}_{g(q)} + \Theta(n)$$

So, we want to maximise $g(q) = q^2 + (n-q-1)^2$

$$\frac{\partial g}{\partial q} = 2q + 2(n-q-1)(-1) = 4q - 2n + 2$$

$$\frac{\partial g}{\partial q} = 0 \implies q = \frac{1}{2}(n-1)$$

Worst case cost analysis (2/3)

$$T(n) \le \max_{0 \le q \le n-1} \left(cq^2 + c(n-q-1)^2\right) + \Theta(n)$$

$$= c \max_{0 \le q \le n-1} \underbrace{\boxed{\left(q^2 + (n-q-1)^2\right)}}_{g(q)} + \Theta(n)$$

So, we want to maximise $\quad g(q) = q^2 + (n-q-1)^2$

$$\frac{\partial g}{\partial q} = 2q + 2(n-q-1)(-1) = 4q - 2n + 2$$

$$\frac{\partial g}{\partial q} = 0 \implies q = \frac{1}{2}(n-1)$$

$$\frac{\partial^2 g}{\partial q^2} = 4 > 0$$

Worst case cost analysis (2/3)

$$T(n) \leq \max_{0 \leq q \leq n-1} \left(cq^2 + c(n-q-1)^2 \right) + \Theta(n)$$

$$= c \max_{0 \leq q \leq n-1} \underbrace{\boxed{\left(q^2 + (n-q-1)^2 \right)}}_{g(q)} + \Theta(n)$$

So, we want to maximise $\quad g(q) = q^2 + (n-q-1)^2$

$$\frac{\partial g}{\partial q} = 2q + 2(n-q-1)(-1) = 4q - 2n + 2$$

$$\frac{\partial g}{\partial q} = 0 \implies \boxed{q = \frac{1}{2}(n-1)}$$ this is a local minimum of $g(q)$

$$\frac{\partial^2 g}{\partial q^2} = 4 > 0$$

Worst case cost analysis (3/3)

$$\mathrm{T}(n) \leq c \max_{0 \leq q \leq n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \dfrac{1}{2}(n-1)$

Worst case cost analysis (3/3)

$$\mathrm{T}(n) \leq c \max_{0 \leq q \leq n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \frac{1}{2}(n-1)$

max of $g(q)$ for $q = 0$ or $q = n-1$

Worst case cost analysis (3/3)

$$T(n) \leq c \max_{0 \leq q \leq n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \frac{1}{2}(n-1)$

max of $g(q)$ for $q = 0$ or $q = n-1$

$$g(0) = g(n-1) = (n-1)^2,\ g\left(\frac{n-1}{2}\right) = \frac{(n-1)^2}{2}$$

Worst case cost analysis (3/3)

$$T(n) \le c \max_{0 \le q \le n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \frac{1}{2}(n-1)$

max of $g(q)$ for $q = 0$ or $q = n - 1$

$$g(0) = g(n-1) = (n-1)^2,\ g\left(\frac{n-1}{2}\right) = \frac{(n-1)^2}{2}$$

$$T(n) \le c(n-1)^2 + \Theta(n) = cn^2 - 2cn + c + \Theta(n) \le cn^2$$

Worst case cost analysis (3/3)

$$T(n) \le c \max_{0 \le q \le n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \frac{1}{2}(n-1)$

max of $g(q)$ for $q = 0$ or $q = n - 1$

$$g(0) = g(n-1) = (n-1)^2, \; g\left(\frac{n-1}{2}\right) = \frac{(n-1)^2}{2}$$

$$T(n) \le c(n-1)^2 + \Theta(n) = cn^2 - 2cn + c + \Theta(n) \le cn^2$$

which results in $\boxed{T(n) = O(n^2)}$

Worst case cost analysis (3/3)

$$T(n) \le c \max_{0\le q\le n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \frac{1}{2}(n-1)$

max of $g(q)$ for $q = 0$ or $q = n-1$

$$g(0) = g(n-1) = (n-1)^2,\ g\left(\frac{n-1}{2}\right) = \frac{(n-1)^2}{2}$$

$$T(n) \le c(n-1)^2 + \Theta(n) = cn^2 - 2cn + c + \Theta(n) \le cn^2$$

which results in $\boxed{T(n) = O(n^2)}$

Also: $T(n) \ge c\frac{1}{2}(n-1)^2 + \Theta(n) = \frac{cn^2}{2} + \frac{c}{2}$

Worst case cost analysis (3/3)

$$T(n) \leq c \max_{0\leq q\leq n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \frac{1}{2}(n-1)$

max of $g(q)$ for $q = 0$ or $q = n-1$

$$g(0) = g(n-1) = (n-1)^2, \; g\left(\frac{n-1}{2}\right) = \frac{(n-1)^2}{2}$$

$$T(n) \leq c(n-1)^2 + \Theta(n) = cn^2 - 2cn + c + \Theta(n) \leq cn^2$$

which results in $\boxed{T(n) = O(n^2)}$

Also: $T(n) \geq c\frac{1}{2}(n-1)^2 + \Theta(n) = \frac{cn^2}{2} + \frac{c}{2}$

which results in $\boxed{T(n) = \Omega(n^2)}$

Worst case cost analysis (3/3)

$$T(n) \le c \max_{0 \le q \le n-1} g(q) + \Theta(n)$$

min of $g(q)$ for $q = \frac{1}{2}(n-1)$

max of $g(q)$ for $q = 0$ or $q = n-1$

$$g(0) = g(n-1) = (n-1)^2,\ g\left(\frac{n-1}{2}\right) = \frac{(n-1)^2}{2}$$

$$T(n) \le c(n-1)^2 + \Theta(n) = cn^2 - 2cn + c + \Theta(n) \le cn^2$$

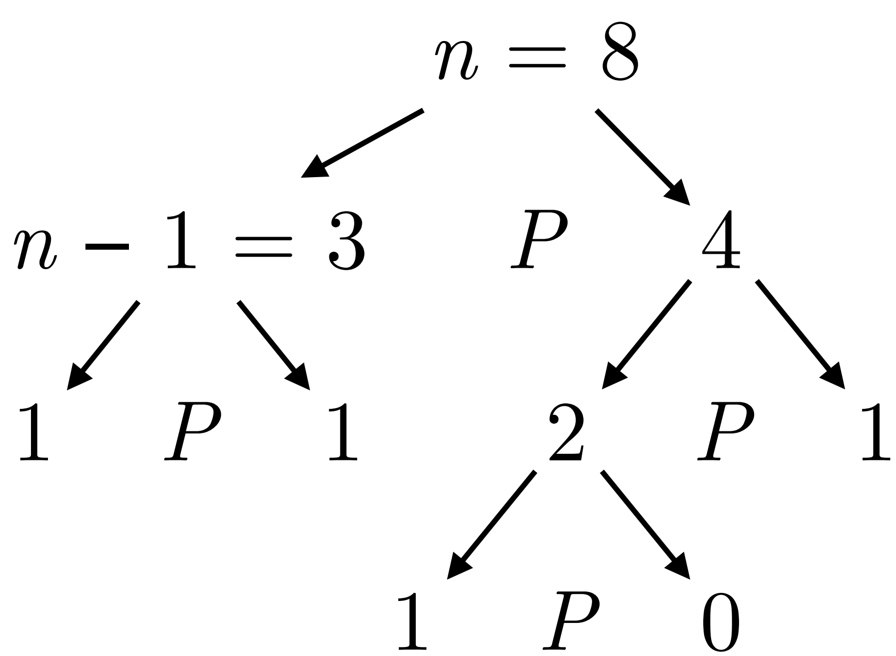
which results in $\boxed{T(n) = O(n^2)}$

Also: $T(n) \ge c\frac{1}{2}(n-1)^2 + \Theta(n) = \frac{cn^2}{2} + \frac{c}{2}$

which results in $\boxed{T(n) = \Omega(n^2)}$

Thus, $T(n) = \Theta(n^2)$.

Best case cost estimation example ($n = 8$)



n-dimensional array, cost: $c * 8$

P: pivot element, cost: $c * 7$

cost: $c * 5$

cost: $c * 2$

total cost: $c * (8+7+5+2) = c * 22 \approx \Theta(n \log n)$

Best case cost analysis

If the splits are even, partition produces two sub-problems, each of which has no size more than $n/2$.

Best case cost analysis

If the splits are even, partition produces two sub-problems, each of which has no size more than $n/2$.

Thus the running time is equal to:

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

Best case cost analysis

If the splits are even, partition produces two sub-problems, each of which has no size more than $n/2$.

Thus the running time is equal to:

$$\mathrm{T}(n) = 2\mathrm{T}(n/2) + \Theta(n)$$

Using case 2 of the master theorem (see Theorem 4.1 in Cormen et al. textbook, 3rd edition), this has the solution:

$$\mathrm{T}(n) = \Theta(n \log n)$$

For $\mathrm{T}(n) = a\mathrm{T}(n/b) + f(n)$,

if $f(n) = \Theta\left(n^{\log_b a}\right)$, then $\mathrm{T}(n) = \Theta\left(n^{\log_b a} \log n\right)$,

where $b = 2$ and $a = 2$.

Randomised quicksort

- Instead of using the right-most element, $A[r]$, as the pivot...

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Randomised quicksort

- Instead of using the right-most element, $A[r]$, as the pivot...

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

- **Why?** By adding randomisation, obtaining the average expected performance is more likely than obtaining the worst case performance.

Average number of comparisons

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Average number of comparisons (1/3)

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

$$c_n = 1 + (n - 1) + \ldots$$

Average number of comparisons


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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

Average number of comparisons (1/3)

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

$$\boxed{c_n} = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

total number of comparisons

Average number of comparisons (1/3)

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
        if A[j] ≤ x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

$$\boxed{c_n} = n + \boxed{\frac{1}{n}}\Big[\big(c_0 + c_{n-1}\big) + \big(c_1 + c_{n-2}\big) + \ldots + \big(c_{n-1} + c_0\big)\Big]$$

total number of comparisons

probability of a split — n pivot choices

Average number of comparisons (1/3)

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

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

$$c_n = n + \frac{1}{n}\Big[\big(c_0 + c_{n-1}\big) + \big(c_1 + c_{n-2}\big) + \ldots + \big(c_{n-1} + c_0\big)\Big]$$

total number of comparisons

probability of a split — n pivot choices

sub-array sizes

Average number of comparisons (2/3)

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

Average number of comparisons (2/3)

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

Average number of comparisons (2/3)

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

$$(n-1)c_{n-1} = (n-1)^2 + 2\left(c_0 + c_1 + \ldots + c_{n-2}\right)$$ Replace $n \to n{-}1$

Average number of comparisons (2/3)

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

subtract

$$(n-1)c_{n-1} = (n-1)^2 + 2\left(c_0 + c_1 + \ldots + c_{n-2}\right)$$

Average number of comparisons

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

subtract

$$(n-1)c_{n-1} = (n-1)^2 + 2\left(c_0 + c_1 + \ldots + c_{n-2}\right)$$

$$nc_n - (n-1)c_{n-1} = 2(n-1) + 2c_{n-1} \implies$$

Average number of comparisons (2/3)

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

subtract

$$(n-1)c_{n-1} = (n-1)^2 + 2\left(c_0 + c_1 + \ldots + c_{n-2}\right)$$

$$nc_n - (n-1)c_{n-1} = 2(n-1) + 2c_{n-1} \implies$$

$$c_n = \frac{2n-1}{n} + \frac{(n+1)c_{n-1}}{n} \implies$$

Average number of comparisons (2/3)

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

subtract

$$(n-1)c_{n-1} = (n-1)^2 + 2\left(c_0 + c_1 + \ldots + c_{n-2}\right)$$

$$nc_n - (n-1)c_{n-1} = 2(n-1) + 2c_{n-1} \implies$$

$$c_n = \frac{2n-1}{n} + \frac{(n+1)c_{n-1}}{n} \implies$$ divide by $n+1$

Average number of comparisons

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

subtract

$$(n-1)c_{n-1} = (n-1)^2 + 2\left(c_0 + c_1 + \ldots + c_{n-2}\right)$$

$$nc_n - (n-1)c_{n-1} = 2(n-1) + 2c_{n-1} \implies$$

$$c_n = \frac{2n-1}{n} + \frac{(n+1)c_{n-1}}{n} \implies$$ divide by $n+1$

$$\frac{c_n}{n+1} = \frac{2}{n+1} - \frac{1}{n(n+1)} + \frac{c_{n-1}}{n}$$

Average number of comparisons (2/3)

$$c_n = n + \frac{1}{n}\left[\left(c_0 + c_{n-1}\right) + \left(c_1 + c_{n-2}\right) + \ldots + \left(c_{n-1} + c_0\right)\right]$$

$$= n + \frac{2}{n}\left(c_0 + c_1 + \ldots + c_{n-1}\right) \implies$$

$$nc_n = n^2 + 2\left(c_0 + c_1 + \ldots + c_{n-1}\right)$$

subtract

$$(n-1)c_{n-1} = (n-1)^2 + 2\left(c_0 + c_1 + \ldots + c_{n-2}\right)$$

$$nc_n - (n-1)c_{n-1} = 2(n-1) + 2c_{n-1} \implies$$

$$c_n = \frac{2n-1}{n} + \frac{(n+1)c_{n-1}}{n} \implies \text{ divide by } n+1$$

$$\frac{c_n}{n+1} = \frac{2}{n+1} - \frac{1}{n(n+1)} + \frac{c_{n-1}}{n} \le \frac{2}{n+1} + \frac{c_{n-1}}{n}$$

Average number of comparisons (3/3)

$$\frac{c_n}{n+1} \leq \frac{c_{n-1}}{n} + \frac{2}{n+1} \qquad \text{replace } n \to n\text{–}1$$

Average number of comparisons (3/3)

$$\frac{c_n}{n+1} \leq \frac{c_{n-1}}{n} + \frac{2}{n+1} \qquad \text{replace } n \rightarrow n\text{–}1$$

$$= \frac{c_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \frac{c_{n-3}}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

Average number of comparisons

$$\frac{c_n}{n+1} \leq \frac{c_{n-1}}{n} + \frac{2}{n+1} \qquad \text{replace } n \to n\text{–}1$$

$$= \frac{c_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \frac{c_{n-3}}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \ldots = \frac{c_1}{2} + 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right)$$

Average number of comparisons (3/3)

$$\frac{c_n}{n+1} \leq \frac{c_{n-1}}{n} + \frac{2}{n+1} \qquad \text{replace } n \to n\text{–}1$$

$$= \frac{c_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \frac{c_{n-3}}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \ldots = \frac{c_1}{2} + 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right)$$

$$= \frac{c_1}{2} + 2\sum_{i=2}^{n} \frac{1}{i+1}$$

Average number of comparisons

$$\frac{c_n}{n+1} \leq \frac{c_{n-1}}{n} + \frac{2}{n+1} \qquad \text{replace } n \to n\text{–}1$$

$$= \frac{c_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \frac{c_{n-3}}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \ldots = \frac{c_1}{2} + 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right)$$

$$= \frac{c_1}{2} + 2\sum_{i=2}^{n} \frac{1}{i+1} \;\leq 2\sum_{i=1}^{n} \frac{1}{i}$$

Average number of comparisons (3/3)

$$\frac{c_n}{n+1} \le \frac{c_{n-1}}{n} + \frac{2}{n+1} \qquad \text{replace } n \to n\text{–}1$$

$$= \frac{c_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \frac{c_{n-3}}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \ldots = \frac{c_1}{2} + 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right) \qquad \text{Harmonic series}$$

or $\ln(n)$

$$= \frac{c_1}{2} + 2\sum_{i=2}^{n} \frac{1}{i+1} \le 2\sum_{i=1}^{n} \frac{1}{i} = 2H_n \approx 2\log_e(n) \implies$$

Average number of comparisons (3/3)

$$\frac{c_n}{n+1} \le \frac{c_{n-1}}{n} + \frac{2}{n+1}$$ replace $n \to n{-}1$

$$= \frac{c_{n-2}}{n-1} + \frac{2}{n} + \frac{2}{n+1} = \frac{c_{n-3}}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \ldots = \frac{c_1}{2} + 2\left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right)$$ Harmonic series

or $\ln(n)$

$$= \frac{c_1}{2} + 2\sum_{i=2}^{n} \frac{1}{i+1} \le 2\sum_{i=1}^{n} \frac{1}{i} = 2H_n \approx 2\log_e(n) \Longrightarrow$$

change of base

$$c_n \le 2(n+1)\log_e(n) = 2(n+1)\frac{\log_2 n}{\log_2 e} \approx 1.39(n\log_2 n + \log_2 n) = O(n\log n)\,.$$

Average number of comparisons, take II (1/4)

- At most n calls to partition over the execution of quicksort because every time partition is called, it will handle at least one element
- Every call to the partition takes O(1) + lines 3-6 (focus on the number of comparisons)

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p - 1
3  for j = p to r - 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Average number of comparisons, take II (1/4)

- At most n calls to partition over the execution of quicksort because every time partition is called, it will handle at least one element
- Every call to the partition takes O(1) + lines 3-6 (focus on the number of comparisons)

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

If the entire quicksort requires m comparisons, then its running time is $O(n+m)$. **Let's estimate m!**

Average number of comparisons, take II (2/4)

If the entire quicksort requires m comparisons, then its running time is $O(n+m)$. **Let's estimate m!**

Notation

- A is a set containing elements $\{z_1, z_2, \ldots, z_n\}$, where z_i is the i-th smallest element — A's are not presumed to be sorted

Average number of comparisons, take II (2/4)

If the entire quicksort requires m comparisons, then its running time is $\mathrm{O}(n+m)$. **Let's estimate m!**

Notation

- A is a set containing elements $\{z_1, z_2, \ldots, z_n\}$, where z_i is the i-th smallest element — A's are not presumed to be sorted
- $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ is a set that contains elements from z_i to z_j

Average number of comparisons, take II (2/4)

If the entire quicksort requires m comparisons, then its running time is $\mathrm{O}(n+m)$. **Let's estimate m!**

Notation

- A is a set containing elements $\{z_1, z_2, \ldots, z_n\}$, where z_i is the i-th smallest element — A's are not presumed to be sorted
- $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ is a set that contains elements from z_i to z_j

How many times does quicksort compare elements z_i and z_j?

Average number of comparisons, take II (2/4)

If the entire quicksort requires m comparisons, then its running time is $O(n+m)$. **Let's estimate m!**

Notation

- A is a set containing elements $\{z_1, z_2, \ldots, z_n\}$, where z_i is the i-th smallest element — A's are not presumed to be sorted
- $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ is a set that contains elements from z_i to z_j

How many times does quicksort compare elements z_i and z_j?

- Any pair of elements of A is compared **at most once**… because elements are only compared to the pivot element.
- **Reminder:** Once used, the pivot element is not used again.

Average number of comparisons, take II (2/4)

If the entire quicksort requires m comparisons, then its running time is $\mathrm{O}(n+m)$. **Let's estimate m!**

Notation

- A is a set containing elements $\{z_1,z_2,\ldots,z_n\}$, where z_i is the i-th smallest element — A's are not presumed to be sorted
- $Z_{ij} = \{z_i,z_{i+1},\ldots,z_j\}$ is a set that contains elements from z_i to z_j

How many times does quicksort compare elements z_i and z_j?

- Any pair of elements of A is compared **at most once**… because elements are only compared to the pivot element.
- **Reminder:** Once used, the pivot element is not used again.

Hence, $m_{ij} = 1$, if z_i is compared to z_j, and 0 otherwise.

Average number of comparisons, take II (3/4)

Hence, $m_{ij} = 1$, if z_i is compared to z_j, and 0 otherwise.

Total number of comparisons: $m = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}$

Average number of comparisons, take II (3/4)

Hence, $m_{ij} = 1$, if z_i is compared to z_j, and 0 otherwise.

Total number of comparisons: $m = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}$

$$\mathrm{E}(m) = \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}\left(m_{ij}\right)$$

what is m's expectation?

Average number of comparisons, take II (3/4)

Hence, $m_{ij} = 1$, if z_i is compared to z_j, and 0 otherwise.

Total number of comparisons: $m = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}$

$$\mathrm{E}(m) = \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}\left(m_{ij}\right)$$ what is m's expectation?

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

Average number of comparisons, take II (3/4)

Hence, $m_{ij} = 1$, if z_i is compared to z_j, and 0 otherwise.

Total number of comparisons: $m = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}$

$$E(m) = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E\left(m_{ij}\right)$$ what is m's expectation?

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

- Note that when the pivot element x is $z_i < x < z_j$, z_i and z_j are not going to be compared. **Why?**

Average number of comparisons, take II (3/4)

Hence, $m_{ij} = 1$, if z_i is compared to z_j, and 0 otherwise.

Total number of comparisons: $m = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}$

$$E(m) = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} m_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E\left(m_{ij}\right)$$ what is m's expectation?

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

- Note that when the pivot element x is $z_i < x < z_j$, z_i and z_j are not going to be compared. **Why?**
- To compare them we need to select z_i or z_j as pivots.

Average number of comparisons, take II (4/4)

- Note that when the pivot element x is $z_i < x < z_j$, z_i and z_j are not going to be compared. **Why?**
- To compare them we need to select z_i or z_j as pivots.

Average number of comparisons, take II (4/4)

- Note that when the pivot element x is $z_i < x < z_j$, z_i and z_j are not going to be compared. **Why?**
- To compare them we need to select z_i or z_j as pivots.
- Z_{ij} has j–i+1 elements. Selecting one of them as the first pivot has a probability of $1/(j$–i+1).

Average number of comparisons, take II (4/4)

- Note that when the pivot element x is $z_i < x < z_j$, z_i and z_j are not going to be compared. **Why?**
- To compare them we need to select z_i or z_j as pivots.
- Z_{ij} has j–i+1 elements. Selecting one of them as the first pivot has a probability of $1/(j–i+1)$.

$$\Pr\{z_i \text{ is compared to } z_j\} = \Pr\{z_i \text{ or } z_j \text{ are the first pivots}\} = \frac{2}{j-i+1}$$

Average number of comparisons, take II (4/4)

- Note that when the pivot element x is $z_i < x < z_j$, z_i and z_j are not going to be compared. **Why?**
- To compare them we need to select z_i or z_j as pivots.
- Z_{ij} has j–i+1 elements. Selecting one of them as the first pivot has a probability of $1/(j$–i+$1)$.

$$\Pr\{z_i \text{ is compared to } z_j\} = \Pr\{z_i \text{ or } z_j \text{ are the first pivots}\} = \frac{2}{j-i+1}$$

$$\mathrm{E}(m) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Average number of comparisons, take II (4/4)

- Note that when the pivot element x is $z_i < x < z_j$, z_i and z_j are not going to be compared. **Why?**
- To compare them we need to select z_i or z_j as pivots.
- Z_{ij} has $j–i+1$ elements. Selecting one of them as the first pivot has a probability of $1/(j–i+1)$.

$$\Pr\{z_i \text{ is compared to } z_j\} = \Pr\{z_i \text{ or } z_j \text{ are the first pivots}\} = \frac{2}{j-i+1}$$

$$\mathrm{E}(m) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$k = j - i, \text{then } \mathrm{E}(m) = \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$\approx \sum_{i=1}^{n-1} \mathrm{O}\left(\log n\right) = \mathrm{O}\left(n \log n\right).$$

Similar to the previous proof

Slides (with potential **revisions**)

lampos.net/slides/quicksort2019.pdf

end_of_lecture

@lampos

lampos.net