Last Week

- Pseudo-code
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - A first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Quicksort Problem

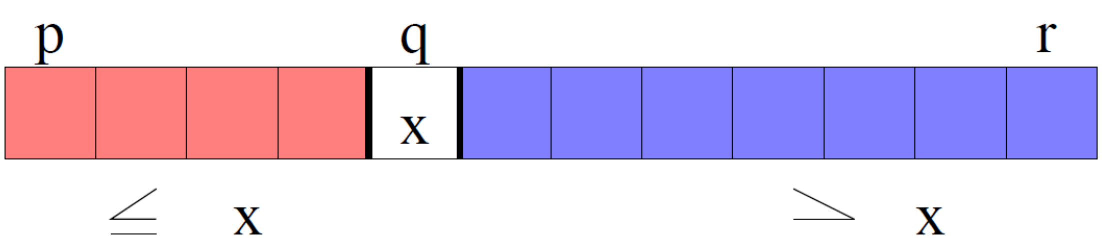
Outline

- Quicksort Problem
  - Basic partition
  - Randomized partition and randomized quicksort
  - Analysis of the randomized quicksort

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

$$A[u] < A[q] < A[v] \;\; for\ any\ p \le u \le q-1\ and\ q+1 \le v \le r$$

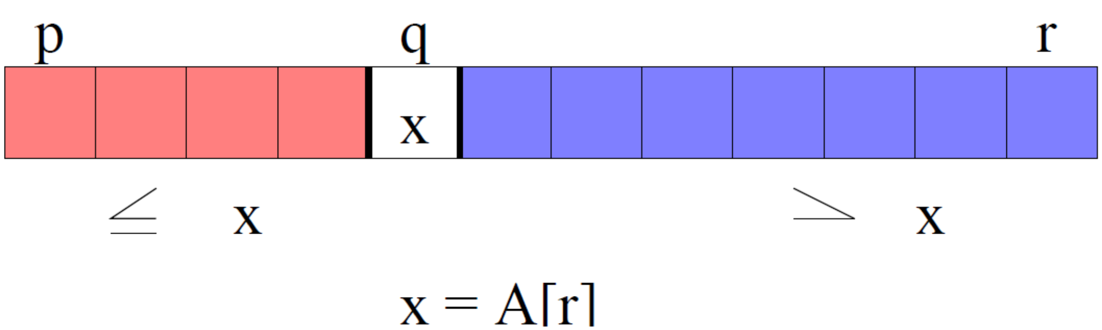


x = A[r]

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

$$A[u] < A[q] < A[v] \;\; for\, any\, p \le u \le q-1 \; and \; q+1 \le v \le r$$

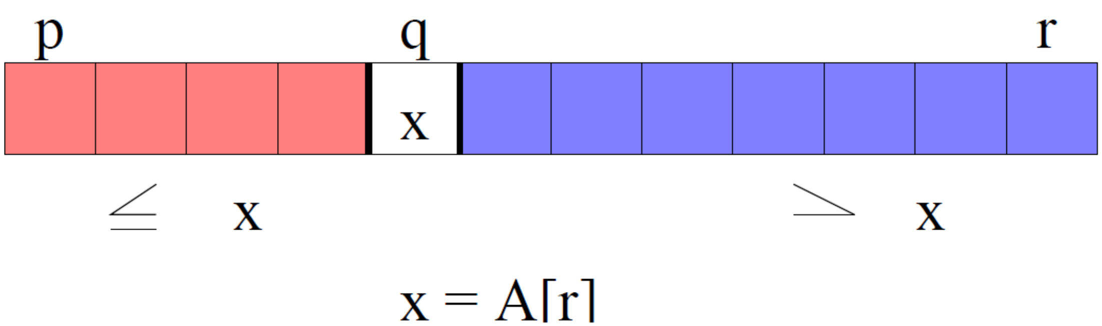


  - x is called the pivot. Assume x = A[r]; if not, swap first

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

$$A[u] < A[q] < A[v] \ \ for\ any\ p \leq u \leq q-1\ and\ q+1 \leq v \leq r$$

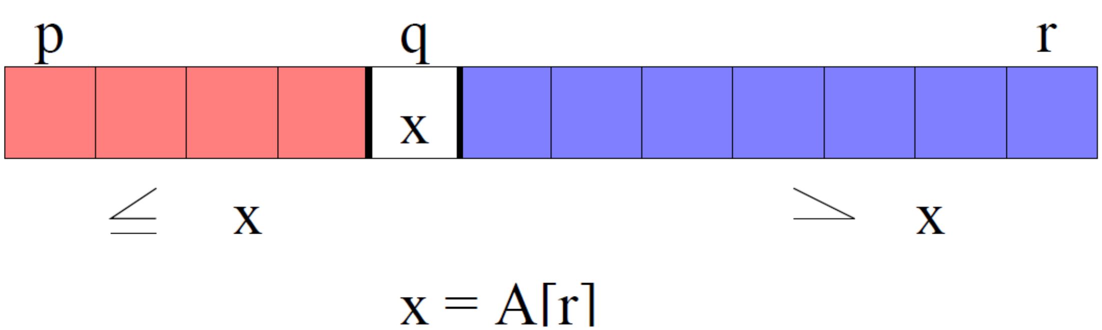


  - x is called the pivot. Assume x = A[r]; if not, swap first
  - Quicksort works by:

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

$$A[u] < A[q] < A[v] \;\; for\, any\, p \le u \le q-1\, and\, q+1 \le v \le r$$

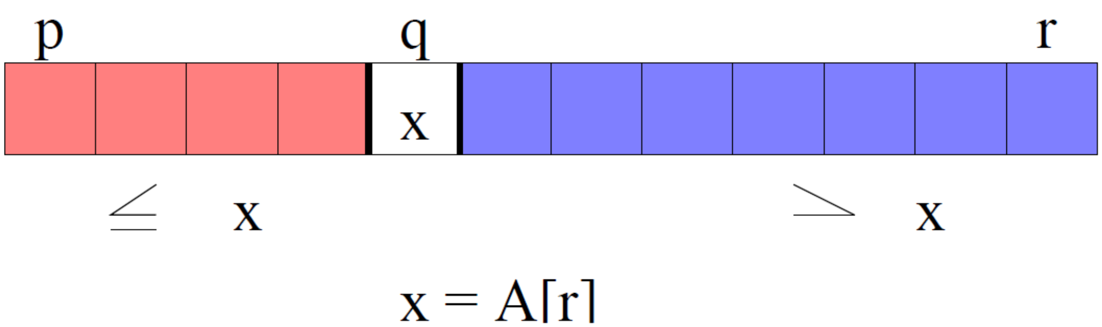


  - x is called the pivot. Assume x = A[r]; if not, swap first
  - Quicksort works by:
    - calling partition first

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

$$A[u] < A[q] < A[v] \;\; for\, any\, p \leq u \leq q-1 \; and \; q+1 \leq v \leq r$$

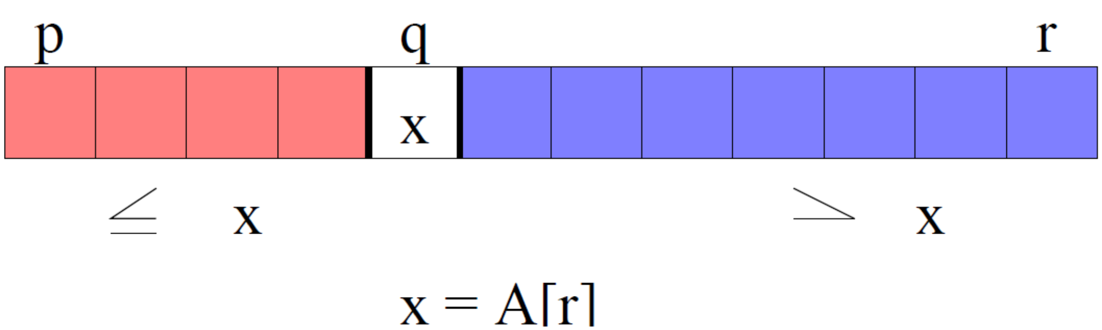


  - x is called the pivot. Assume x = A[r]; if not, swap first
  - Quicksort works by:
    - calling partition first
    - recursively sorting A[ ] and A[ ]

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

$$A[u] < \ A[q] < A[v] \ \ for\ any\ p \leq u \leq q-1\ and\ q+1 \leq v \leq r$$

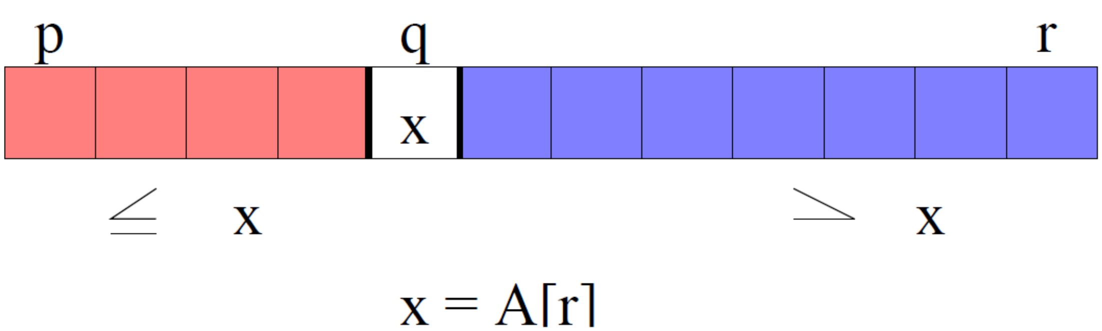


  - x is called the pivot. Assume x = A[r]; if not, swap first
  - Quicksort works by:
    - calling partition first
    - recursively sorting A[p..q-1] and A[ ]

Partition

- Partition
  - Given: An array of numbers
  - Partition: Rearrange the array A[p..r] in place into two (possibly empty) subarrays A[p..q-1] and A[q+1..r] such that

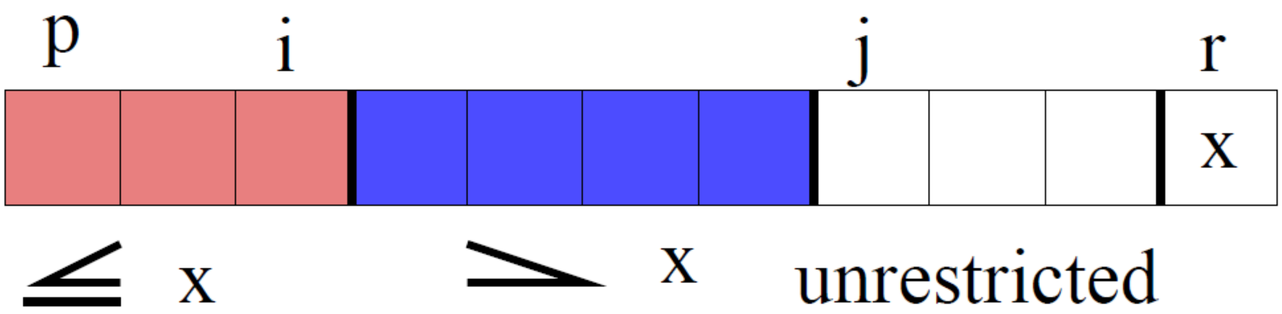
$$A[u] < A[q] < A[v] \;\; for\ any\ p \leq u \leq q-1\ and\ q+1 \leq v \leq r$$



  - x is called the pivot. Assume x = A[r]
  - Quicksort works by:
    - calling partition first
    - recursively sorting A[p..q-1] and A[q+1..r]

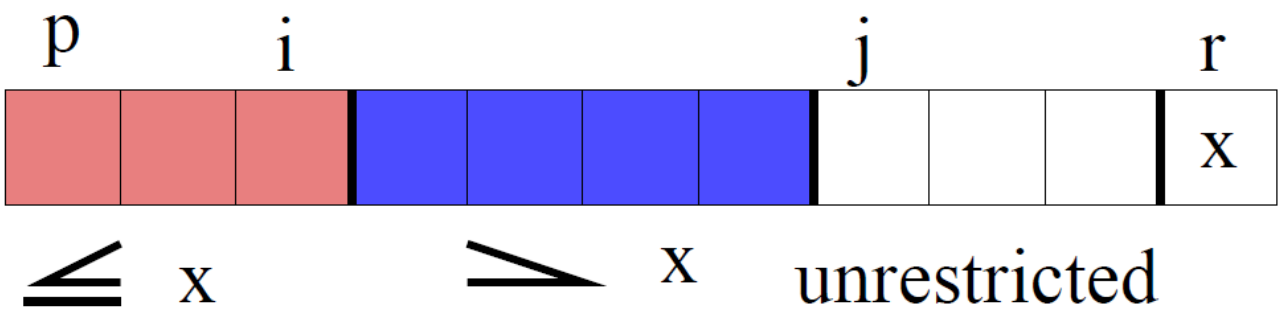
Partition

- The idea of Partition(A, p, r)
  - Use A[r] as the pivot, and grow partition from left to right



Partition

- The idea of Partition(A, p, r)
  - Use A[r] as the pivot, and grow partition from left to right



  - Initially (i, j) = (p–1, p)
  - Increase j by 1 each time to find a place for A[j]
    At the same time increase i when necessary
  - Stops when j = r

Partition

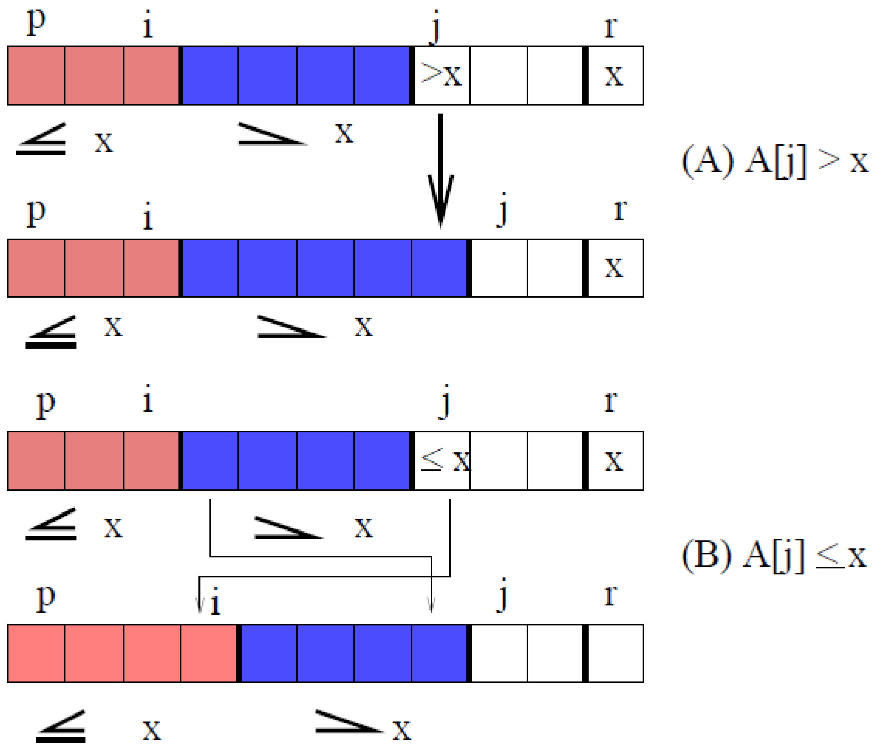
- One Iteration of the Procedure Partition
  - Increase j by 1 each time to find a place for A[j]

    At the same time increase i when necessary

Partition

- One Iteration of the Procedure Partition
  - Increase j by 1 each time to find a place for A[j]

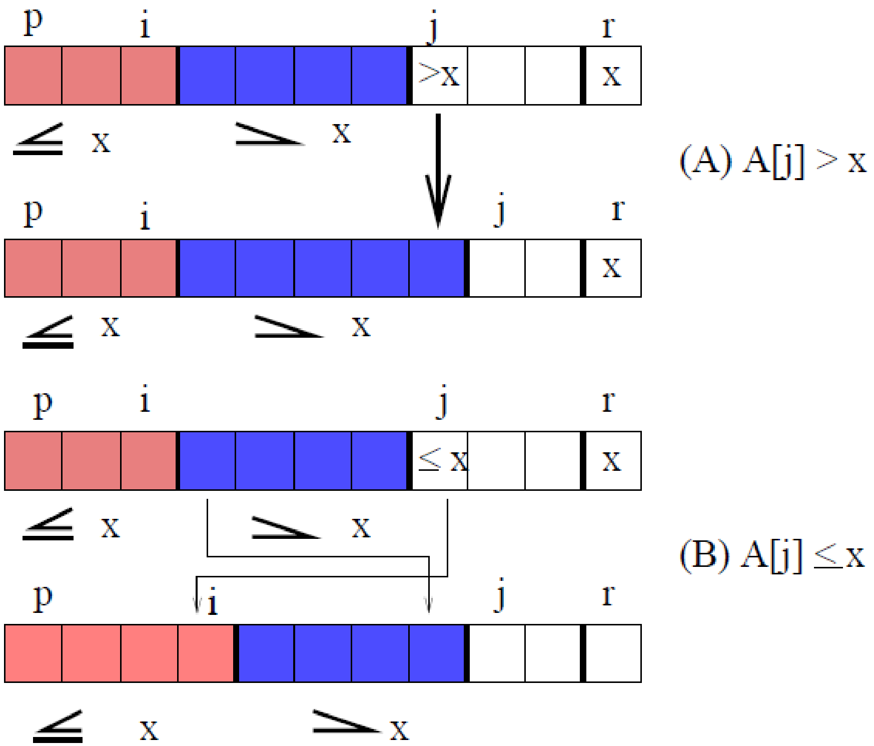
    At the same time increase i when necessary



Partition

- One Iteration of the Procedure Partition
  - Increase j by 1 each time to find a place for A[j]

    At the same time increase i when necessary

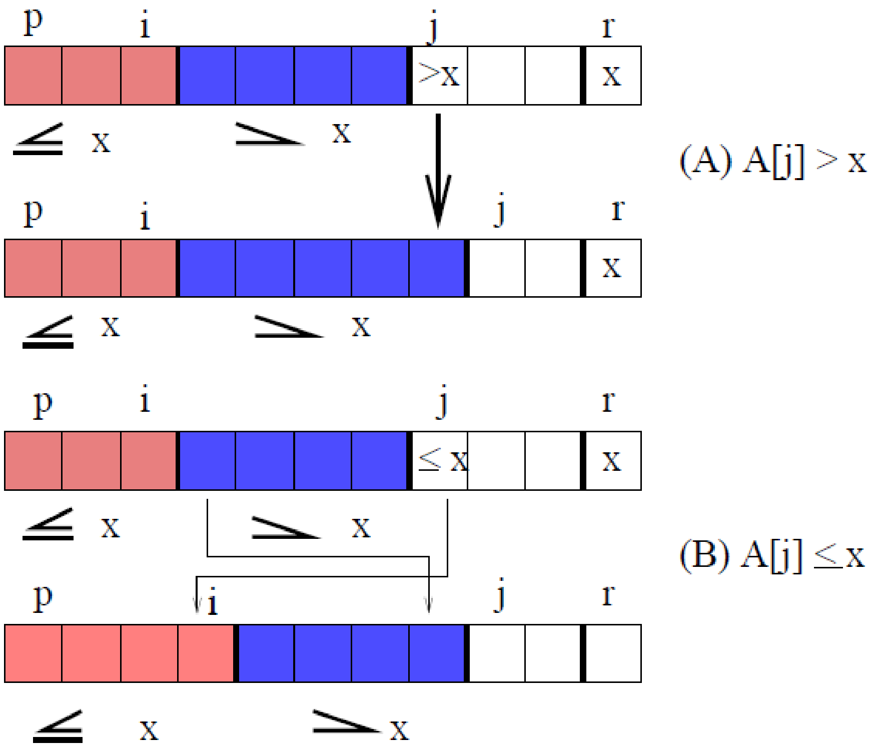


  - Case (A): Only increase j by 1

Partition

- One Iteration of the Procedure Partition
  - Increase j by 1 each time to find a place for A[j]

    At the same time increase i when necessary



  - Case (A): Only increase j by 1
  - Case (B): i = i + 1; A[i] ↔ A[j]; j = j + 1.

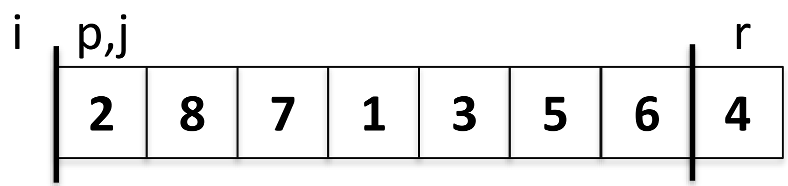
Partition-Example

- The Operation of Partition(A, p, r)

i	p,j						r
	2	8	7	1	3	5	6	4

Partition-Example

- The Operation of Partition(A, p, r)



$$A[j] < A[r]$$

Partition-Example

- The Operation of Partition(A, p, r)

p,i	j						r
2	8	7	1	3	5	6	4

$$i = i + 1, A[i] \leftrightarrow A[j], j = j + 1$$

Partition-Example

- The Operation of Partition(A, p, r)

p,i	j						r
2	8	7	1	3	5	6	4

$$A[j] > A[r]$$

Partition-Example

- The Operation of Partition(A, p, r)

p,i		j					r
2	**8**	**7**	**1**	**3**	**5**	**6**	**4**

$$Increase\ j\ by\ 1$$

Partition-Example

- The Operation of Partition(A, p, r)

p,i		j					r
2	8	7	1	3	5	6	4

$$A[j] > A[r]$$

Partition-Example

- The Operation of Partition(A, p, r)

p,i			j				r
2	8	7	1	3	5	6	4

$$Increase\ j\ by\ 1$$

Partition-Example

- The Operation of Partition(A, p, r)

p,i			j				r
2	8	7	1	3	5	6	4

$$A[j] < A[r]$$

Partition-Example

- The Operation of Partition(A, p, r)

p	i			j			r
2	1	7	8	3	5	6	4

$$i = i + 1, A[i] \leftrightarrow A[j], j = j + 1$$

Partition-Example

- The Operation of Partition(A, p, r)

p	i			j			r
2	1	7	8	3	5	6	4

$$A[j] < A[r]$$

Partition-Example

- The Operation of Partition(A, p, r)

p		i			j		r
2	1	3	8	7	5	6	4

$$i = i + 1, A[i] \leftrightarrow A[j], j = j + 1$$

Partition-Example

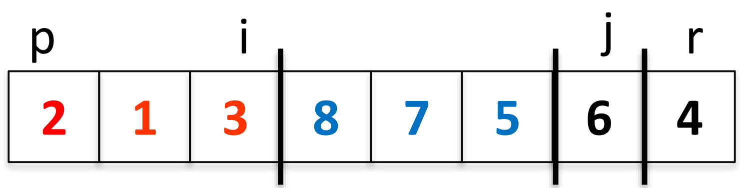
- The Operation of Partition(A, p, r)

p		i			j		r
2	1	3	8	7	5	6	4

$$A[j] > A[r]$$

Partition-Example

- The Operation of Partition(A, p, r)



$Increase\ j\ by\ 1$

Partition-Example

- The Operation of Partition(A, p, r)

p		i				j	r
2	1	3	8	7	5	6	4

$$A[j] > A[r]$$

Partition-Example

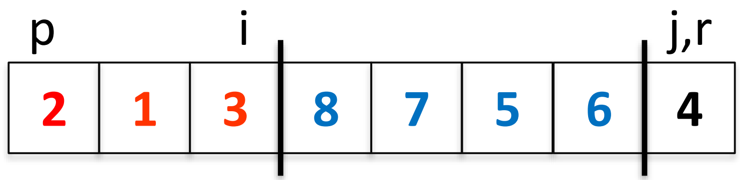
- The Operation of Partition(A, p, r)

p		i					j,r
2	1	3	8	7	5	6	4

$$Increase\ j\ by\ 1$$

Partition-Example

- The Operation of Partition(A, p, r)



Partition-Example

- The Operation of Partition(A, p, r)

p		i					j,r
2	1	3	4	7	5	6	8

$$A[i+1] \leftrightarrow A[r]$$

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition
$x \leftarrow A[r]$;//$A[r]$ is the pivot element

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition
$x \leftarrow A[r]$;//$A[r]$ is the pivot element
$i \leftarrow p - 1$;

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition
$x \leftarrow A[r]$;//$A[r]$ is the pivot element
$i \leftarrow p - 1$;
for $j \leftarrow p$ *to* $r - 1$ **do**

end

Partition - Pseudocode

Partition(A,p,r)

```
Input: An array A waiting to be sorted,the range of index p,r
Output: Index of the pivot after partition
x ← A[r];//A[r] is the pivot element
i ← p − 1;
for j ← p to r − 1 do
    if A[j] ≤ x then

    end
end
```

Partition - Pseudocode

Partition(A,p,r)

```
Input: An array A waiting to be sorted,the range of index p,r
Output: Index of the pivot after partition
x ← A[r];//A[r] is the pivot element
i ← p − 1;
for j ← p to r − 1 do
    if A[j] ≤ x then
        i ← i + 1;
        exchange A[i] and A[j];
    end
end
```

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition

```
x ← A[r];//A[r] is the pivot element
i ← p − 1;
for j ← p to r − 1 do
    if A[j] ≤ x then
        i ← i + 1;
        exchange A[i] and A[j];
    end
end
exchangeA[i + 1] and A[r];//Put pivot in position
```

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$

Output: Index of the pivot after partition

$x \leftarrow A[r]$;//$A[r]$ is the pivot element

$i \leftarrow p - 1$;

for $j \leftarrow p$ *to* $r - 1$ **do**

    if $A[j] \leq x$ **then**

        $i \leftarrow i + 1$;

        exchange $A[i]$ and $A[j]$;

    end

end

exchange$A[i + 1]$ and $A[r]$;//Put pivot in position

return

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition

```
x ← A[r];//A[r] is the pivot element
i ← p − 1;
for j ← p to r − 1 do
    if A[j] ≤ x then
        i ← i + 1;
        exchange A[i] and A[j];
    end
end
exchangeA[i + 1] and A[r];//Put pivot in position
return i + 1;//q ← i + 1
```

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition

```
x ← A[r];//A[r] is the pivot element
i ← p − 1;
for j ← p to r − 1 do
    if A[j] ≤ x then
        i ← i + 1;
        exchange A[i] and A[j];
    end
end
exchangeA[i + 1] and A[r];//Put pivot in position
return i + 1;//q ← i + 1
```

- Running time is O(   )

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p},\boldsymbol{r}$
Output: Index of the pivot after partition

$x \leftarrow A[r]$;$//A[r]$ is the pivot element
$i \leftarrow p - 1$;
for $j \leftarrow p$ *to* $r - 1$ **do**
  if $A[j] \leq x$ **then**
    $i \leftarrow i + 1$;
    exchange $A[i]$ and $A[j]$;
  end
end
exchange$A[i + 1]$ and $A[r]$;//Put pivot in position
return $i + 1$;$//q \leftarrow i + 1$

- Running time is O(r – p)

Partition - Pseudocode

Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Index of the pivot after partition

```
x ← A[r];//A[r] is the pivot element
i ← p − 1;
for j ← p to r − 1 do
    if A[j] ≤ x then
        i ← i + 1;
        exchange A[i] and A[j];
    end
end
exchangeA[i + 1] and A[r];//Put pivot in position
return i + 1;//q ← i + 1
```

- Running time is O(r – p)
  - linear in the length of the array A[p..r]

Quicksort

Quicksort(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Sorted array $\boldsymbol{A}$
if $p < r$ **then**
    $q \leftarrow$Partition(A, p, r);
    Quicksort(A,   );
    Quicksort(A,   );
end
return A;

Quicksort

Quicksort(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Sorted array $\boldsymbol{A}$
if $p < r$ **then**
  $q \leftarrow$Partition(A, p, r);
  Quicksort$(A, p, q - 1)$;
  Quicksort$(A,$ $)$;
end
return A;

Quicksort

Quicksort(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Sorted array $\boldsymbol{A}$
if $p < r$ **then**
  $q \leftarrow$Partition(A, p, r);
  Quicksort$(A, p, q - 1)$;
  Quicksort$(A, q + 1, r)$;
end
return A;

A Divide-and-Conquer Framework

Quicksort

Quicksort(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Sorted array $\boldsymbol{A}$
if $p < r$ **then**
    $q \leftarrow$Partition(A, p, r);
    Quicksort($A, p, q-1$);
    Quicksort($A, q+1, r$);
end
return A;

A Divide-and-Conquer Framework

- If we could always partition the array into halves, then we have the recurrence $T(n) \le 2T(n/2) + O(n)$, hence $T(n) = O(n \log n)$.

Quicksort

Quicksort(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: Sorted array $\boldsymbol{A}$
if $p < r$ **then**
    $q \leftarrow$Partition(A, p, r);
    Quicksort$(A, p, q - 1)$;
    Quicksort$(A, q + 1, r)$;
end
return A;

A Divide-and-Conquer Framework

- If we could always partition the array into halves, then we have the recurrence $T(n) \leq 2T(n/2) + O(n)$, hence $T(n) = O(n \log n)$.
- However, if we always get unlucky with very unbalanced partitions, then $T(n) \leq T(n-1) + O(n)$, hence $T(n) = O(n^2)$.

Outline

- Quicksort Problem
  - Basic partition
  - Randomized partition and randomized quicksort
  - Analysis of the randomized quicksort

Randomized-Partition(A, p, r)

- Idea
  - In the algorithm Partition(A, p, r), A[r] is always used as the pivot *x* to partition the array A[p..r].

Randomized-Partition(A, p, r)

- Idea
  - In the algorithm Partition(A, p, r), A[r] is always used as the pivot *x* to partition the array A[p..r].
  - In the algorithm Randomized-Partition(A, p, r), we randomly choose an j , p ≤ j ≤ r , and use A[j] as pivot.

Randomized-Partition(A, p, r)

- Idea
  - In the algorithm Partition(A, p, r), A[r] is always used as the pivot x to partition the array A[p..r].
  - In the algorithm Randomized-Partition(A, p, r), we randomly choose an j , p ≤ j ≤ r , and use A[j] as pivot.
  - Idea is that if we choose randomly, then the chance that we get unlucky every time is extremely low.

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Partition
  - Let random(p, r) be a pseudorandom-number generator that returns a random number between p and r.

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Partition
  - Let random(p, r) be a pseudorandom-number generator that returns a random number between p and r.

Randomized-Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: A random index in $[p..j]$

Partition(A, p, r);
return j;

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Partition
  - Let random(p, r) be a pseudorandom-number generator that returns a random number between p and r.

Randomized-Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: A random index in $[p..j]$
$j \leftarrow$random(p, r);

Partition(A, p, r);
return j;

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Partition
  - Let random(p, r) be a pseudorandom-number generator that returns a random number between p and r.

Randomized-Partition(A,p,r)

Input: An array $\boldsymbol{A}$ waiting to be sorted,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$
Output: A random index in $[p..j]$
$j \leftarrow$random(p, r);
exchange $A[r]$ and $A[j]$;
Partition(A, p, r);
return j;

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Quicksort
  - We make use of the Randomized-Partition idea to develop a new version of quicksort.

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Quicksort
  - We make use of the Randomized-Partition idea to develop a new version of quicksort.

Randomized-Quicksort(A,p,r)

```
Input: An array A waiting to be sorted,the range of index p,r
Output: Sorted array A
if p < r then
    q ← Randomized-Partition(A, p, r);
    Randomized-Quicksort(A,          );
    Randomized-Quicksort(A,          );
end
return A;
```

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Quicksort
  - We make use of the Randomized-Partition idea to develop a new version of quicksort.

Randomized-Quicksort(A,p,r)

```
Input: An array A waiting to be sorted,the range of index p,r
Output: Sorted array A
if p < r then
    q ← Randomized-Partition(A, p, r);
    Randomized-Quicksort(A, p, q − 1);
    Randomized-Quicksort(A,          );
end
return A;
```

Randomized-Partition(A, p, r)

- Pseudocode of Randomized-Quicksort
  - We make use of the Randomized-Partition idea to develop a new version of quicksort.

Randomized-Quicksort(A,p,r)

```
Input: An array A waiting to be sorted,the range of index p,r
Output: Sorted array A
if p < r then
    q ← Randomized-Partition(A, p, r);
    Randomized-Quicksort(A, p, q − 1);
    Randomized-Quicksort(A, q + 1, r);
end
return A;
```

Quicksort - Example

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

Quicksort - Example

Divide

2	8	7	1	3	5	6	4

Quicksort - Example

Divide

2	8	7	1	3	5	6	4

Quicksort - Example

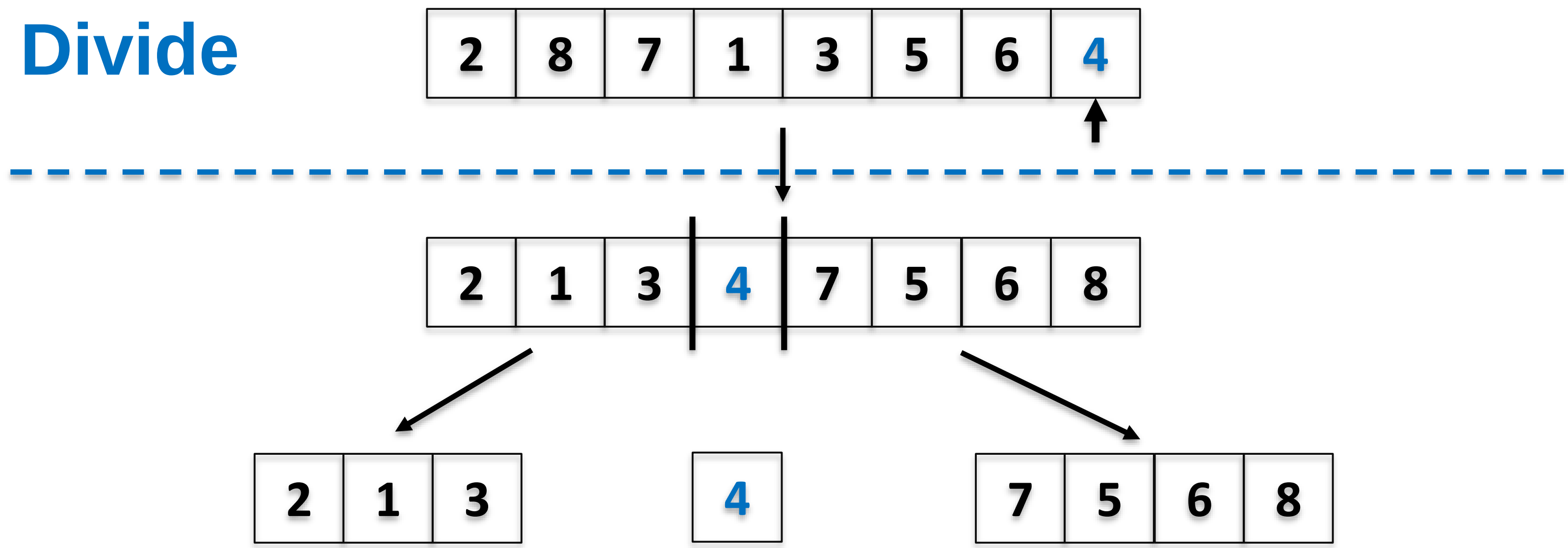
Divide

2	8	7	1	3	5	6	4

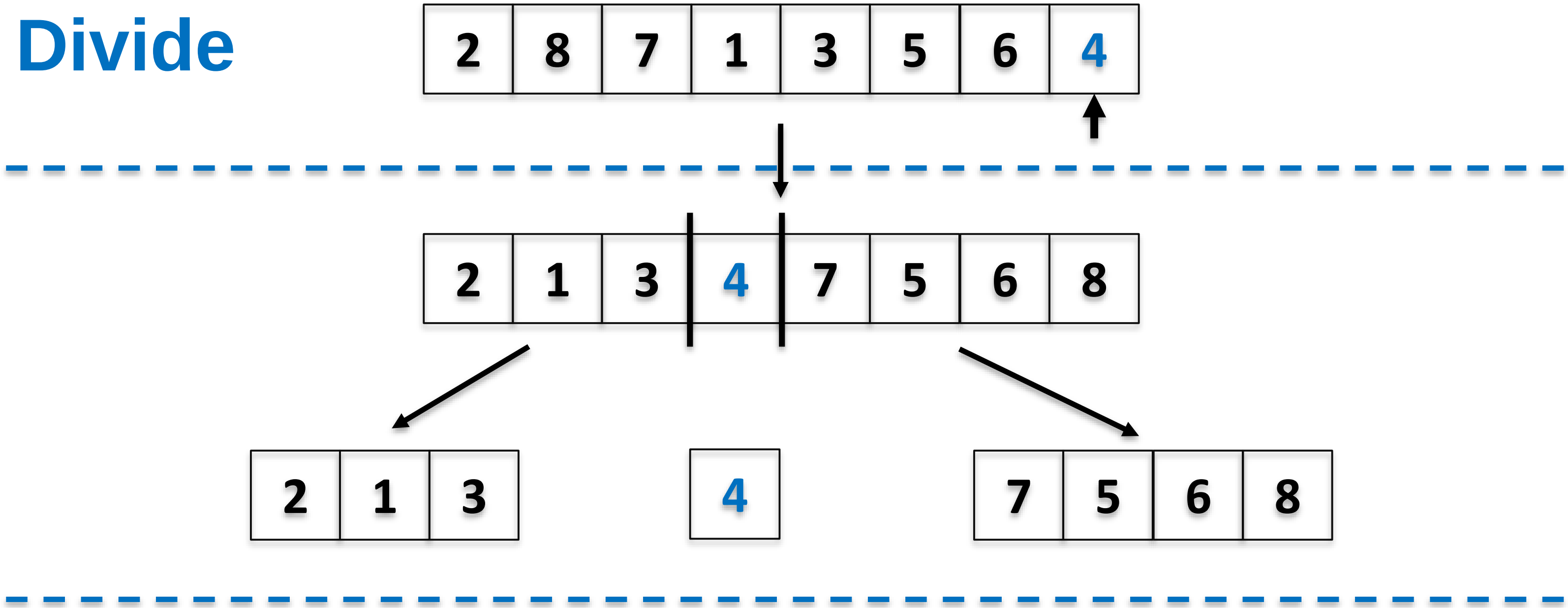
2	1	3	4	7	5	6	8

Quicksort - Example

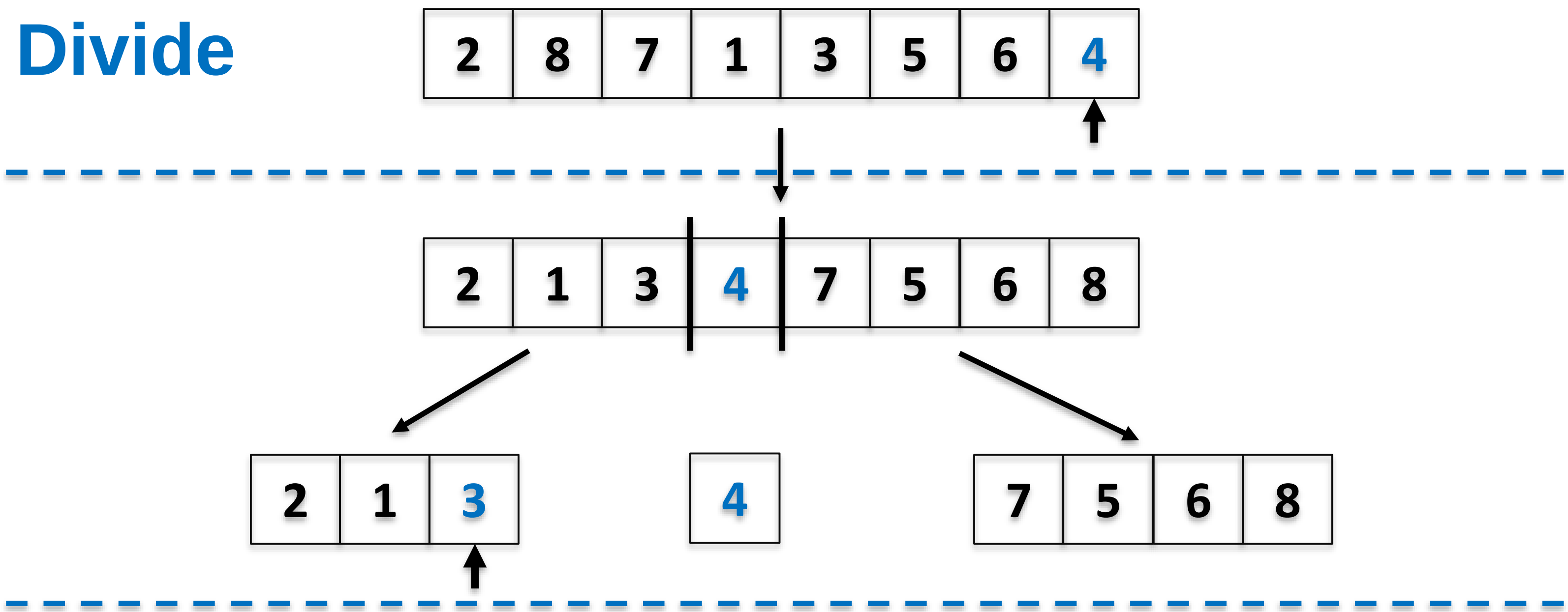
Divide



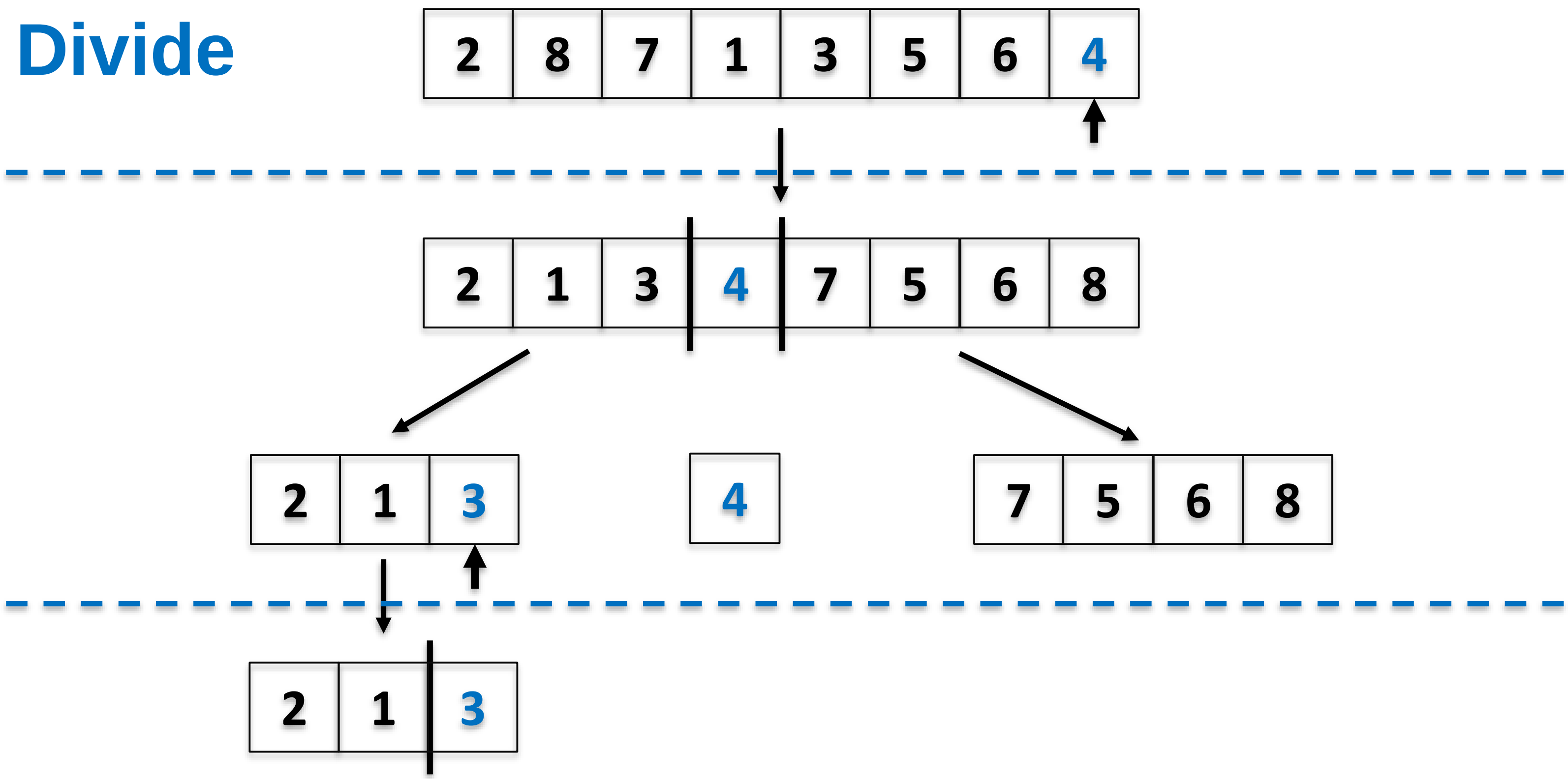
Quicksort - Example



Quicksort - Example

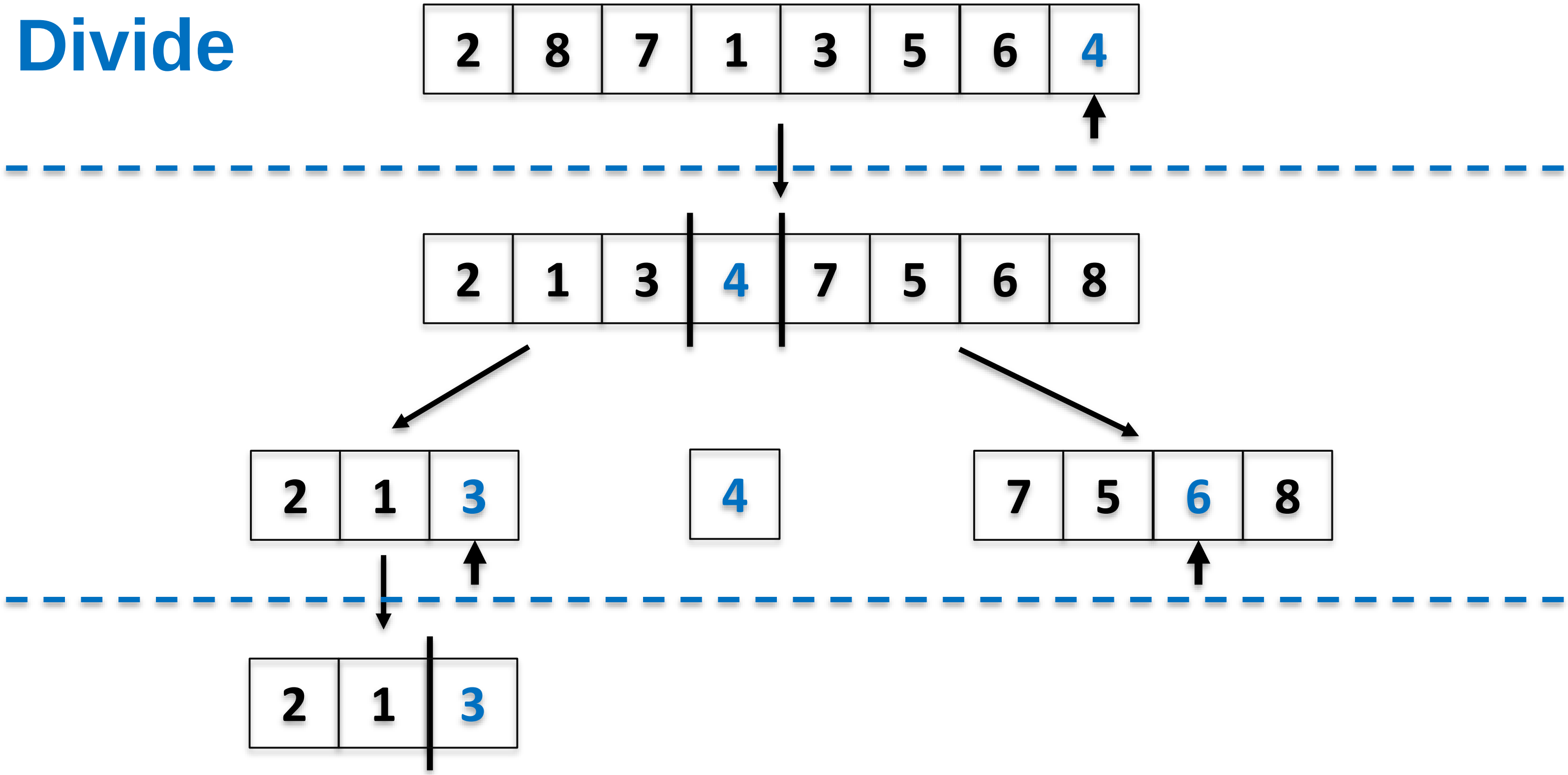


Quicksort - Example

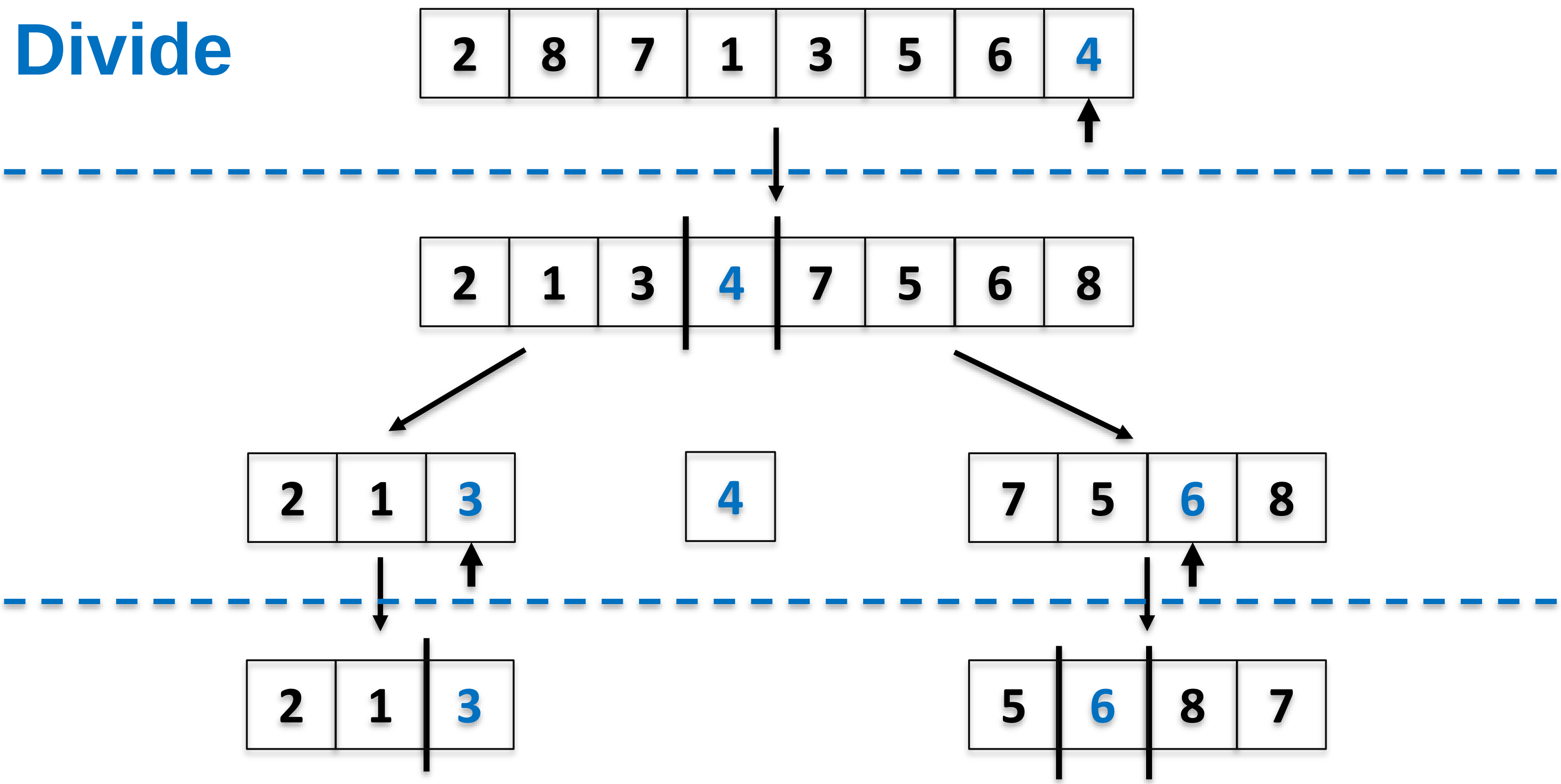


Quicksort - Example

Divide

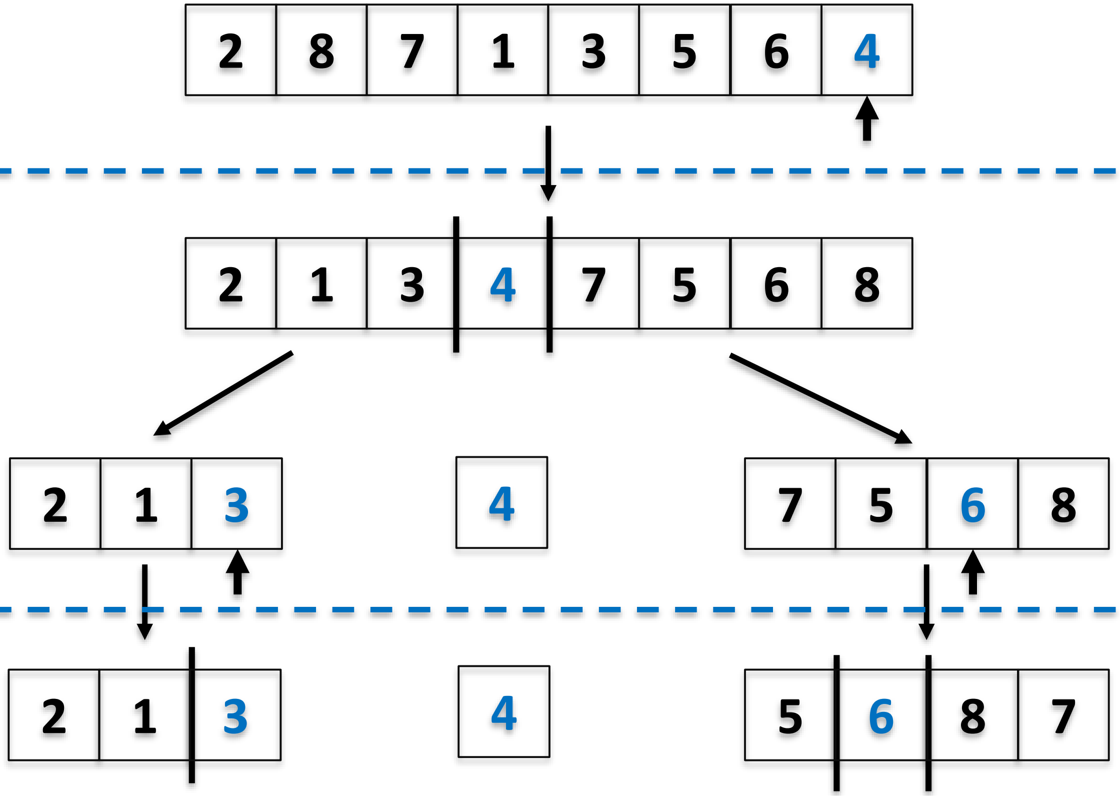


Quicksort - Example



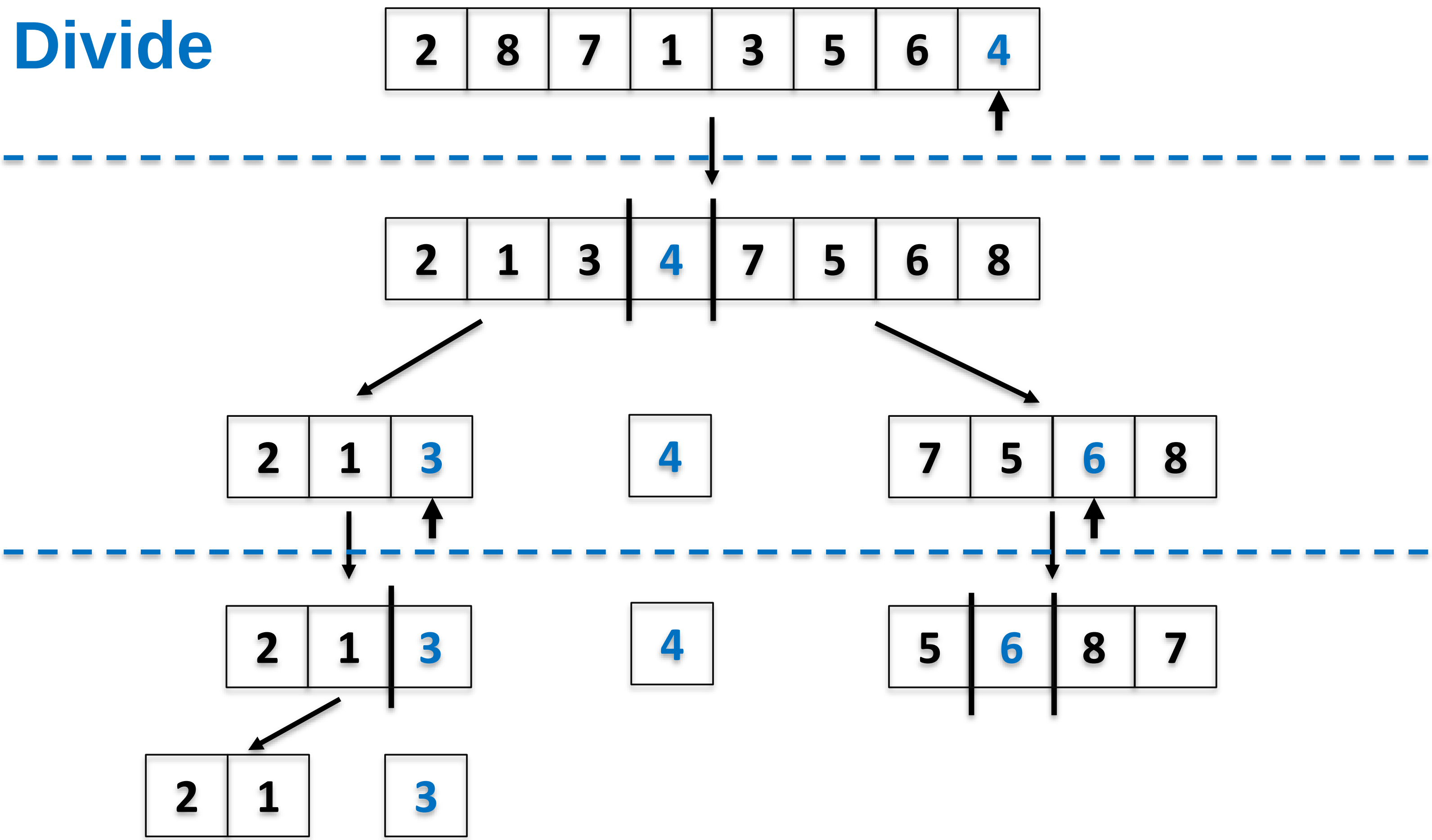
Quicksort - Example

Divide

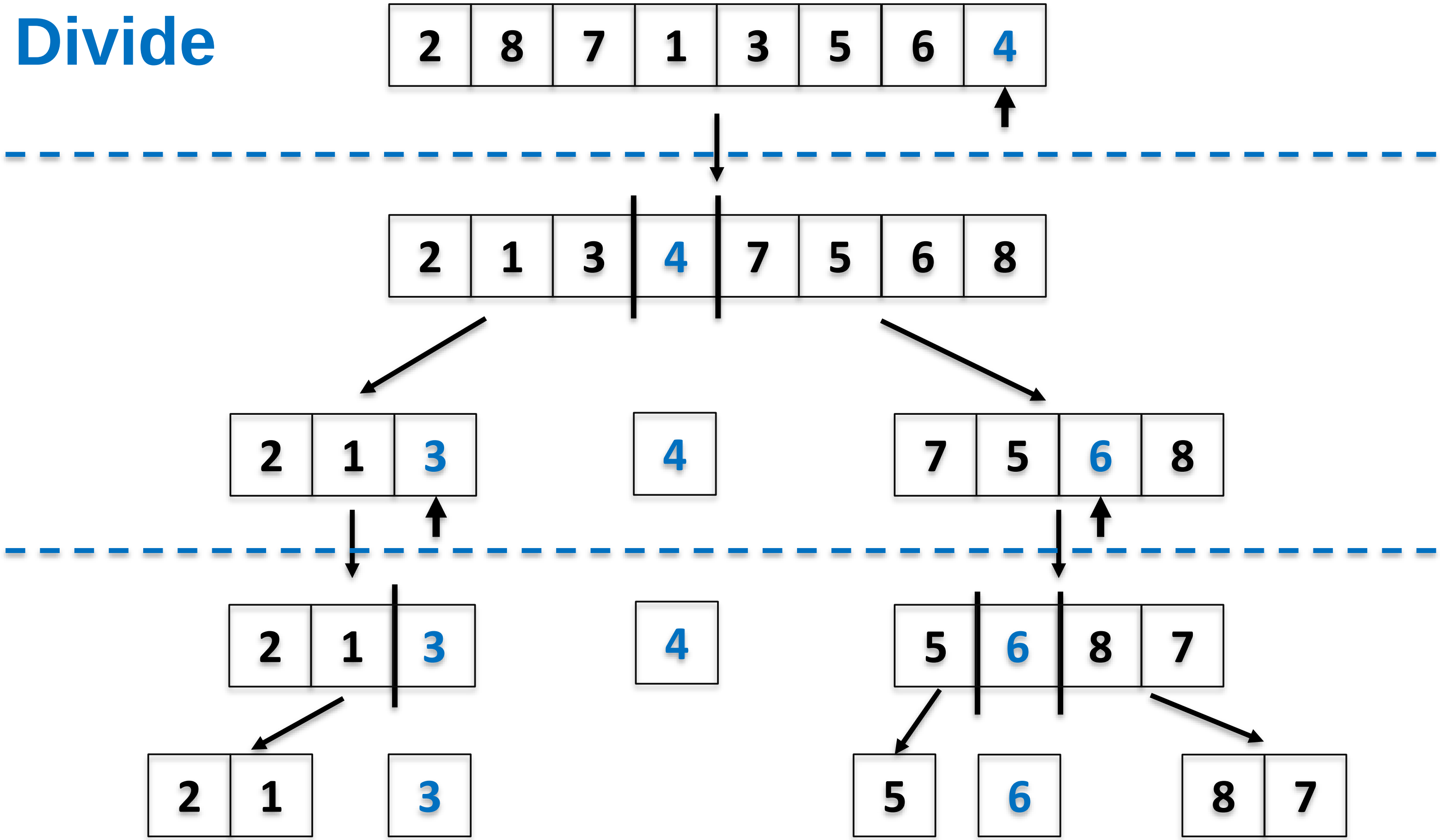


Quicksort - Example

Divide

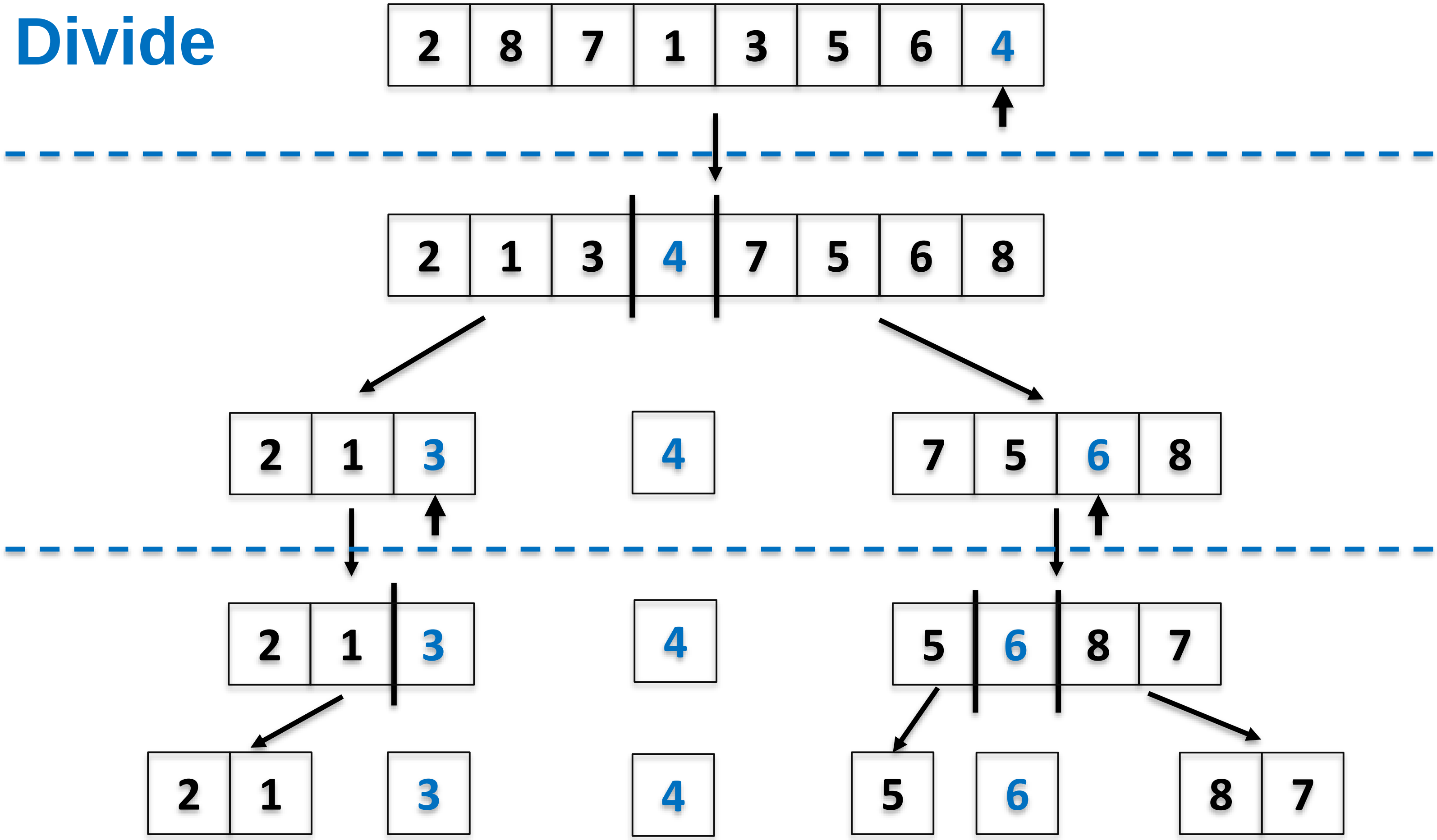


Quicksort - Example



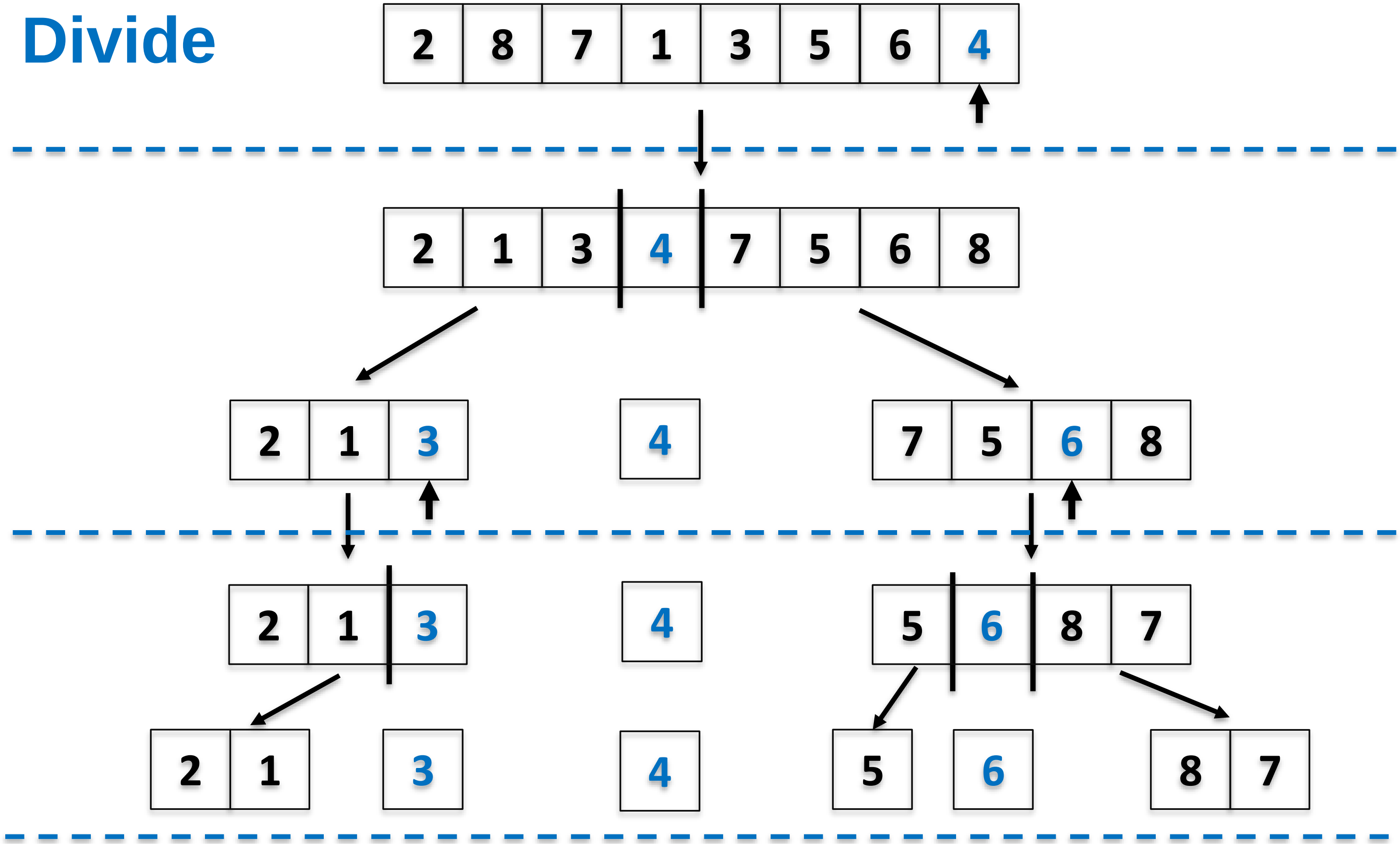
Quicksort - Example

Divide

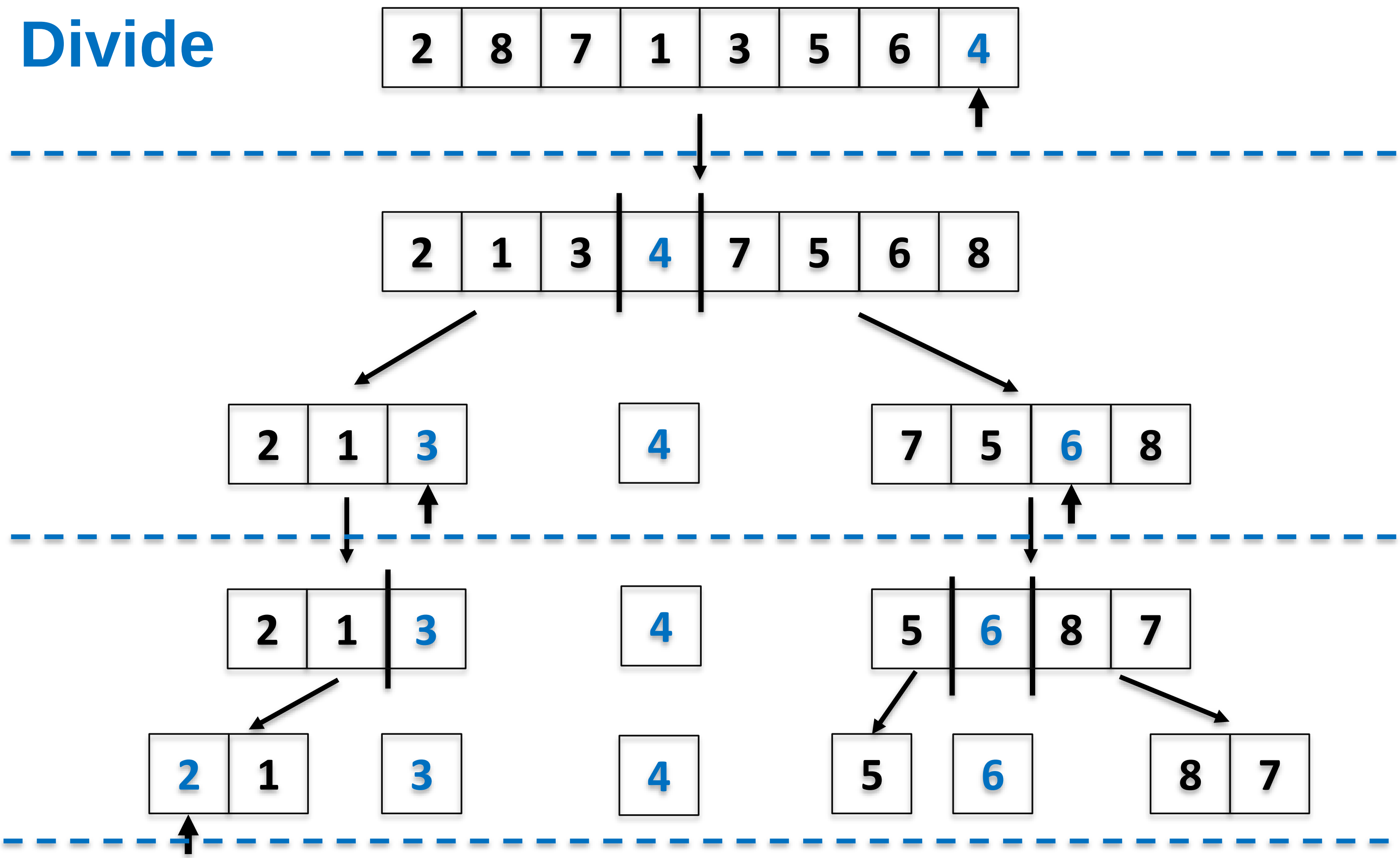


Quicksort - Example

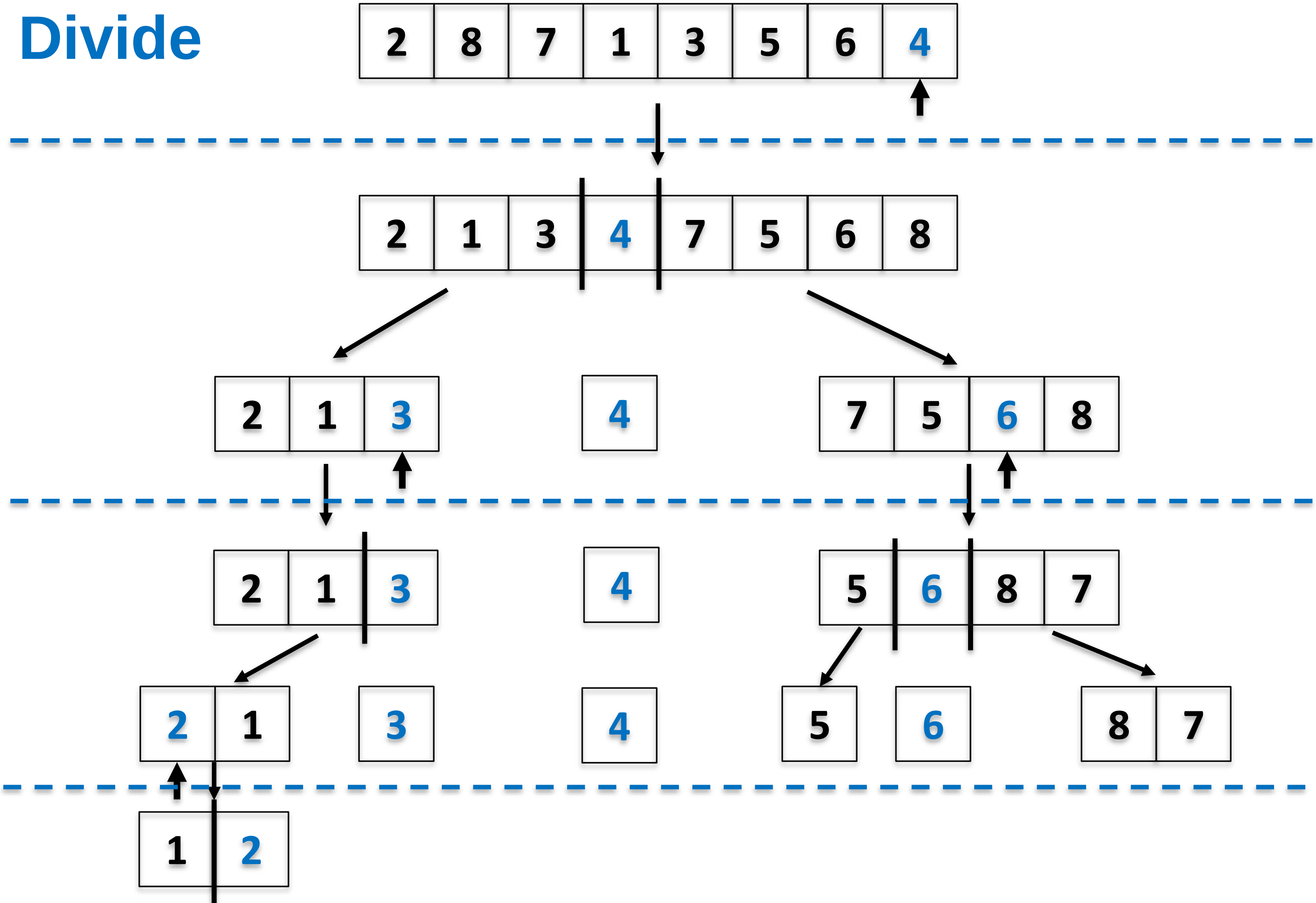
Divide



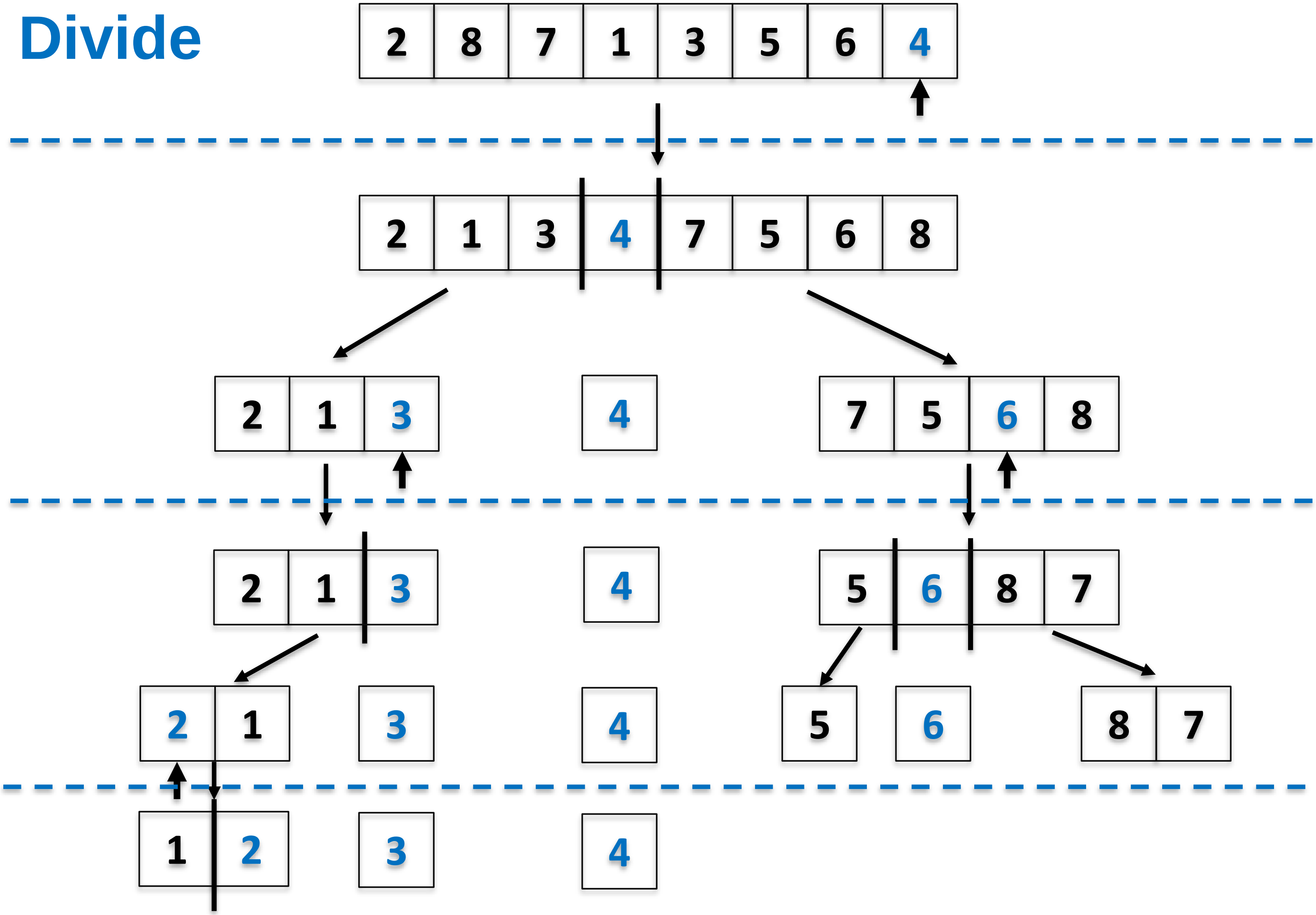
Quicksort - Example



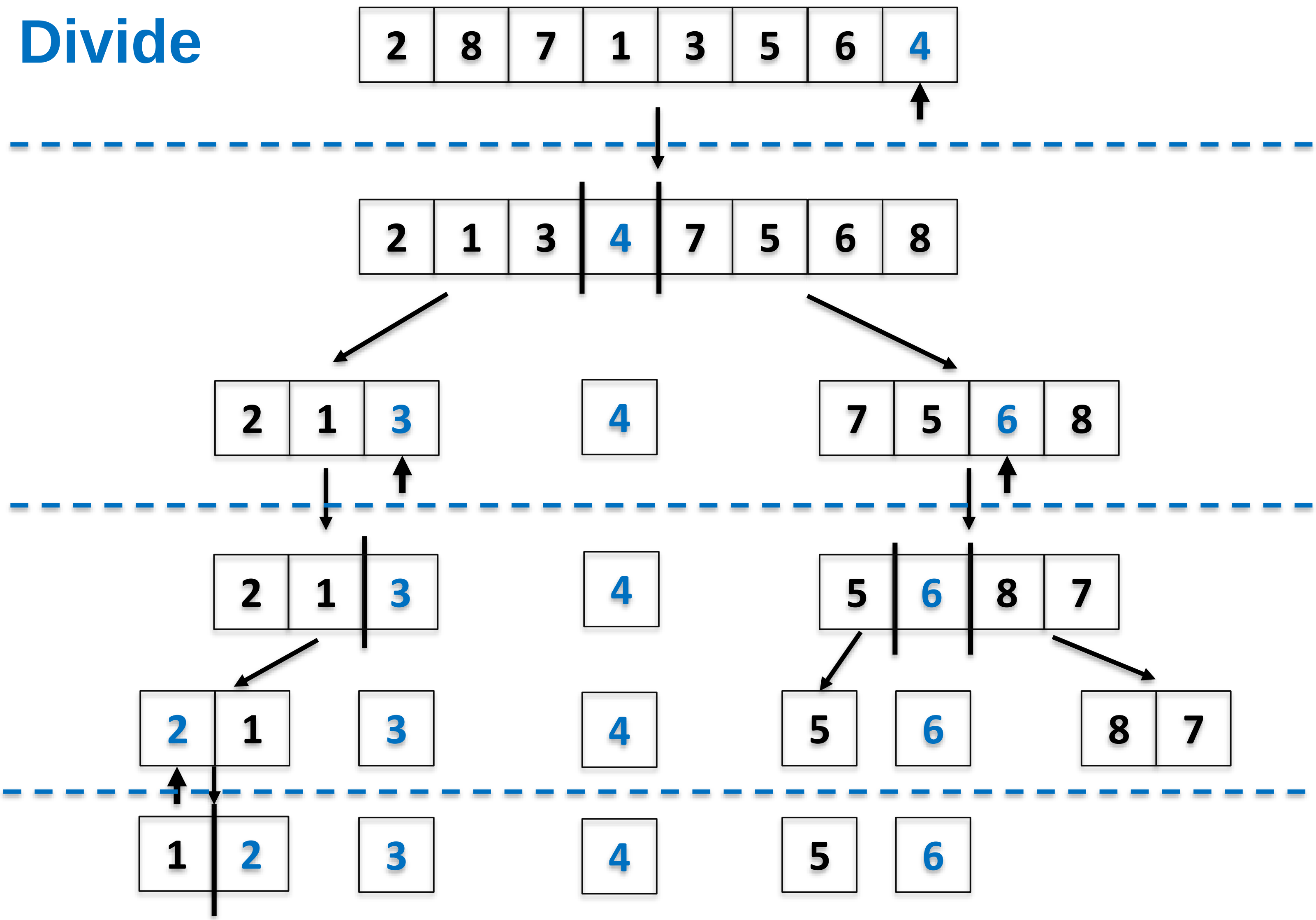
Quicksort - Example



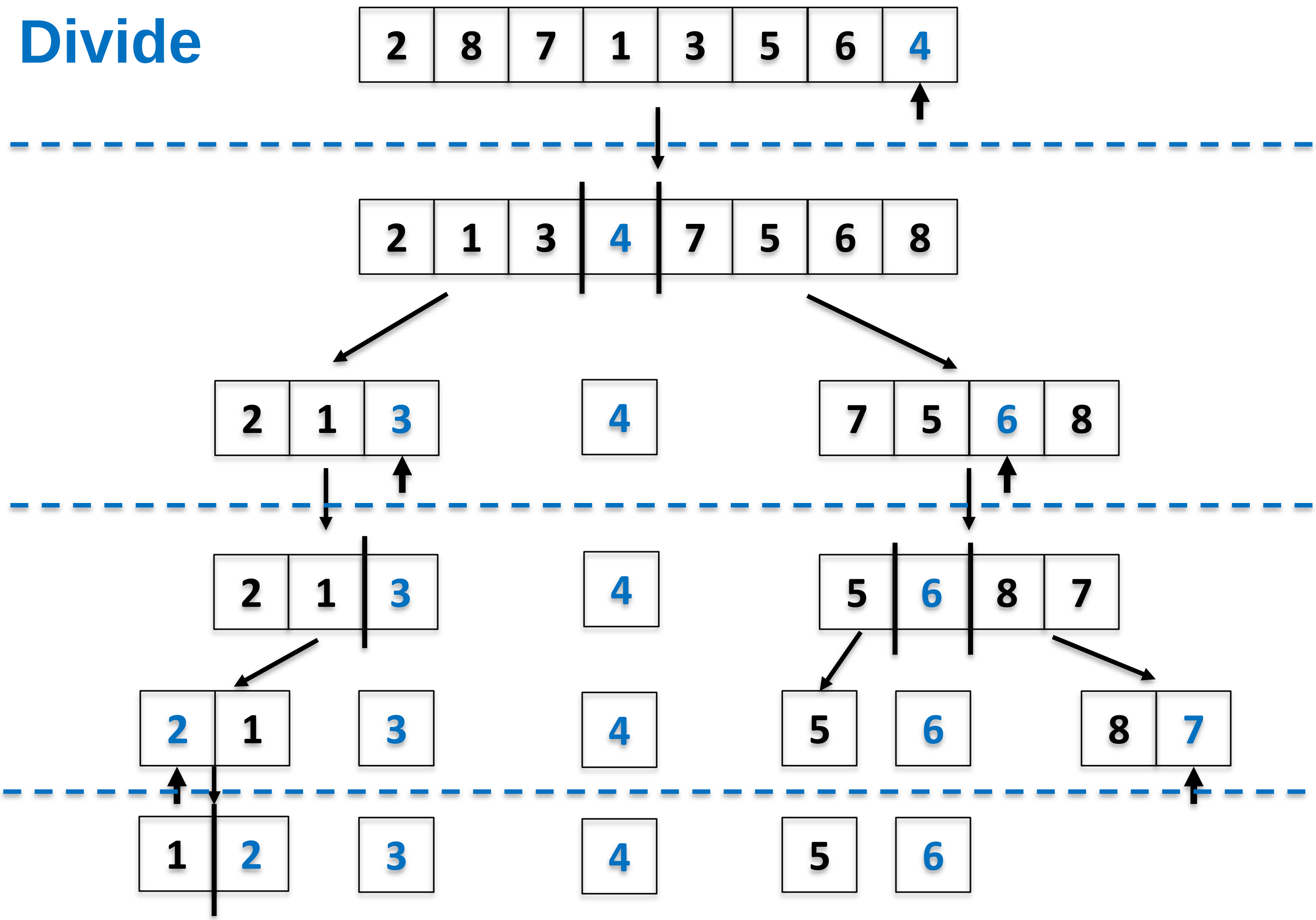
Quicksort - Example



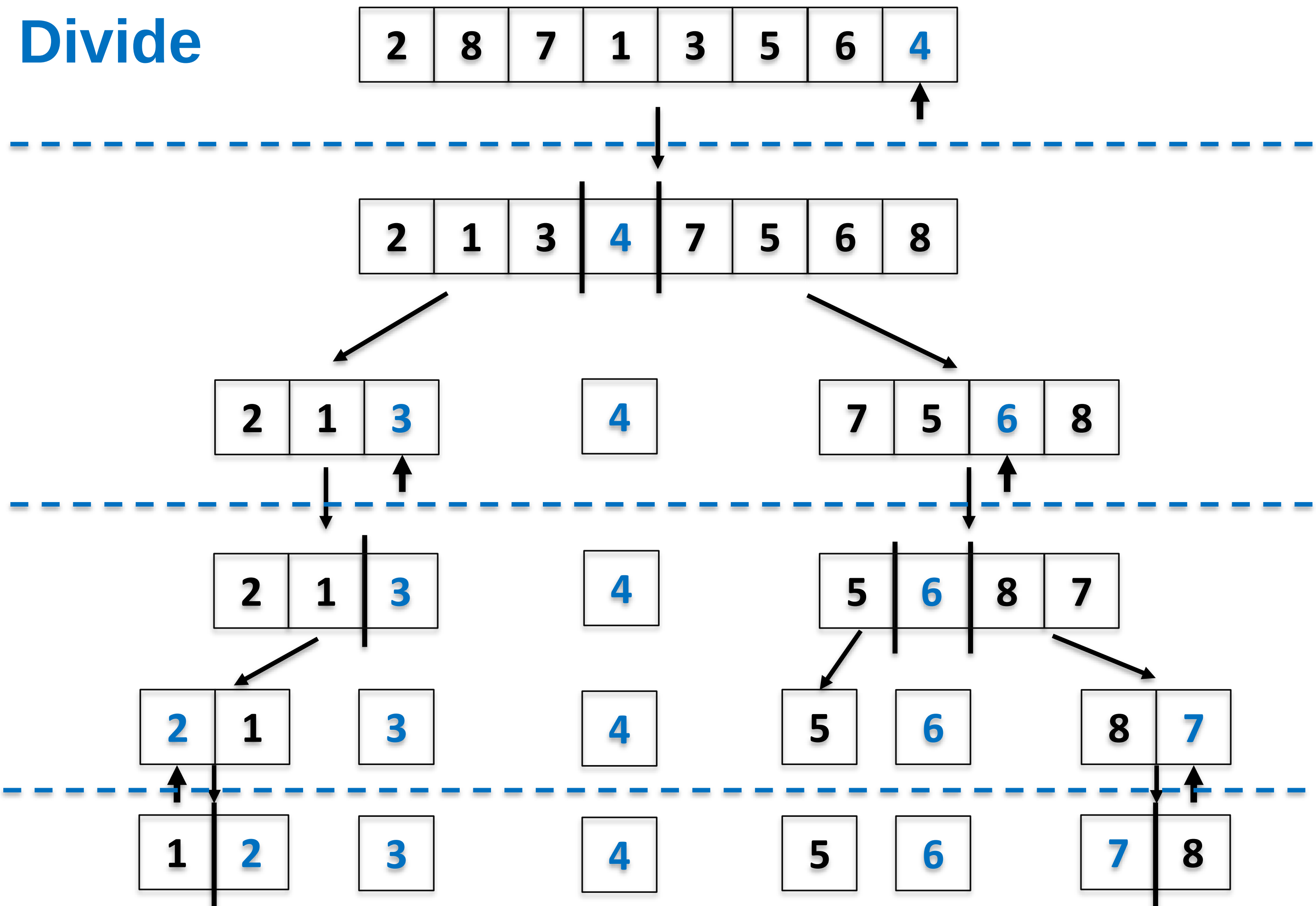
Quicksort - Example



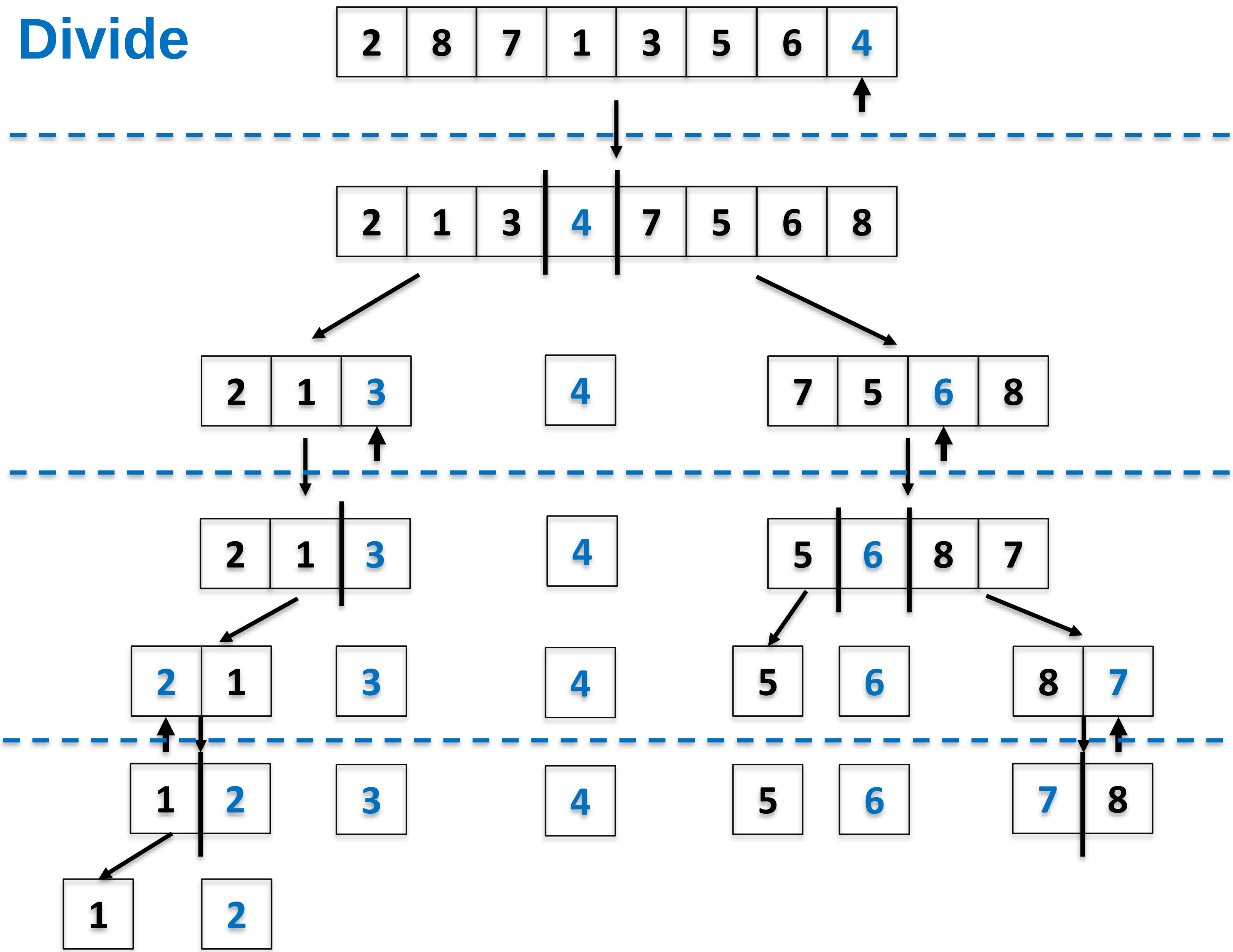
Quicksort - Example



Quicksort - Example

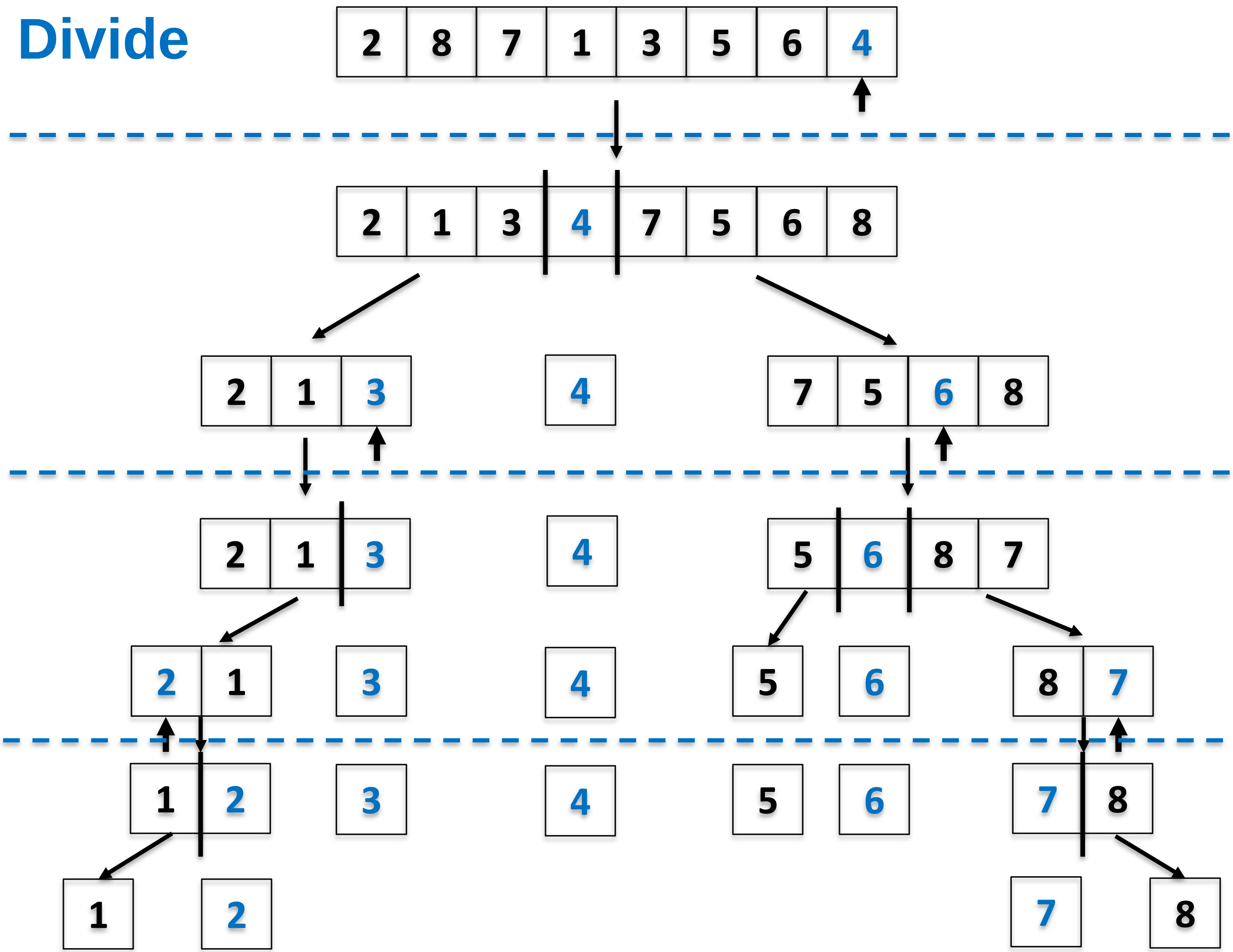


Quicksort - Example

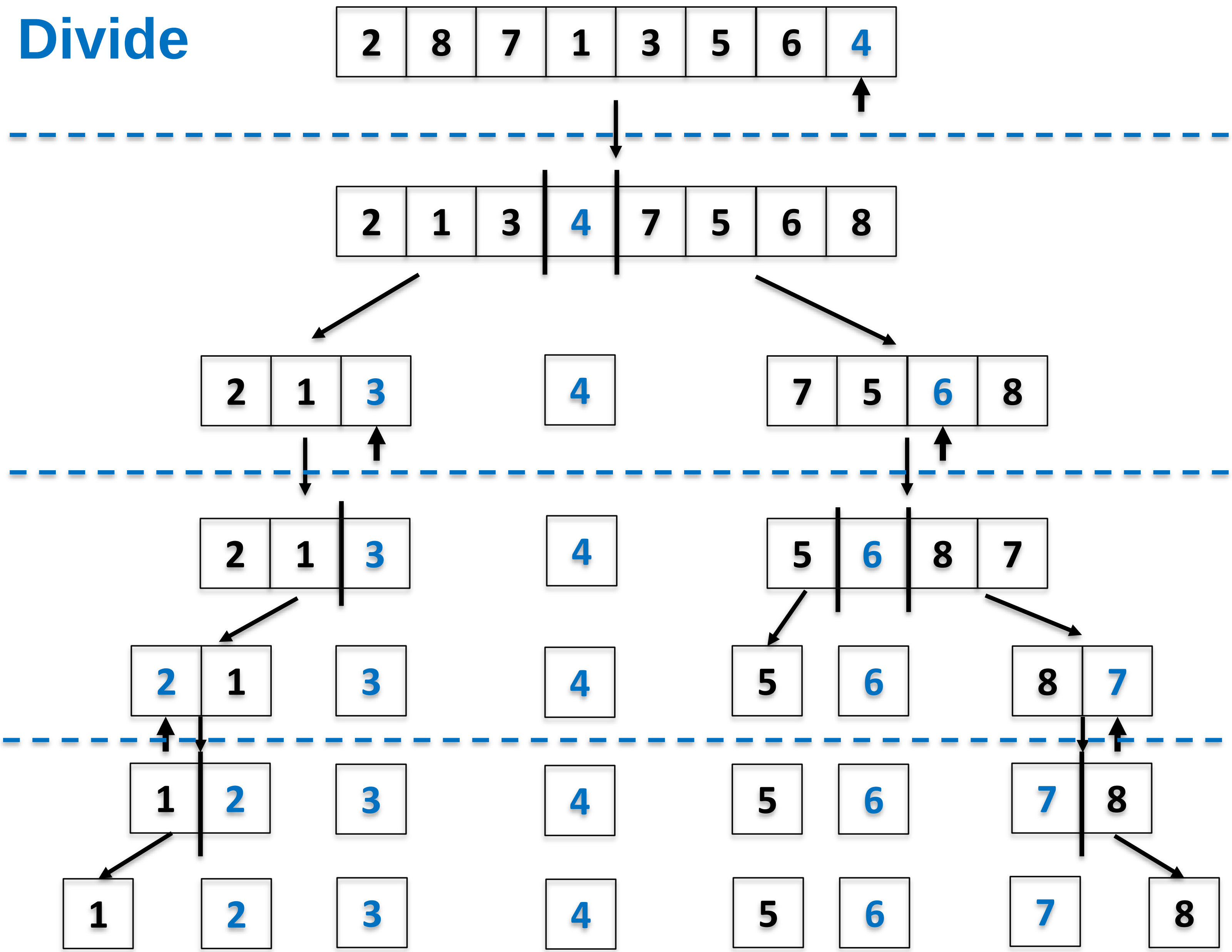


Quicksort - Example

Divide



Quicksort - Example



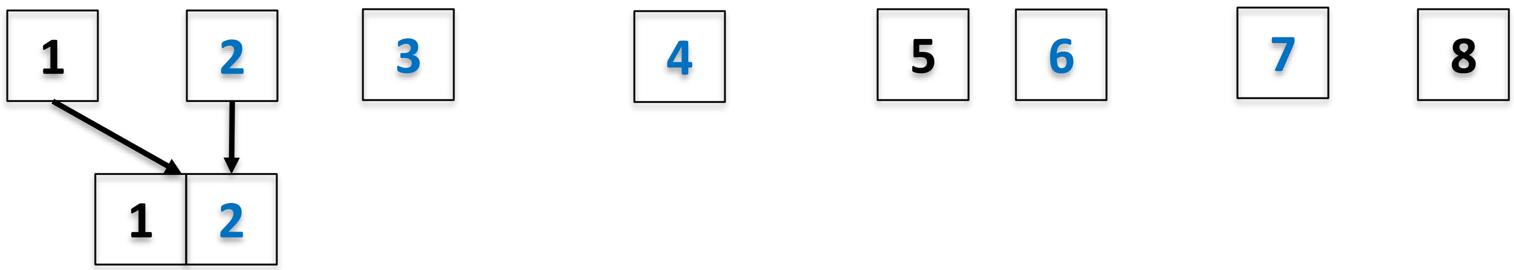
Quicksort - Example

Conquer

1	2	3	4	5	6	7	8

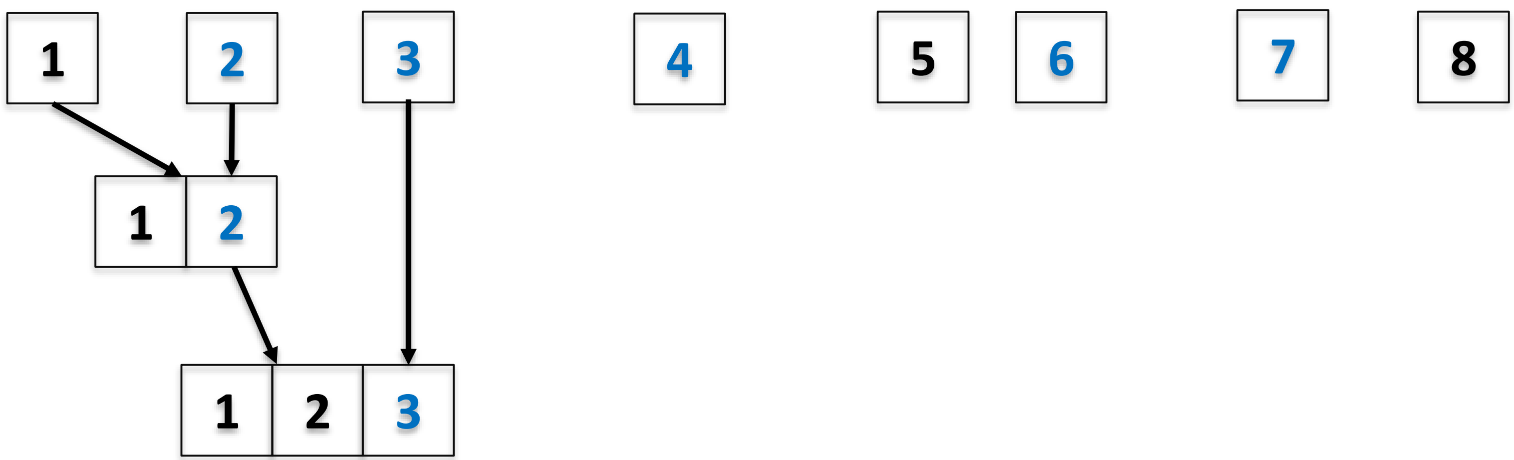
Quicksort - Example

Conquer



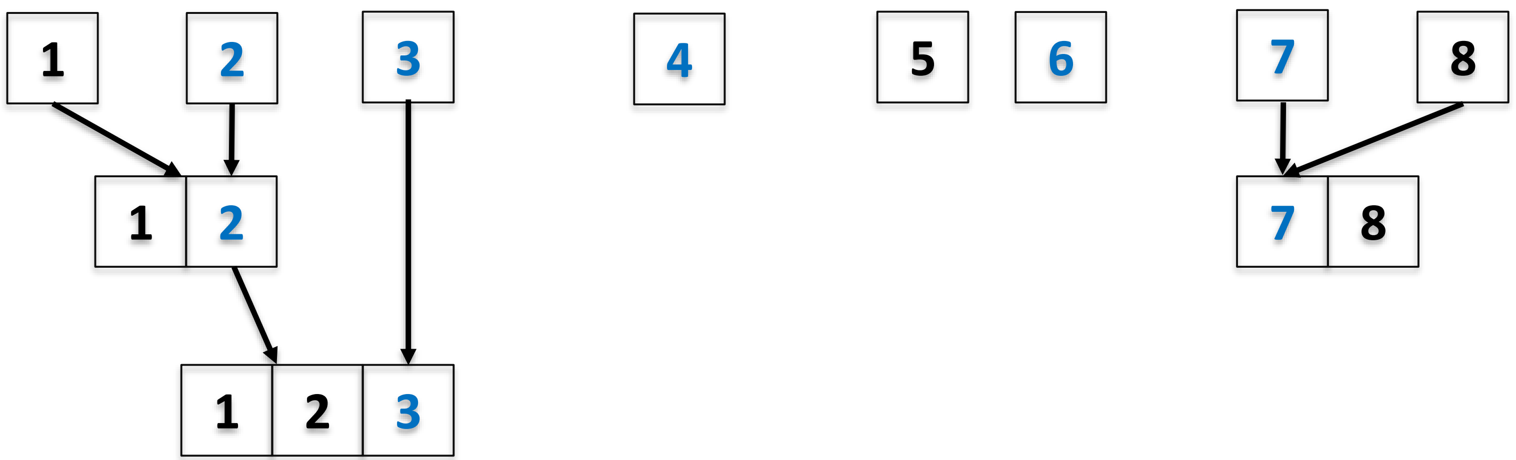
Quicksort - Example

Conquer



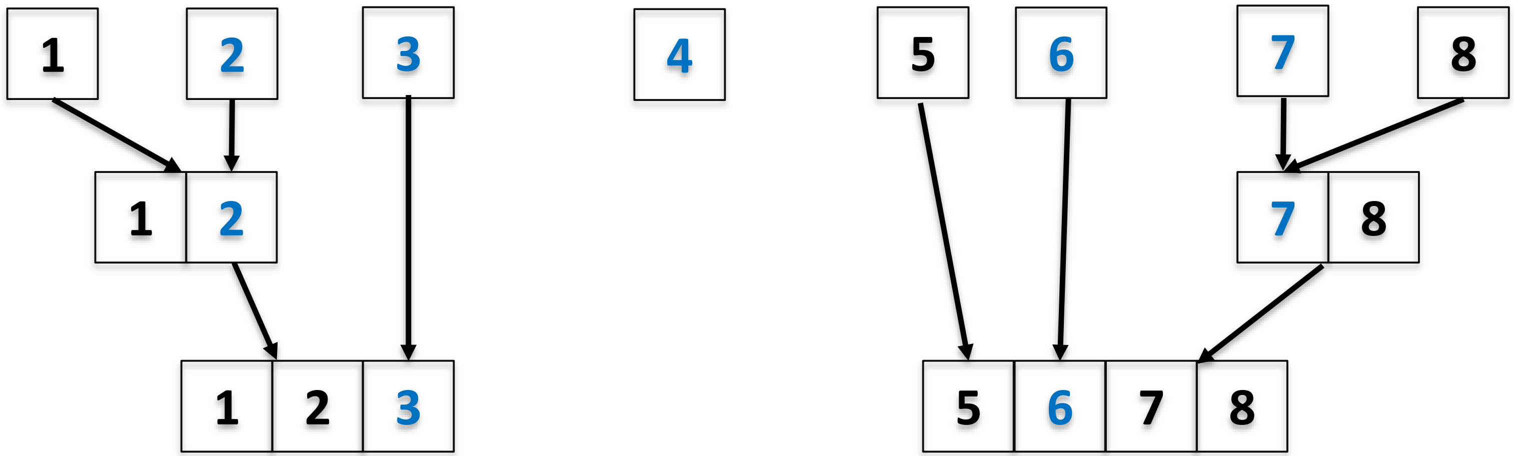
Quicksort - Example

Conquer



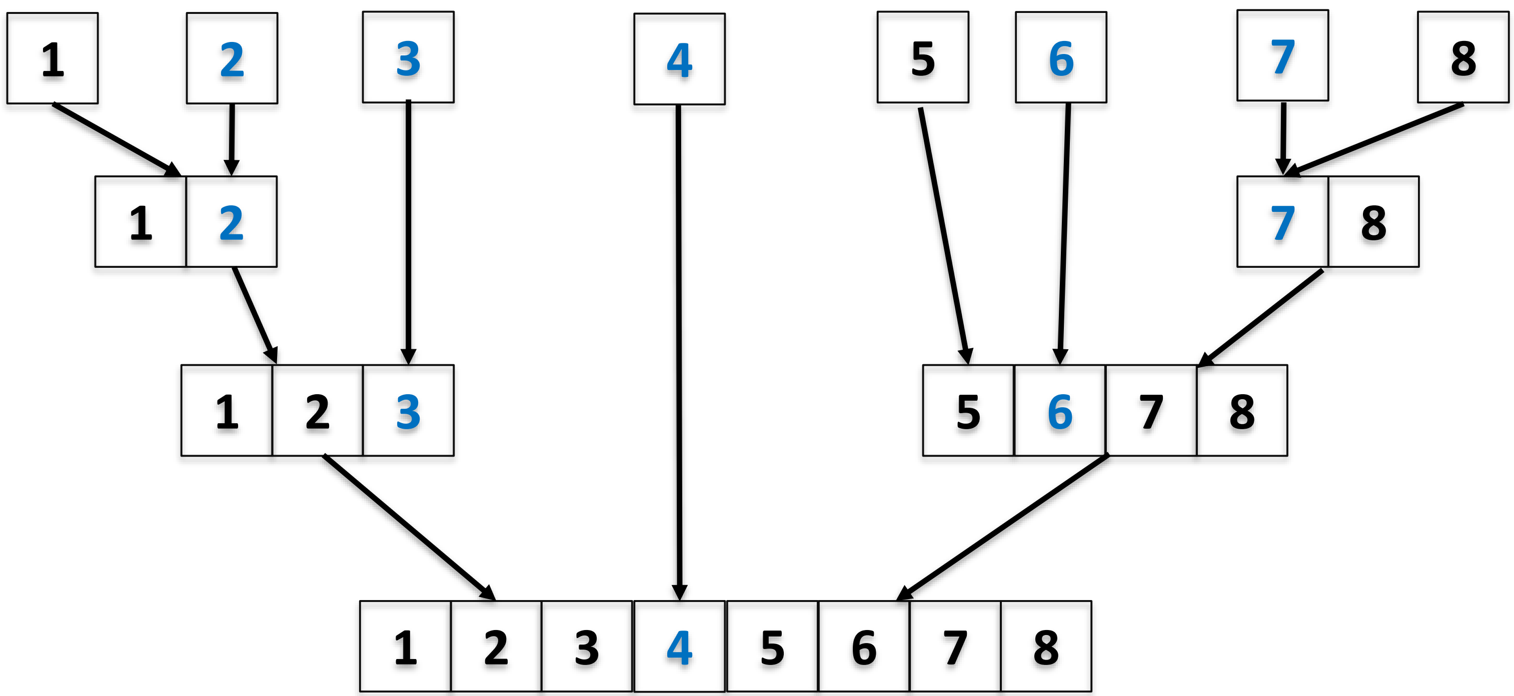
Quicksort - Example

Conquer



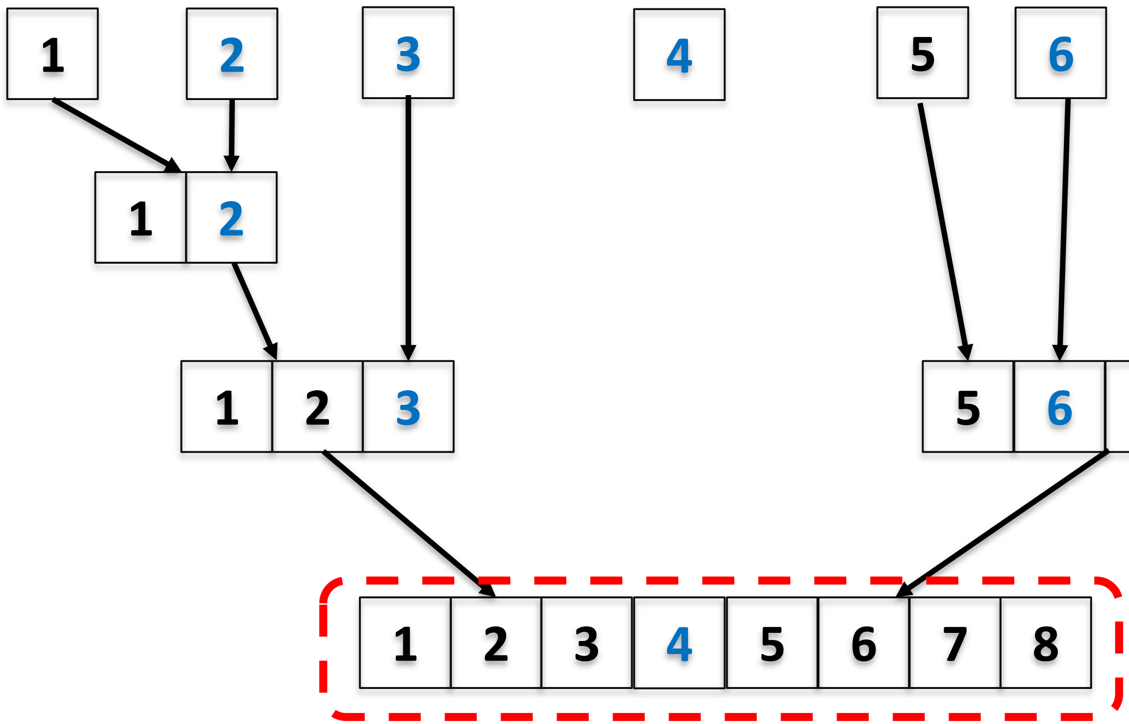
Quicksort - Example

Conquer



Quicksort - Example

Conquer



Outline

- Quicksort Problem
  - Basic partition
  - Randomized partition and randomized quicksort
  - Analysis of the randomized quicksort

Running Time of the Quicksort

- Measuring running time:

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) =$

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) +$

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n-m-1) +$

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$
- Worst Case:

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$
- Worst Case:

$$T(n) = T(0) + T(n-1) + O(n)$$

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n-m-1) + O(n)$
- Worst Case:

$$T(n) = T(0) + T(n-1) + O(n)$$
$$T(n) = O(\quad)$$

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$
- Worst Case:

$$T(n) = T(0) + T(n-1) + O(n)$$
$$T(n) = O(n^2)$$

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n-m-1) + O(n)$
- Worst Case:

$$T(n) \;=\; T(0) \;+\; T(n-1) \;+\; O(n)$$
$$T(n) = \; O(n^2)$$

- What inputs give worst case performance?

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n-m-1) + O(n)$
- Worst Case:

$$T(n) = T(0) + T(n-1) + O(n)$$
$$T(n) = O(n^2)$$

- What inputs give worst case performance?
  - Whether performance is the worst is not determined by input.

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$
- Worst Case:

$$T(n) = T(0) + T(n-1) + O(n)$$
$$T(n) = O(n^2)$$

- What inputs give worst case performance?
  - Whether performance is the worst is not determined by input.
  - An important property of randomized algorithms.

Running Time of the Quicksort

- Measuring running time:
  - The running time is dominated by the time spent in partition.
  - The running time of the partition procedure can be measured by the number of key comparisons.
- T(n): running time on array of size n.
- Recurrence: $T(n) = T(m) + T(n - m - 1) + O(n)$
- Worst Case:

$$T(n) = T(0) + T(n-1) + O(n)$$
$$T(n) = O(n^2)$$

- What inputs give worst case performance?
  - Whether performance is the worst is not determined by input.
  - An important property of randomized algorithms.
  - Worst case performance results only if the random number generator always produces the worst choice.

Randomized Algorithms

- Analysis for Randomized Algorithms:
  - Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen.

Randomized Algorithms

- Analysis for Randomized Algorithms:
  - Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen.
  - Use expected running time analysis for randomized algorithms!

Randomized Algorithms

- Analysis for Randomized Algorithms:
  - Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen.
  - Use expected running time analysis for randomized algorithms!

Average case analysis

- Used for deterministic algorithms

Expected case analysis

- Used for randomized algorithms

Randomized Algorithms

- Analysis for Randomized Algorithms:
  - Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen.
  - Use expected running time analysis for randomized algorithms!

Average case analysis

- Used for deterministic algorithms
- Assume the input is chosen randomly from some distribution

Expected case analysis

- Used for randomized algorithms
- Need to work for any given input

Randomized Algorithms

- Analysis for Randomized Algorithms:
  - Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen.
  - Use expected running time analysis for randomized algorithms!

Average case analysis

- Used for deterministic algorithms
- Assume the input is chosen randomly from some distribution
- Depends on assumptions on the input, weaker

Expected case analysis

- Used for randomized algorithms
- Need to work for any given input
- Randomization is inherent within the algorithm, stronger

Randomized Algorithms

- Analysis for Randomized Algorithms:
  - Worst-case doesn't make sense: for any given input, the worst case is very unlikely to happen.
  - Use expected running time analysis for randomized algorithms!

Average case analysis

- Used for deterministic algorithms
- Assume the input is chosen randomly from some distribution
- Depends on assumptions on the input, weaker
- Not required in this course

Expected case analysis

- Used for randomized algorithms
- Need to work for any given input
- Randomization is inherent within the algorithm, stronger
- Required in this course

Expected Case

- Two methods to analyze the expected running time of a divide-and-conquer randomized algorithm:
  - Old fashioned: Write our a recurrence on T(n), where T(n) is the expected running time of the algorithm on an input of size n, and solve it.
    —— (Almost) always works but needs complicated maths.

Expected Case

- Two methods to analyze the expected running time of a divide-and-conquer randomized algorithm:
  - Old fashioned: Write a recurrence on T(n), where T(n) is the expected running time of the algorithm on an input of size n, and solve it.
    —— (Almost) always works but needs complicated maths.
  - New: Indicator variables.
    —— Simple and elegant, but needs practice to master.

Expected Case

- Two facts about key comparisons:

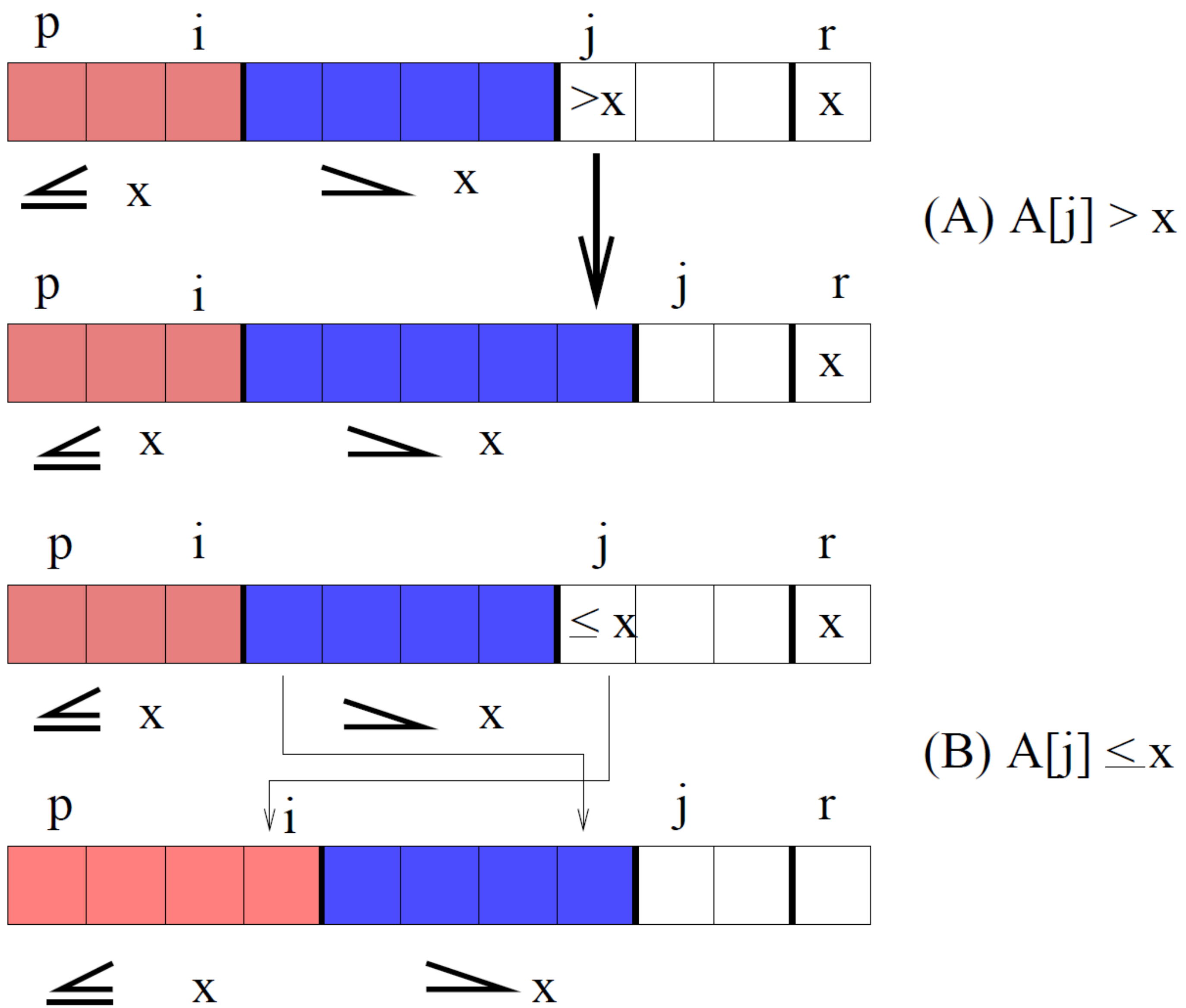
Expected Case

- Two facts about key comparisons:
  - When a pivot is selected, the pivot is compared with every other elements, then the elements are partitioned into two parts accordingly

Expected Case

- Two facts about key comparisons:
  - When a pivot is selected, the pivot is compared with every other elements, then the elements are partitioned into two parts accordingly
  - Elements in different partitions are never compared with each other in all operations

Expected Case



Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] =$$

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] =$$

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$

$$=$$

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 \ < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [\Pr\{$$

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} [\Pr\{z_i \textit{ is compared with } z_j\} \times 1$$

$$+ \Pr\{z_i \textit{ is not compared with } z_j\} \times 0]$$

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [\Pr\{z_i \textit{ is compared with } z_j\} \times 1$$

$$+ \Pr\{z_i \textit{ is not compared with } z_j\} \times 0]$$

$$=$$

Expected Running Time

- Expected Running Time of Randomized-Quicksort:
  - Let $z_1 < z_2 < \cdots < z_n$ be the n elements in sorted order
  - X: total number of comparisons performed in all calls to randomized-partition
  - X_{ij} : number of comparisons between z_i and z_j
    - can only be 0 or 1

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}]$$

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

$$+ \Pr\{z_i \text{ is not compared with } z_j\} \times 0]$$

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

Observations

For $1 \le i \le j \le n$, let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

- remember $z_i < z_{i+1} < \cdots < z_j$

Observations

For $1 \leq i \leq j \leq n$, let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

- remember $z_i < z_{i+1} < \cdots < z_j$

Observations:

- Partition divides an array into three segments, left, pivot, and Right.
- When z_i and z_j are first placed in DIFFERENT segments of the array by partitioning, the pivot is one of the elements in Z_{ij}

Observations

For $1 \leq i \leq j \leq n$, let $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$

- remember $z_i < z_{i+1} < \cdots < z_j$

Observations:

- Partition divides an array into three segments, left, pivot, and Right.
- When z_i and z_j are first placed in DIFFERENT segments of the array by partitioning, the pivot is one of the elements in Z_{ij}
- If the pivot is either z_i or z_j

Observations

For $1 \le i \le j \le n$, let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

- remember $z_i < z_{i+1} < \cdots < z_j$

Observations:

- Partition divides an array into three segments, left, pivot, and Right.
- When z_i and z_j are first placed in DIFFERENT segments of the array by partitioning, the pivot is one of the elements in Z_{ij}
- If the pivot is either z_i or z_j
  - z_i and z_j will be compared

Observations

For $1 \leq i \leq j \leq n$, let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

- remember $z_i < z_{i+1} < \cdots < z_j$

Observations:

- Partition divides an array into three segments, left, pivot, and Right.
- When z_i and z_j are first placed in DIFFERENT segments of the array by partitioning, the pivot is one of the elements in Z_{ij}
- If the pivot is either z_i or z_j
  - z_i and z_j will be compared
- If the pivot is any element in Z_{ij} other than z_i or z_j
  - z_i and z_j are not compared with each other in all randomized-partition calls

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \ is \ compared \ with \ z_j\}$$

$$=$$

How to Find Pr{z_i is compared with z_j}?

$\Pr\{z_i \textit{ is compared with } z_j\}$

$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$

$=$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \textit{ is compared with } z_j\}$$
$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$
$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$
$$+$$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \ is\ compared\ with\ z_j\}$$

$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$

$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} +$$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \; is \; compared \; with \; z_j\}$$

$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$

$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} =$$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \textit{ is compared with } z_j\}$$

$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$

$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

How to Find Pr{z_i is compared with z_j}?

$$
\begin{aligned}
&\Pr\{z_i \textit{ is compared with } z_j\} \\
&\quad = \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad = \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} \\
&\qquad + \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad = \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}
\end{aligned}
$$

$$
E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \textit{ is compared with } z_j\} =
$$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \text{ is compared with } z_j\}$$

$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$

$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared with } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$=$$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \text{ is compared with } z_j\}$$

$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$

$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared with } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

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

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i\ is\ compared\ with\ z_j\}$$
$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$
$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$
$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$
$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i\ is\ compared\ with\ z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$
$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \textit{ is compared with } z_j\}$$

$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$

$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \textit{ is compared with } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\log n)$$

Note: $\sum_{k=1}^{n} \frac{1}{k} \leq \log(n)$

How to Find Pr{z_i is compared with z_j}?

$$
\begin{aligned}
&\Pr\{z_i \textit{ is compared with } z_j\}\\
&\quad = \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}\\
&\quad = \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}\\
&\qquad + \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}\\
&\quad = \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}
\end{aligned}
$$

$$
E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \textit{ is compared with } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}
$$

$$
= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\log n) = O(n \log n)
$$

Note: $\sum_{k=1}^{n} \frac{1}{k} \leq \log(n)$

How to Find Pr{z_i is compared with z_j}?

$$\Pr\{z_i \textit{ is compared with } z_j\}$$
$$= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\}$$
$$= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\}$$
$$+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\}$$
$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \textit{ is compared with } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$
$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} O(\log n) = O(n \log n)$$

Note: $\sum_{k=1}^{n} \frac{1}{k} \le \log(n)$

Hence, the expected number of comparisons is $O(n \log n)$, which is the expected running time of Randomized-Quicksort

Selection Problem

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Review to Divide-and-Conquer Paradigm

- Divide-and-conquer (D&C) is an important algorithm design paradigm.
  - **Divide**
    Dividing a given problem into two or more subproblems (ideally of approximately equal size)
  - **Conquer**
    Solving each subproblem (directly if small enough or recursively)
  - **Combine**
    Combining the solutions of the subproblems into a global solution

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Linear Time Selection

Definition (Selection Problem)

Given a sequence of numbers $\langle a_1, \ldots, a_n \rangle$, and an integer i, $1 \leq i \leq n$, find the ith smallest element. When $i = \lceil n/2 \rceil$, it is called the median problem.

Linear Time Selection

Definition (Selection Problem)

Given a sequence of numbers $\langle a_1, \ldots, a_n \rangle$, and an integer i, $1 \leq i \leq n$, find the ith smallest element. When $i = \lceil n/2 \rceil$, it is called the median problem.

Example

Given $\langle 1, 8, 23, 10, 19, 33, 100 \rangle$, the 4th smallest element is 19.

Linear Time Selection

Definition (Selection Problem)

Given a sequence of numbers $\langle a_1, \ldots, a_n \rangle$, and an integer i, $1 \le i \le n$, find the ith smallest element. When $i = \lceil n/2 \rceil$, it is called the median problem.

Example

Given $\langle 1, 8, 23, 10, 19, 33, 100 \rangle$, the 4th smallest element is 19.

Question

How do you solve this problem?

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

First Solution: Selection by Sorting

- Sort the elements in ascending order with any algorithm of complexity O(n log n).

First Solution: Selection by Sorting

- Sort the elements in ascending order with any algorithm of complexity O(n log n).
- Return the *i*th element of the sorted array.

First Solution: Selection by Sorting

- Sort the elements in ascending order with any algorithm of complexity O(n log n).
- Return the *i*th element of the sorted array.

The complexity of this solution is O(   )

First Solution: Selection by Sorting

- Sort the elements in ascending order with any algorithm of complexity O(n log n).
- Return the *i*th element of the sorted array.

The complexity of this solution is O(n log n)

First Solution: Selection by Sorting

- Sort the elements in ascending order with any algorithm of complexity O(n log n).
- Return the *i*th element of the sorted array.

The complexity of this solution is O(n log n)

Question

Can we do better?

First Solution: Selection by Sorting

- Sort the elements in ascending order with any algorithm of complexity O(n log n).
- Return the *i*th element of the sorted array.

The complexity of this solution is O(n log n)

Question

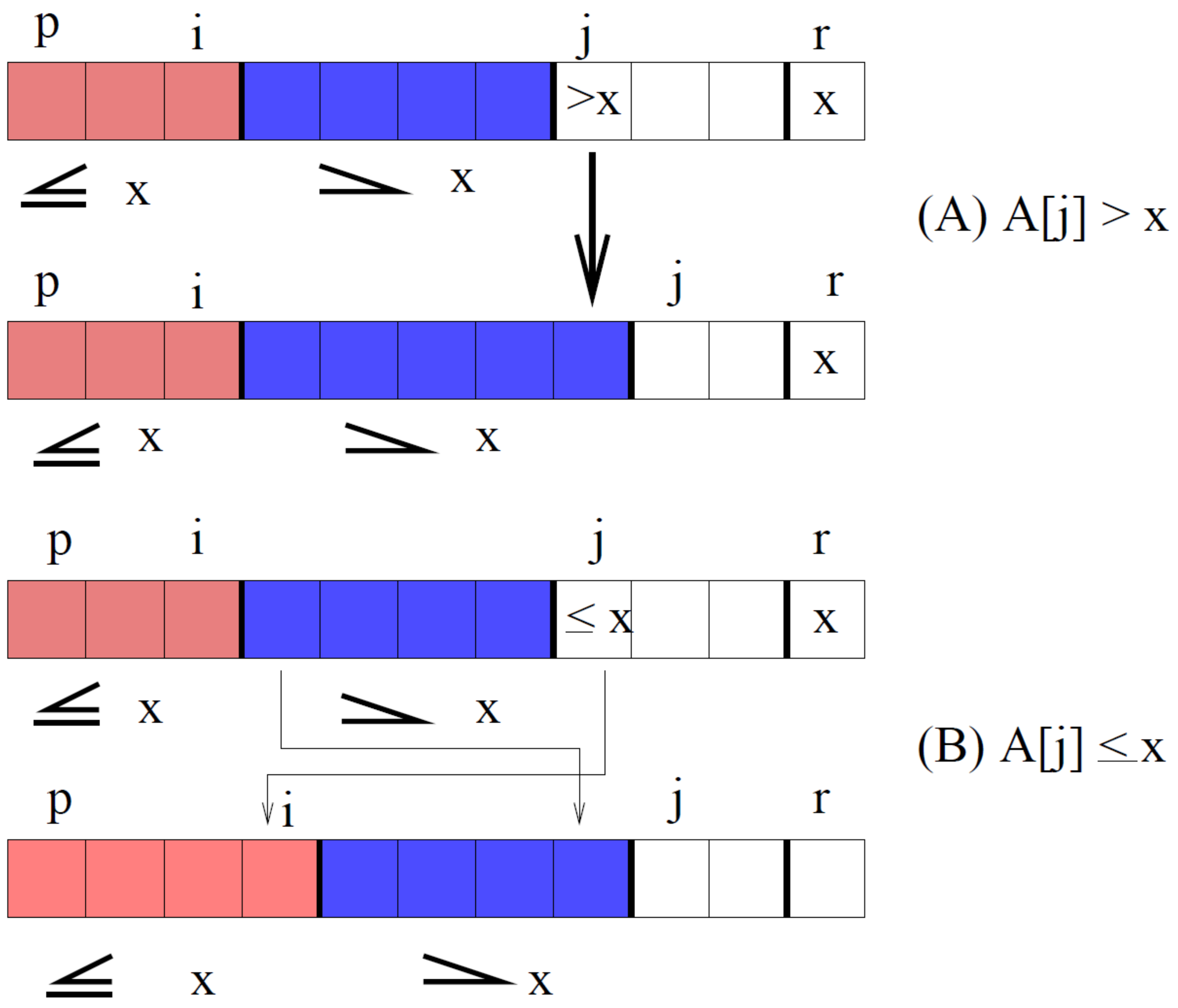
Can we do better?

Answer: YES, but we need to recall Partition(A,p,r) used in Quicksort!

Outline

- Review to Divide-and-Conquer Paradigm
- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Review of Randomized-Partition (A,p,r)



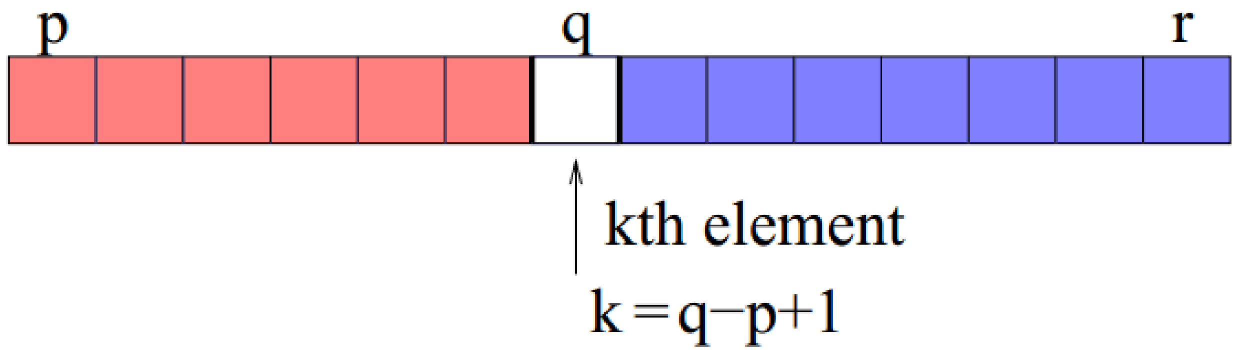
Randomized-Select(A,p,r,i), $1 \le i \le r-p+1$

Problem: Select the ith smallest element in A[p..r], where $1 \le i \le r-p+1$

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Problem: Select the *i*th smallest element in A[p..r], where 1 ≤ i ≤ r−p+1

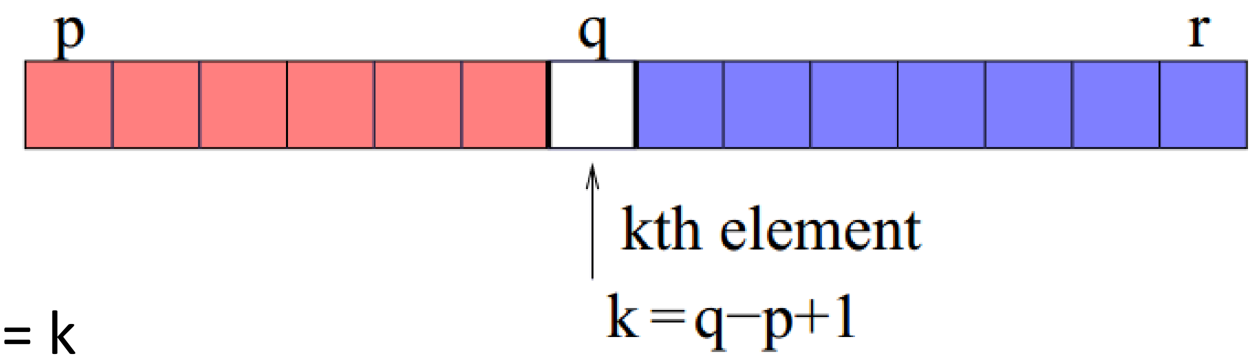
Solution: Apply Randomized-Partition(A, p, r), getting



Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Problem: Select the *i*th smallest element in A[p..r], where $1 \le i \le r-p+1$

Solution: Apply Randomized-Partition(A, p, r), getting

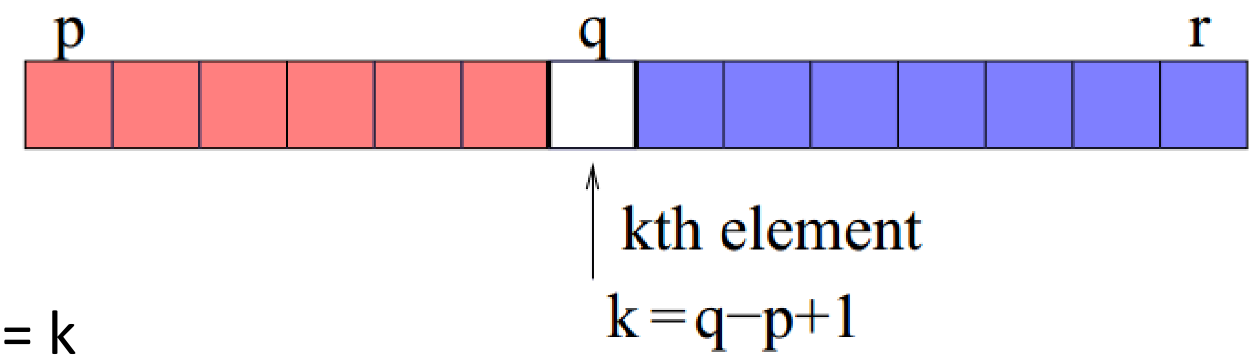


- i = k
  - pivot is the solution

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Problem: Select the *i*th smallest element in A[p..r], where $1 \le i \le r-p+1$

Solution: Apply Randomized-Partition(A, p, r), getting

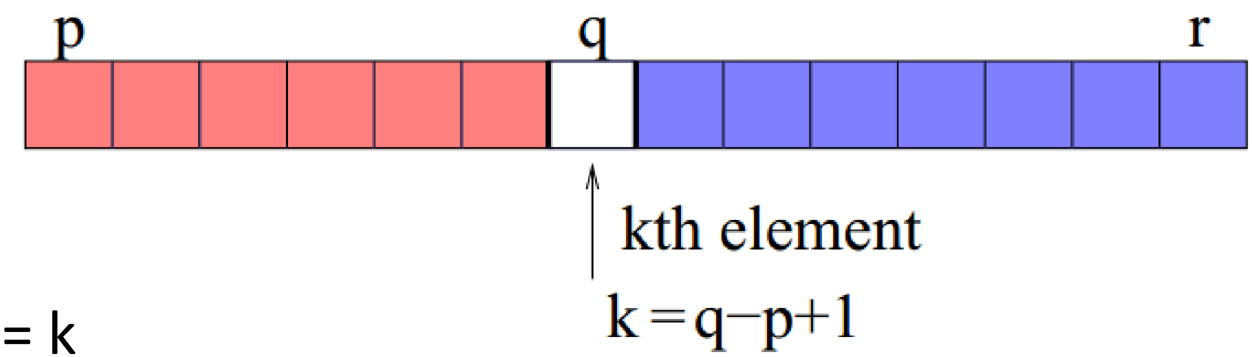


- i = k
  - pivot is the solution
- i < k
  - the *i*th smallest element in A[p..r] must be the *i*th smallest element in A[p..q−1]

Randomized-Select(A,p,r,i), $1 \le i \le r-p+1$

Problem: Select the *i*th smallest element in A[p..r], where $1 \le i \le r-p+1$

Solution: Apply Randomized-Partition(A, p, r), getting



- i = k
  - pivot is the solution
- i < k
  - the *i*th smallest element in A[p..r] must be the *i*th smallest element in A[p..q−1]
- i > k
  - the *i*th smallest element in A[p..r] must be the (*i* − *k*)th smallest element in A[q+1..r]

If necessary, recursively call the same procedure to the subarray

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

Input: An array $\boldsymbol{A}$,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$,the $\boldsymbol{i}$th smallest element that we want to select

Output: The ith smallest element $\boldsymbol{A}[\boldsymbol{i}]$

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

Input: An array $\boldsymbol{A}$,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$,the $\boldsymbol{i}$th smallest element that we want to select

Output: The ith smallest element $\boldsymbol{A}[\boldsymbol{i}]$

if p *is equal to* r **then**
  | **return** $A[p]$;
end

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

Input: An array $\boldsymbol{A}$,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$,the $\boldsymbol{i}$th smallest element that we want to select

Output: The ith smallest element $\boldsymbol{A}[\boldsymbol{i}]$

if p *is equal to* r **then**

  return $A[p]$;

end

$q \leftarrow$Randomized-Partition(A, p, r);

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

Input: An array $\boldsymbol{A}$,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$,the $\boldsymbol{i}$th smallest element that we want to select

Output: The ith smallest element $\boldsymbol{A}[\boldsymbol{i}]$

if p *is equal to* r **then**
  return $A[p]$;
end
$q \leftarrow$Randomized-Partition(A, p, r);
$k \leftarrow q - p + 1$;
if $i \leftarrow k$ **then**

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

```
Input: An array A,the range of index p,r,the ith smallest element that
       we want to select
Output: The ith smallest element A[i]
if p is equal to r then
 | return A[p];
end
q ←Randomized-Partition(A, p, r);
k ← q − p + 1;
if i ← k then
 | return A[q];//The pivot is the answer
end
```

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

Input: An array $\boldsymbol{A}$,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$,the $\boldsymbol{i}$th smallest element that we want to select
Output: The ith smallest element $\boldsymbol{A}[\boldsymbol{i}]$
if p *is equal to* r **then**
  | **return** $A[p]$;
end
$q \leftarrow$Randomized-Partition(A, p, r);
$k \leftarrow q - p + 1$;
if $i \leftarrow k$ **then**
  | **return** $A[q]$;//The pivot is the answer
end
else if $i < k$ **then**
  | **return** Randomized-Select$(A, p, q - 1, i)$;
end

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

```
Input: An array A,the range of index p,r,the ith smallest element that
       we want to select
Output: The ith smallest element A[i]
if p is equal to r then
 |  return A[p];
end
q ←Randomized-Partition(A, p, r);
k ← q − p + 1;
if i ← k then
 |  return A[q];//The pivot is the answer
end
else if i < k then
 |  return Randomized-Select(A, p, q − 1, i);
end
else
 |  return Randomized-Select(A, q + 1, r, i − k);
end
```

Randomized-Select(A,p,r,i), 1≤i≤r−p+1

Randomized-Select(A,p,r,i)

Input: An array $\boldsymbol{A}$,the range of index $\boldsymbol{p}$,$\boldsymbol{r}$,the $\boldsymbol{i}$th smallest element that we want to select

Output: The ith smallest element $\boldsymbol{A}[\boldsymbol{i}]$

if p *is equal to* r **then**
  | **return** $A[p]$;
end
$q \leftarrow$Randomized-Partition(A, p, r);
$k \leftarrow q - p + 1$;
if $i \leftarrow k$ **then**
  | **return** $A[q]$;//The pivot is the answer
end
else if $i < k$ **then**
  | **return** Randomized-Select$(A, p, q - 1, i)$;
end
else
  | **return** Randomized-Select$(A, q + 1, r, i - k)$;
end

To find the ith smallest element in A[1..n], call Randomized-Select(A, 1, n, i)

Randomized Selection - Example

- Find the 8th smallest element of the following list of numbers:
  - 8 25 2 14 3 20 15 13 12 11 7 5 9 10 16 18 1 23 26 21

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i-k*)th smallest element in A[q+1..r]

p																			r
8	25	2	14	3	20	15	13	12	11	7	5	9	10	16	18	1	23	26	21

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i-k*)th smallest element in A[q+1..r]

p																			r
8	25	2	14	3	20	15	13	12	11	7	5	9	10	16	18	1	23	26	21

$$i = 8, p = 1, r = 20$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

p																			r
8	25	2	14	3	20	15	13	12	11	7	5	9	10	16	18	1	23	26	21

$$i = 8, p = 1, r = 20$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i-k*)th smallest element in A[q+1..r]

p															q				r
8	2	14	3	15	13	12	11	7	5	9	10	16	18	1	20	25	23	26	21

$$i = 8, p = 1, r = 20$$
$$q = 16, \mathrm{k} = 16$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i-k*)th smallest element in A[q+1..r]

p															q				r
8	2	14	3	15	13	12	11	7	5	9	10	16	18	1	20	25	23	26	21

$$i = 8, p = 1, r = 20$$
$$q = 16, \mathrm{k} = 16$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

p														r					
8	2	14	3	15	13	12	11	7	5	9	10	16	18	1	20	25	23	26	21

$$i = 8, p = 1, r = 15$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

p														**r**					
8	**2**	**14**	**3**	**15**	**13**	**12**	**11**	**7**	**5**	**9**	**10**	**16**	**18**	**1**	20	25	23	26	21

$$i = 8, p = 1, r = 15$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

p						q								r					
8	2	3	7	5	1	9	11	14	15	13	10	16	18	12	20	25	23	26	21

$$i = 8, p = 1, r = 15$$
$$q = 7, k = 7$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

p						q								r					
8	2	3	7	5	1	9	11	14	15	13	10	16	18	12	20	25	23	26	21

$$i = 8, p = 1, r = 15$$
$$q = 7, k = 7$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i-k*)th smallest element in A[q+1..r]

							p							r					
8	2	3	7	5	1	9	11	14	15	13	10	16	18	12	20	25	23	26	21

$$i = 1, p = 8, r = 15$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

							p							r					
8	2	3	7	5	1	9	**11**	**14**	**15**	**13**	**10**	**16**	**18**	**12**	20	25	23	26	21

$$i = 1, p = 8, r = 15$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

							p			q				r					
8	2	3	7	5	1	9	**11**	**12**	**10**	**13**	**15**	**14**	**18**	**16**	20	25	23	26	21

$$i = 1, p = 8, r = 15$$
$$q = 11, k = 4$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

							p			**q**				**r**					
8	2	3	7	5	1	9	**11**	**12**	**10**	**13**	**15**	**14**	**18**	**16**	20	25	23	26	21

$$i = 1, p = 8, r = 15$$
$$q = 11, k = 4$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i-k*)th smallest element in A[q+1..r]

							p		r										
8	2	3	7	5	1	9	**11**	**12**	**10**	13	15	14	18	16	20	25	23	26	21

$$i = 1, p = 8, r = 10$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

							p		r										
8	2	3	7	5	1	9	**11**	**12**	**10**	13	15	14	18	16	20	25	23	26	21

$$i = 1, p = 8, r = 10$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

							p,q		r										
8	2	3	7	5	1	9	10	**12**	**11**	13	15	14	18	16	20	25	23	26	21

$$i = 1, p = 8, r = 10$$
$$q = 8, k = 1$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

							p,q		r										
8	2	3	7	5	1	9	10	12	11	13	15	14	18	16	20	25	23	26	21

$$i = 1, p = 8, r = 10$$
$$q = 8, k = 1$$

Randomized Selection - Example

- Select the *i*th smallest element in A[p..r], pivot is A[q], k = q-p+1.
  - i = k : pivot is the solution
  - i < k : the *i*th smallest element in A[p..q-1]
  - i > k : the (*i*-*k*)th smallest element in A[q+1..r]

							p,q		r										
8	2	3	7	5	1	9	10	12	11	13	15	14	18	16	20	25	23	26	21

$$i = 1, p = 8, r = 10$$
$$q = 8, k = 1$$

10 is the 8th smallest element of the array.

Outline

- Review to Divide-and-Conquer Paradigm

- Selection Problem
  - Problem Definition
  - First solution: Selection by sorting
  - A randomized divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.
- Therefore, for each k such that $1 \leq k \leq n$, the subarray $A[p..q]$ has k elements (all less than or equal to the pivot) with probability .

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.
- Therefore, for each k such that $1 \le k \le n$, the subarray $A[p..q]$ has k elements (all less than or equal to the pivot) with probability 1/n.

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.
- Therefore, for each k such that $1 \leq k \leq n$, the subarray $A[p..q]$ has k elements (all less than or equal to the pivot) with probability 1/n.
- For k = 1,2,…,n, we define indicator random variables X_k where

$$X_k = I\{the\ subarray\ A[p..q] has\ exactly\ k\ elements\}$$

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.
- Therefore, for each k such that $1 \leq k \leq n$, the subarray $A[p..q]$ has k elements (all less than or equal to the pivot) with probability 1/n.
- For k = 1,2,…,n, we define indicator random variables X_k where

$$X_k = I\{the\ subarray\ A[p..q] has\ exactly\ k\ elements\}$$

we have

Running Time of Randomized-Select(A,1,n,i)

- We let the running time on an input array A[p..r] of n elements be a random variable denoted by $T(n)$.
- The procedure RANDOMIZED-PARTITION is equally likely to return any element as the pivot.
- Therefore, for each k such that $1 \leq k \leq n$, the subarray $A[p..q]$ has k elements (all less than or equal to the pivot) with probability 1/n.
- For k = 1,2,…,n, we define indicator random variables X_k where

$$X_k = I\{the\ subarray\ A[p..q] has\ exactly\ k\ elements\}$$

we have

$$E[X_k] = 1/n$$

Running Time of Randomized-Select(A,1,n,i)

- When we choose $A[q]$ as the pivot element, we will either

Running Time of Randomized-Select(A,1,n,i)

- When we choose $A[q]$ as the pivot element, we will either
  - terminate immediately with the correct answer

Running Time of Randomized-Select(A,1,n,i)

- When we choose $A[q]$ as the pivot element, we will either
  - terminate immediately with the correct answer
  - recurse on the subarray $A[p..q-1]$
  - recurse on the subarray $A[q+1..r]$

Running Time of Randomized-Select(A,1,n,i)

- When we choose $A[q]$ as the pivot element, we will either
  - terminate immediately with the correct answer
  - recurse on the subarray $A[p..q-1]$
  - recurse on the subarray $A[q+1..r]$
- To obtain an upper bound, we assume that the *i*th element is always on the side of the partition with the greater number of elements.

Running Time of Randomized-Select(A,1,n,i)

- When we choose $A[q]$ as the pivot element, we will either
  - terminate immediately with the correct answer
  - recurse on the subarray $A[p..q-1]$
  - recurse on the subarray $A[q+1..r]$
- To obtain an upper bound, we assume that the *i*th element is always on the side of the partition with the greater number of elements.
- When $X_k = 1$, the two subarrays on which we might recurse have sizes $k-1$ and $n-k$.Hence,

Running Time of Randomized-Select(A,1,n,i)

- When we choose $A[q]$ as the pivot element, we will either
  - terminate immediately with the correct answer
  - recurse on the subarray $A[p..q-1]$
  - recurse on the subarray $A[q+1..r]$
- To obtain an upper bound, we assume that the *i*th element is always on the side of the partition with the greater number of elements.
- When $X_k = 1$, the two subarrays on which we might recurse have sizes $k-1$ and $n-k$.Hence,

$$T(n) \leq \sum_{k=1}^{n} X_k \cdot \left(T(\max(k-1, n-k)) + O(n)\right)$$

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$E[T(n)] \le E\left[\sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)\right]$$

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$E[T(n)] \leq E\left[\sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)\right]$$

$$= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n)$$

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$E[T(n)] \leq E\left[\sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)\right]$$

$$= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n)$$

Linearity of expectation

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$
\begin{aligned}
E[T(n)] &\leq E\left[\sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)\right] \\
&= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n) \\
&= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n)
\end{aligned}
$$

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$E[T(n)] \le E\left[\sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)\right]$$

$$= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n)$$

$$= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n)$$

Independence between X_k and $T(max(k-1, n-k))$

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$E[T(n)] \leq E\left[\sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)\right]$$

$$= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n)$$

$$= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n)$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot E[T(\max(k-1, n-k))] + O(n)$$

Running Time of Randomized-Select(A,1,n,i)

Taking expected values, we have

$$E[T(n)] \le E\left[\sum_{k=1}^{n} X_k \cdot \big(T(\max(k-1, n-k)) + O(n)\big)\right]$$

$$= \sum_{k=1}^{n} E[X_k \cdot T(\max(k-1, n-k))] + O(n)$$

$$= \sum_{k=1}^{n} E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n)$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot E[T(\max(k-1, n-k))] + O(n)$$

$\boldsymbol{E[X_k] = 1/n}$

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 \;\; if \; k > \lceil n/2 \rceil \\ n-k \; if \; k \leq \lceil n/2 \rceil \end{cases}$$

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 & if\ k > \lceil n/2 \rceil \\ n-k & if\ k \leq \lceil n/2 \rceil \end{cases}$$

- Assuming that $T(n)$ is monotonically increasing

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 \;\; if \;\; k > \lceil n/2 \rceil \\ n-k \; if \;\; k \leq \lceil n/2 \rceil \end{cases}$$

- Assuming that $T(n)$ is monotonically increasing
  - If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 \ \ if \ k > \lceil n/2 \rceil \\ n-k \ if \ k \le \lceil n/2 \rceil \end{cases}$$

- Assuming that $T(n)$ is monotonically increasing
  - If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation
  - if n is odd, all these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once

Thus we have

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 \ \ if\ k > \lceil n/2 \rceil \\ n-k\ if\ k \leq \lceil n/2 \rceil \end{cases}$$

- Assuming that $T(n)$ is monotonically increasing
  - If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation
  - if n is odd, all these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once

Thus we have

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n)$$

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 \ \ if\ k > \lceil n/2 \rceil \\ n-k\ if\ k \leq \lceil n/2 \rceil \end{cases}$$

- Assuming that $T(n)$ is monotonically increasing
  - If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation
  - if n is odd, all these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once

  Thus we have

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n)$$

- It is a complicated recurrence!
- Use

Running Time of Randomized-Select(A,1,n,i)

- Consider the expression $\max(k-1, n-k)$,

$$\max(k-1, n-k) = \begin{cases} k-1 \ \ if\ k > \lceil n/2 \rceil \\ n-k\ if\ k \le \lceil n/2 \rceil \end{cases}$$

- Assuming that $T(n)$ is monotonically increasing
  - If n is even, each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation
  - if n is odd, all these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once

  Thus we have

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n)$$

- It is a complicated recurrence!
- Use Guess and Induction (Substitution Method)

Running Time of Randomized-Select(A,1,n,i)

- Guess:

$$T(n) \leq c\,n, \qquad for\ all\ n$$

  for some constant *c* to be figured out later.

Running Time of Randomized-Select(A,1,n,i)

- Guess:

$$T(n) \leq c\, n, \qquad for\ all\ n$$

  for some constant c to be figured out later.

Induction step: Assume that $T(m) \leq c\ m$ for all $m \leq n-1$.
Then try to show $T(n) \leq cn$:

Running Time of Randomized-Select(A,1,n,i)

- Guess:

$$T(n) \leq c\,n, \qquad for\ all\ n$$

for some constant c to be figured out later.

Induction step: Assume that $T(m) \leq c\,m$ for all $m \leq n-1$.
Then try to show $T(n) \leq cn$:

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n \rfloor}^{n-1} ck + an$$

Running Time of Randomized-Select(A,1,n,i)

- Guess:

$$T(n) \leq c\,n, \qquad for\ all\ n$$

  for some constant *c* to be figured out later.

Induction step: Assume that $T(m) \leq c\,m$ for all $m \leq n - 1$.
Then try to show $T(n) \leq cn$:

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n \rfloor}^{n-1} ck + an$$

$$= \frac{2c}{n} \Big(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\left\lfloor \frac{n}{2} \right\rfloor - 1} k \Big) + an$$

Running Time of Randomized-Select(A,1,n,i)

- Guess:

$$T(n) \le c\, n, \qquad for\ all\ n$$

for some constant *c* to be figured out later.

Induction step: Assume that $T(m) \le c\, m$ for all $m \le n - 1$.
Then try to show $T(n) \le cn$:

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n \rfloor}^{n-1} ck + an$$

$$= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\left\lfloor \frac{n}{2} \right\rfloor - 1} k \right) + an$$

$$= \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an$$

Running Time of Randomized-Select(A,1,n,i)

$$= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2}\right) + an$$

Running Time of Randomized-Select(A,1,n,i)

$$= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2}\right) + an$$

$$\leq \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an$$

Running Time of Randomized-Select(A,1,n,i)

$$= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2}\right) + an$$

$$\leq \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an$$

$$= \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2}\right) + an$$

$$= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an$$

$$= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an$$

Running Time of Randomized-Select(A,1,n,i)

$$
\begin{aligned}
&= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2}\right) + an \\
&\le \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an \\
&= \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2}\right) + an \\
&= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an \\
&= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \\
&\le \frac{3cn}{4} + \frac{c}{2} + an \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)
\end{aligned}
$$

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or equivalently,

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or equivalently, that $\frac{cn}{4} - \frac{c}{2} - an \geq 0$.
- If $\frac{c}{4} - a > 0$, we have

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or, equivalently, that $\frac{cn}{4} - \frac{c}{2} - an \geq 0$.
- If $\frac{c}{4} - a > 0$, we have

$$n \geq \frac{2c}{c - 4a}$$

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or, equivalently, that $\frac{cn}{4} - \frac{c}{2} - an \geq 0$.
- If $\frac{c}{4} - a > 0$, we have

$$n \geq \frac{2c}{c - 4a}$$

If we choose $c \geq 12a$, then the induction step works for $n \geq 3$.

Induction basis:

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or, equivalently, that $\frac{cn}{4} - \frac{c}{2} - an \geq 0$.
- If $\frac{c}{4} - a > 0$, we have

$$n \geq \frac{2c}{c - 4a}$$

If we choose $c \geq 12a$, then the induction step works for $n \geq 3$.

Induction basis:

$$T(1) \leq c \cdot 1, T(2) \leq c \cdot 2$$

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or, equivalently, that $\frac{cn}{4} - \frac{c}{2} - an \geq 0$.
- If $\frac{c}{4} - a > 0$, we have

$$n \geq \frac{2c}{c - 4a}$$

If we choose $c \geq 12a$, then the induction step works for $n \geq 3$.

Induction basis:

$$T(1) \leq c \cdot 1, T(2) \leq c \cdot 2$$

So if we choose then the entire proof works.

Running Time of Randomized-Select(A,1,n,i)

- In order to complete the proof, we need to show that for sufficiently large n, this last expression is at most $c \cdot n$ or, equivalently, that $\frac{cn}{4} - \frac{c}{2} - an \geq 0$.
- If $\frac{c}{4} - a > 0$, we have

$$n \geq \frac{2c}{c - 4a}$$

If we choose $c \geq 12a$, then the induction step works for $n \geq 3$.

Induction basis:

$$T(1) \leq c \cdot 1, T(2) \leq c \cdot 2$$

So if we choose $c = \max\{12a, T(1), T(2)/2\}$, then the entire proof works.

Running Time of Randomized-Select(A,1,n,i)

Worst Case:

$$T(n) = n - 1 + T(n-1), T(n) = O(n^2)$$

Running Time of Randomized-Select(A,1,n,i)

Worst Case:

$$T(n) = n - 1 + T(n-1), T(n) = O(n^2)$$

Expected running time much better than worst case!

Randomized Quicksort vs Randomized Selection

Question

Why does Randomized Selection take $O(n)$ time while Randomized Quicksort takes $O(n \log n)$ time?

Randomized Quicksort vs Randomized Selection

Question

Why does Randomized Selection take $O(n)$ time while Randomized Quicksort takes $O(n \log n)$ time?

Answer:

- Randomized Selection needs to work on only 1 of the two subproblems.

Randomized Quicksort vs Randomized Selection

Question

Why does Randomized Selection take $O(n)$ time while Randomized Quicksort takes $O(n \log n)$ time?

Answer:

- Randomized Selection needs to work on only 1 of the two subproblems.
- Randomize Quicksort needs to work on both of the two subproblems.

Summary of Divide-and-Conquer and Randomization

- Randomization is needed when it is not easy to divide evenly.
- Use expected case analysis for randomized algorithms.
- The running time is the expected running time for any given input, expectation is with respect to the random choices made by the algorithm internally.

End of Section