Divide-and-Conquer: Searching in an Array

Neil Rhodes

Department of Computer Science and Engineering
University of California, San Diego

Data Structures and Algorithms
Algorithmic Toolbox

Outline

1. Main Idea of Divide-and-Conquer
2. Linear Search
3. Binary Search

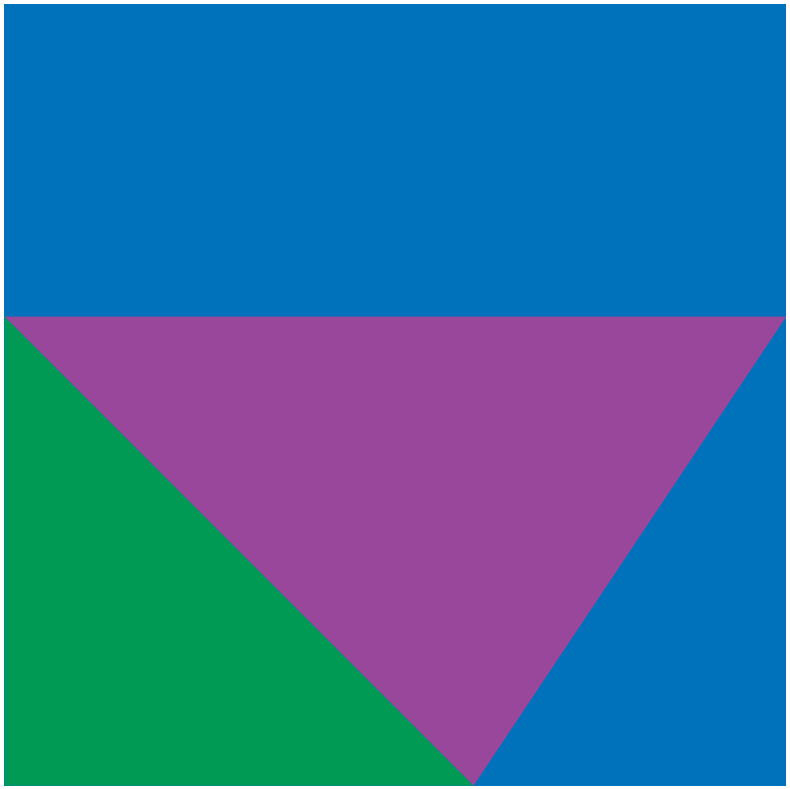
a problem to be solved

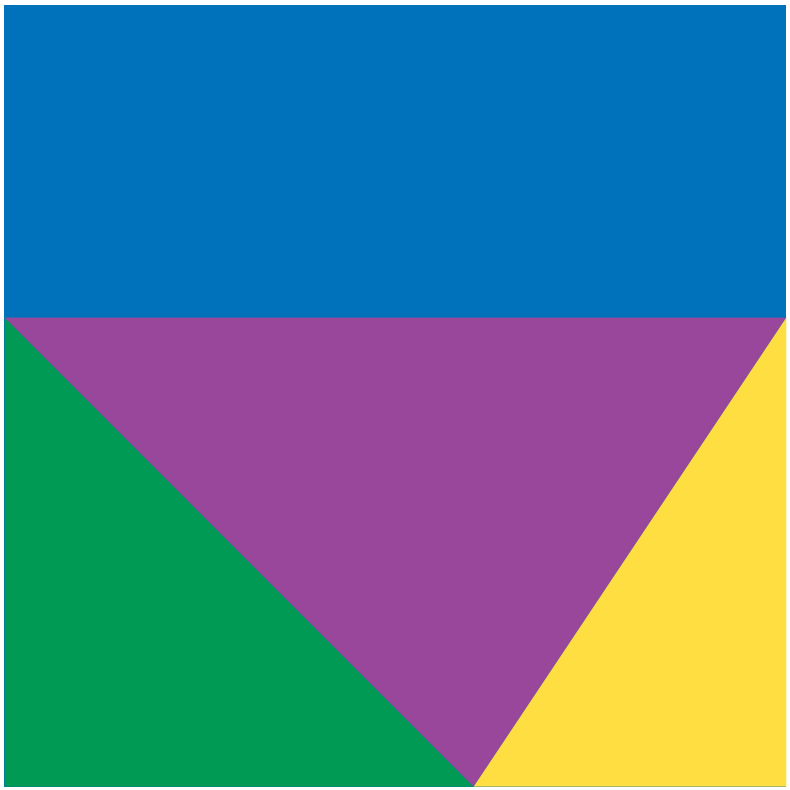
Divide: Break into non-overlapping subproblems of the same type

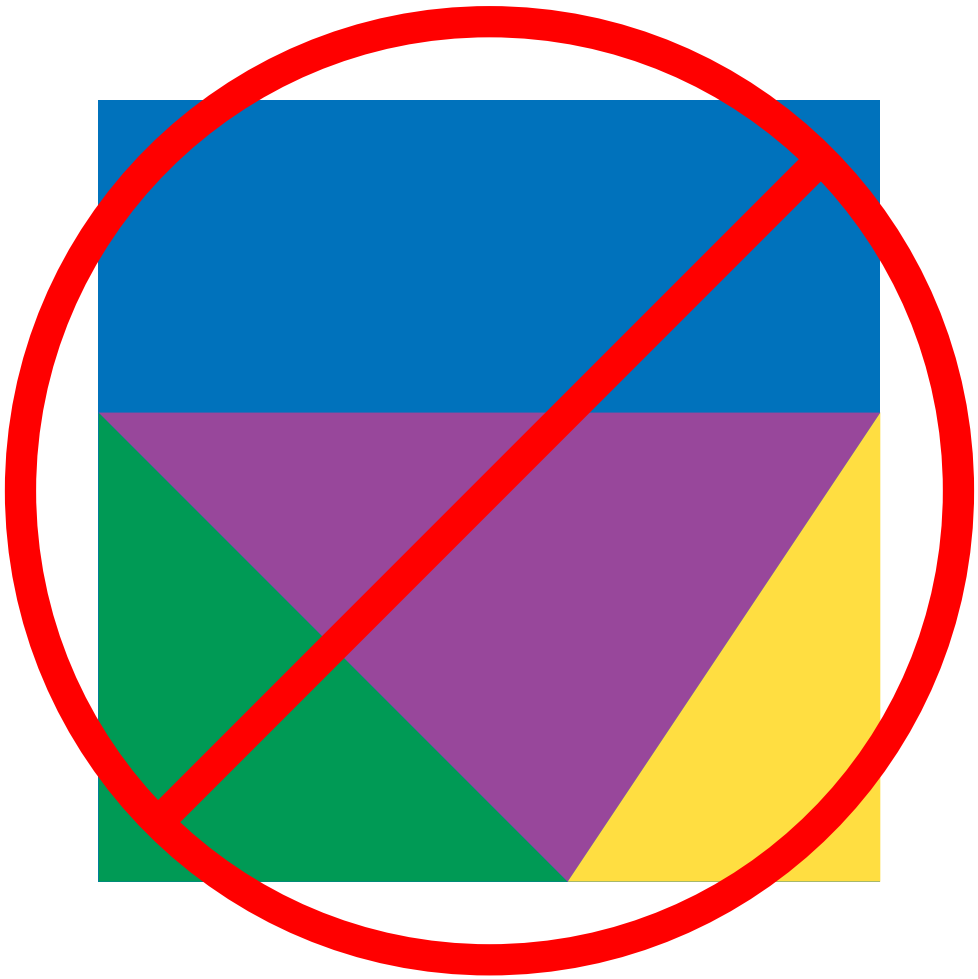
Divide: Break into non-overlapping subproblems of the same type

Divide: Break into non-overlapping subproblems of the same type

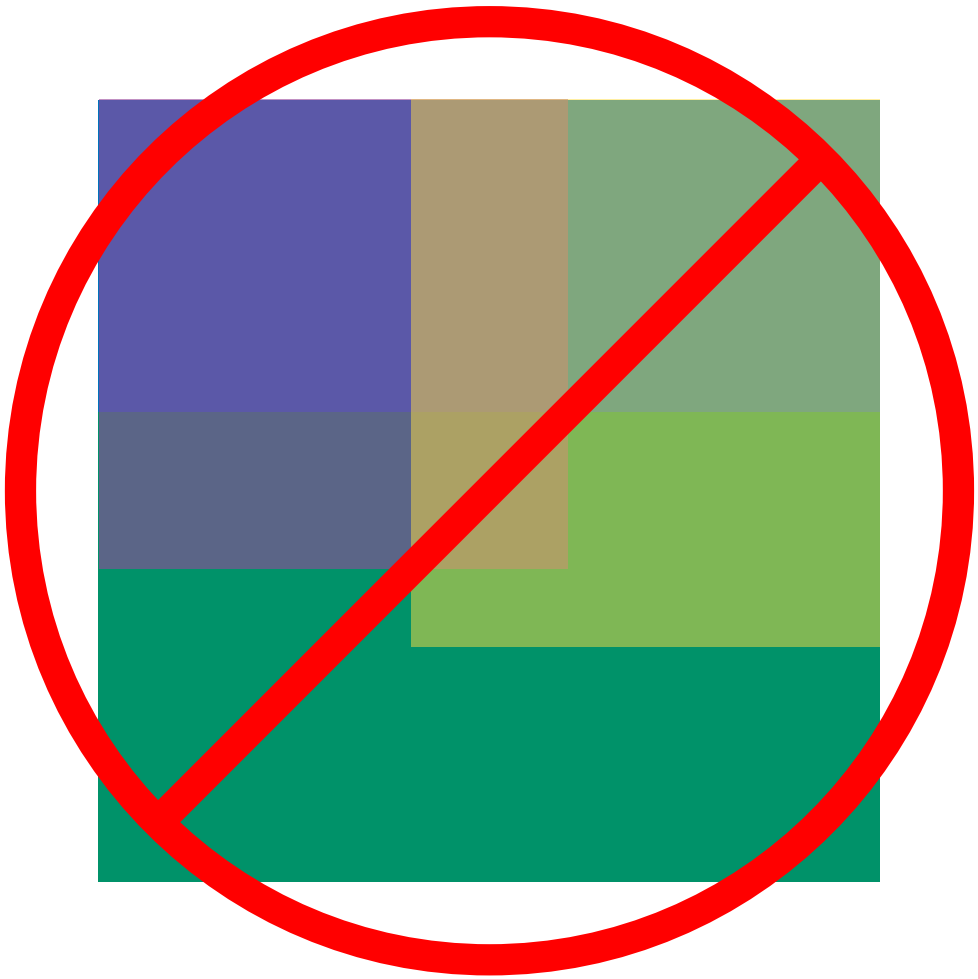
Divide: Break into non-overlapping subproblems of the same type







not the
same type



overlapping

Divide: break apart

Conquer: solve subproblems

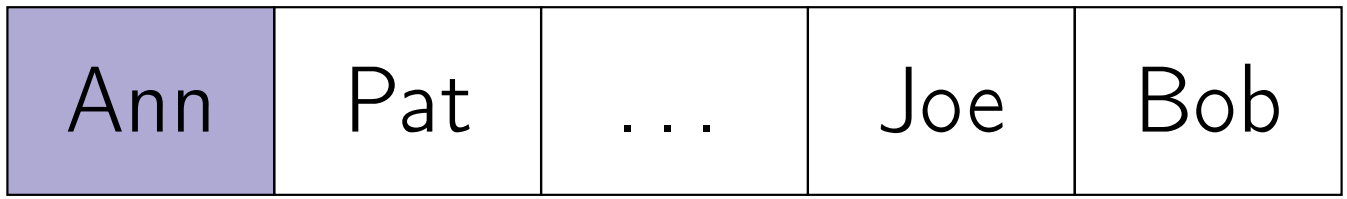
Conquer: combine

1. Break into non-overlapping subproblems of the same type
2. Solve subproblems
3. Combine results

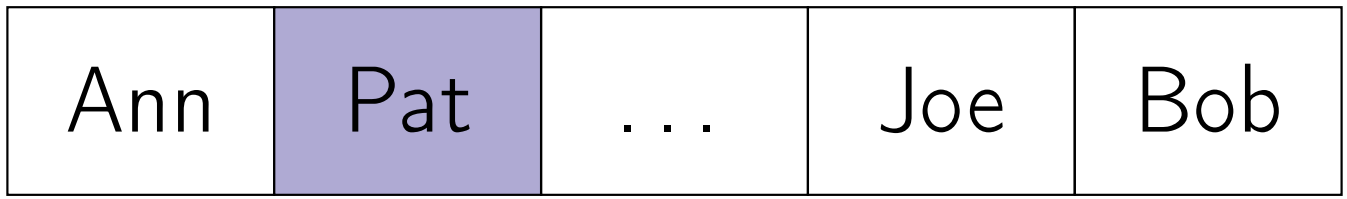
Outline

1. Main Idea of Divide-and-Conquer
2. Linear Search
3. Binary Search

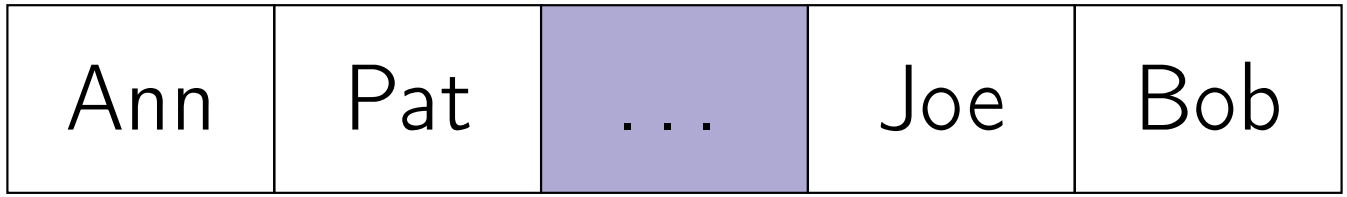
Linear Search in Array



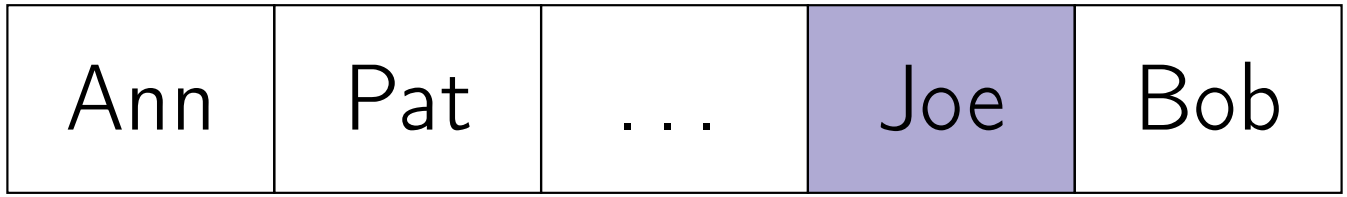
Linear Search in Array



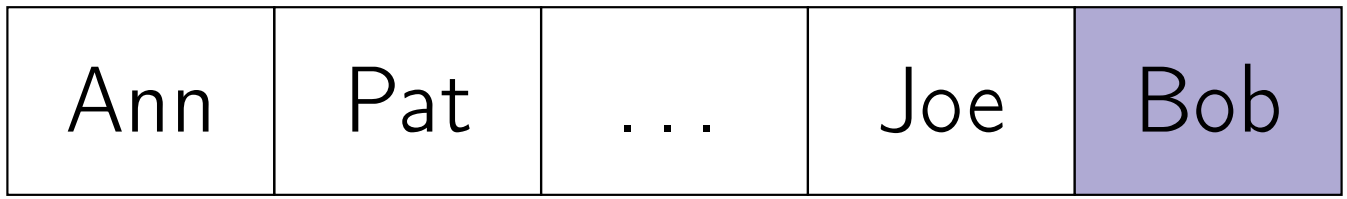
Linear Search in Array



Linear Search in Array



Linear Search in Array



Real-life Example

english	**french**	**italian**	**german**	**spanish**
house	maison	casa	Haus	casa
car	voiture	auto	Auto	auto
table	table	tavola	Tabelle	mesa

Searching in an array

Input: An array A with n elements.
A key k.

Output: An index, i, where $A[i] = k$.
If there is no such i, then
NOT_FOUND.

Recursive Solution

```
LinearSearch(A, low, high, key)
if high < low:
    return NOT_FOUND
if A[low] = key:
    return low
return LinearSearch(A, low + 1, high, key)
```

Definition

A recurrence relation is an equation recursively defining a sequence of values.

Fibonacci recurrence relation

$$F(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

$$0, 1, 1, 2, 3, 5, 8, \ldots$$

LinearSearch(A, low, $high$, key)

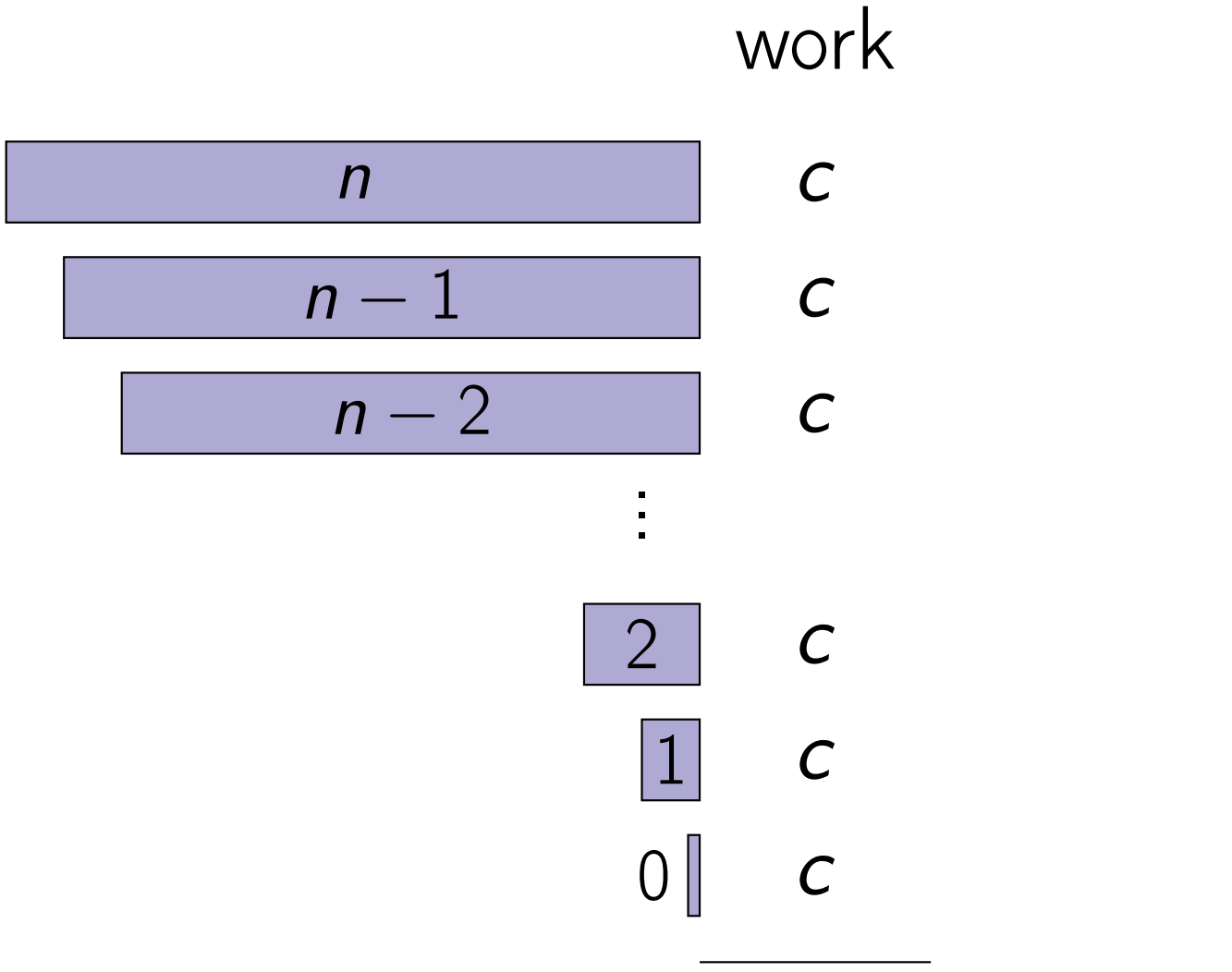
```
if high < low:
  return NOT_FOUND
if A[low] = key:
  return low
return LinearSearch(A, low + 1, high, key)
```

Recurrence defining worst-case time:

$$T(n) = T(n-1) + c$$

$$T(0) = c$$

Runtime of Linear Search



Iterative Version

```
LinearSearchIt(A, low, high, key)
for i from low to high:
    if A[i] = key:
        return i
return NOT_FOUND
```

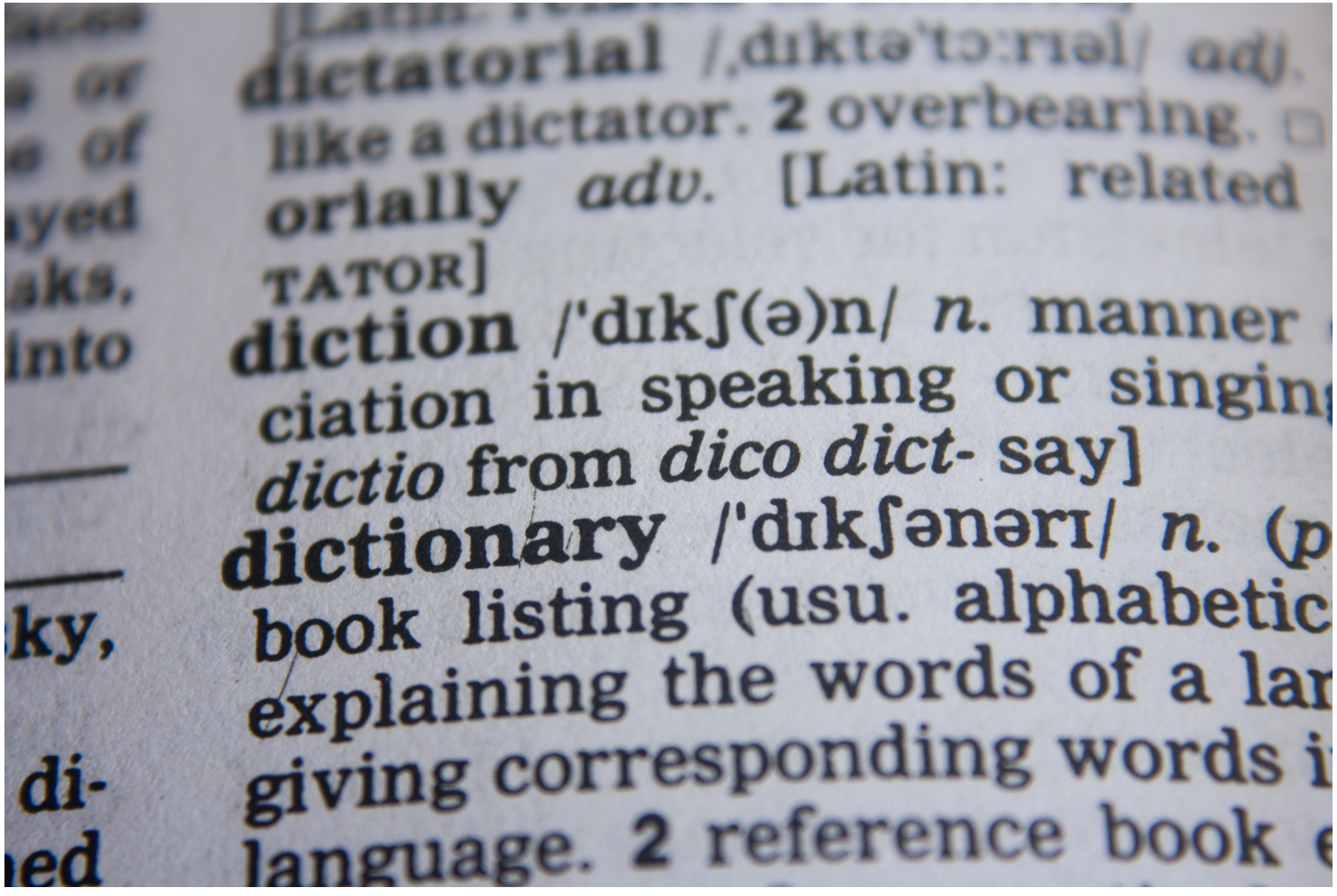
Summary

- Create a recursive solution
- Define a corresponding recurrence relation, T
- Determine $T(n)$: worst-case runtime
- Optionally, create iterative solution

Outline

1. Main Idea of Divide-and-Conquer
2. Linear Search
3. Binary Search

Searching Sorted Data



Searching in a sorted array

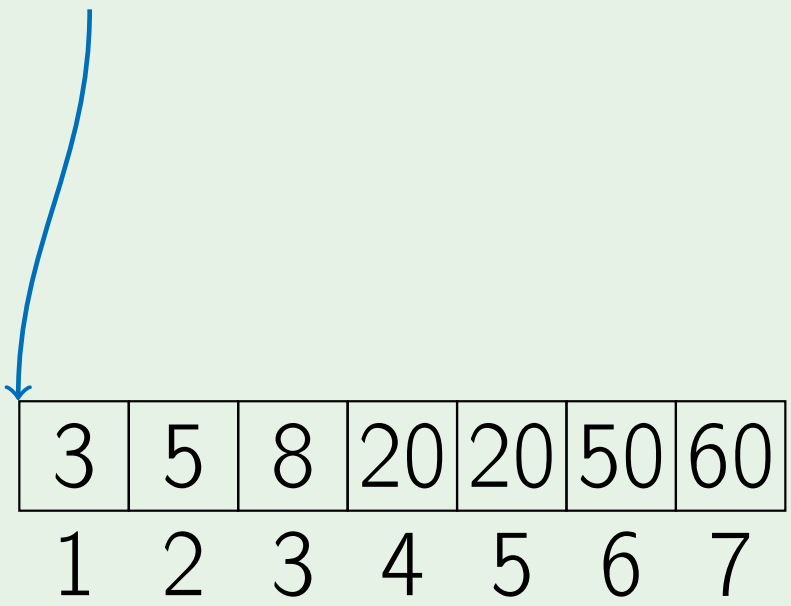
Input: A sorted array $A[low \ldots high]$
$(\forall low \leq i < high : A[i] \leq A[i+1])$.
A key k.

Output: An index, i, $(low \leq i \leq high)$ where $A[i] = k$.
Otherwise, the greatest index i, where $A[i] < k$.
Otherwise ($k < A[low]$), the result is $low - 1$.

Searching in a Sorted Array

Example

$search(2) \rightarrow 0$

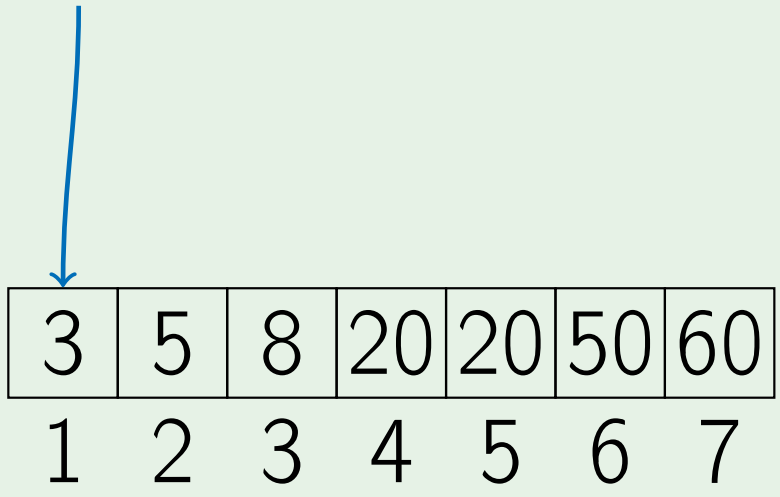


Searching in a Sorted Array

Example

$search(2) \rightarrow 0$

$search(3) \rightarrow 1$



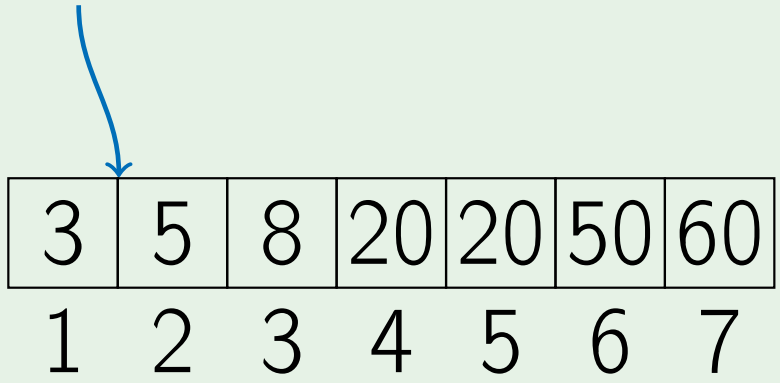
Searching in a Sorted Array

Example

$search(2) \rightarrow 0$

$search(3) \rightarrow 1$

$search(4) \rightarrow 1$



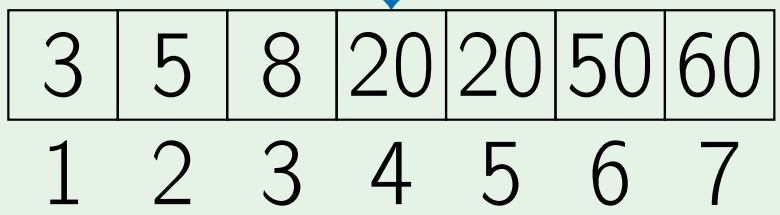
Searching in a Sorted Array

Example

$search(2) \rightarrow 0$ $search(20) \rightarrow 4$

$search(3) \rightarrow 1$

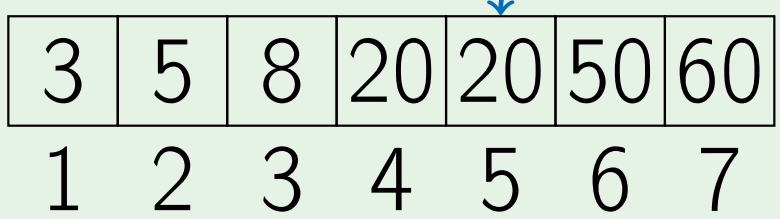
$search(4) \rightarrow 1$



Searching in a Sorted Array

Example

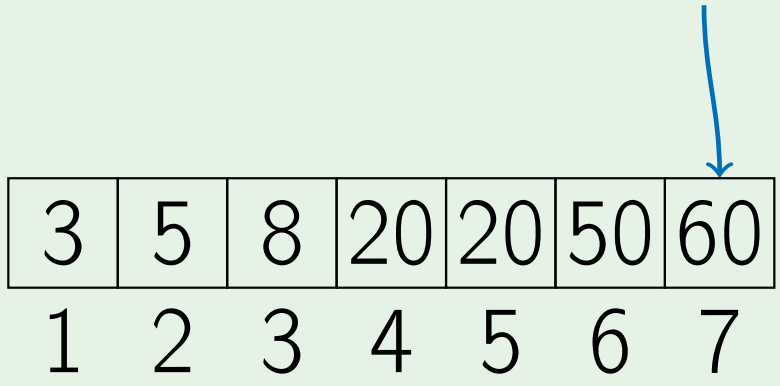
$search(2) \rightarrow 0$ $search(20) \rightarrow 4$
$search(3) \rightarrow 1$ $search(20) \rightarrow 5$
$search(4) \rightarrow 1$



Searching in a Sorted Array

Example

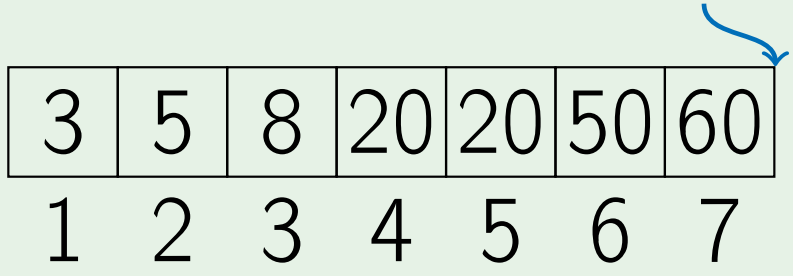
$$\textit{search}(2) \to 0 \quad \textit{search}(20) \to 4$$
$$\textit{search}(3) \to 1 \quad \textit{search}(20) \to 5$$
$$\textit{search}(4) \to 1 \quad \textit{search}(60) \to 7$$



Searching in a Sorted Array

Example

$search(2) \rightarrow 0$ $search(20) \rightarrow 4$
$search(3) \rightarrow 1$ $search(20) \rightarrow 5$
$search(4) \rightarrow 1$ $search(60) \rightarrow 7$
$search(90) \rightarrow 7$

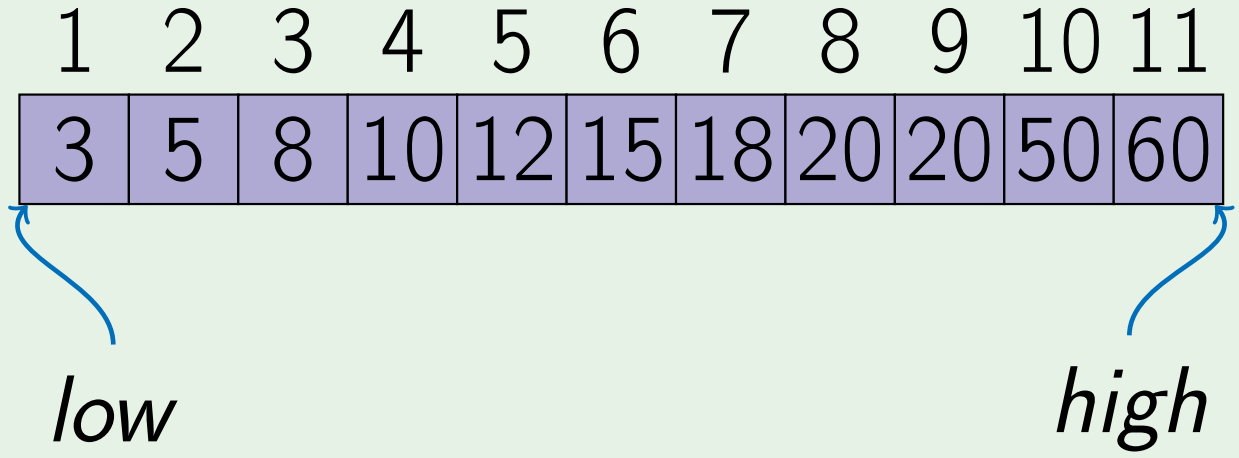


BinarySearch(A, low, $high$, key)

```
if high < low:
    return low − 1
mid ← ⌊low + (high − low)/2⌋
if key = A[mid]:
    return mid
else if key < A[mid]:
    return BinarySearch(A, low, mid − 1, key)
else:
    return BinarySearch(A, mid + 1, high, key)
```

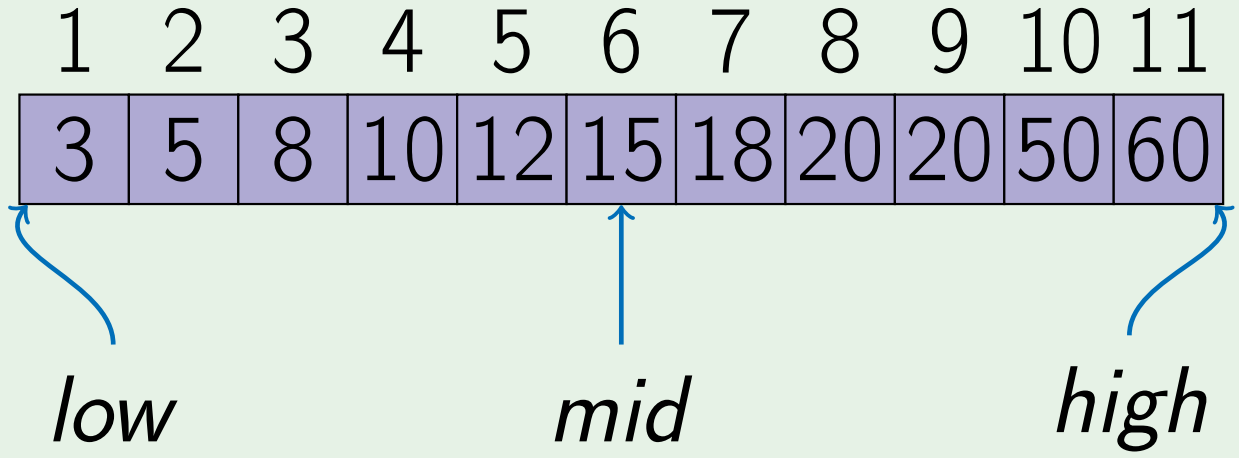
Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$



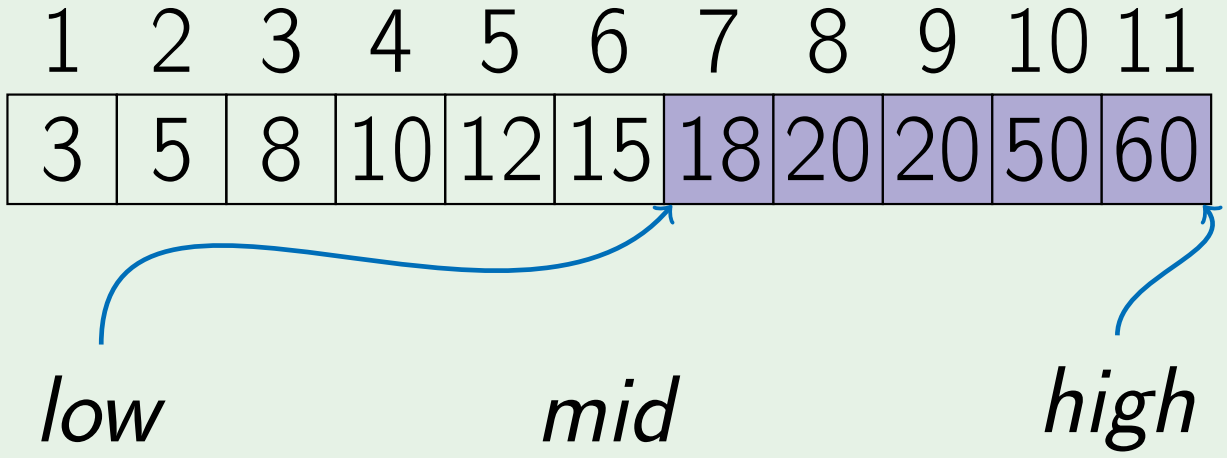
Example: Searching for the key 50

`BinarySearch`$(A, 1, 11, 50)$



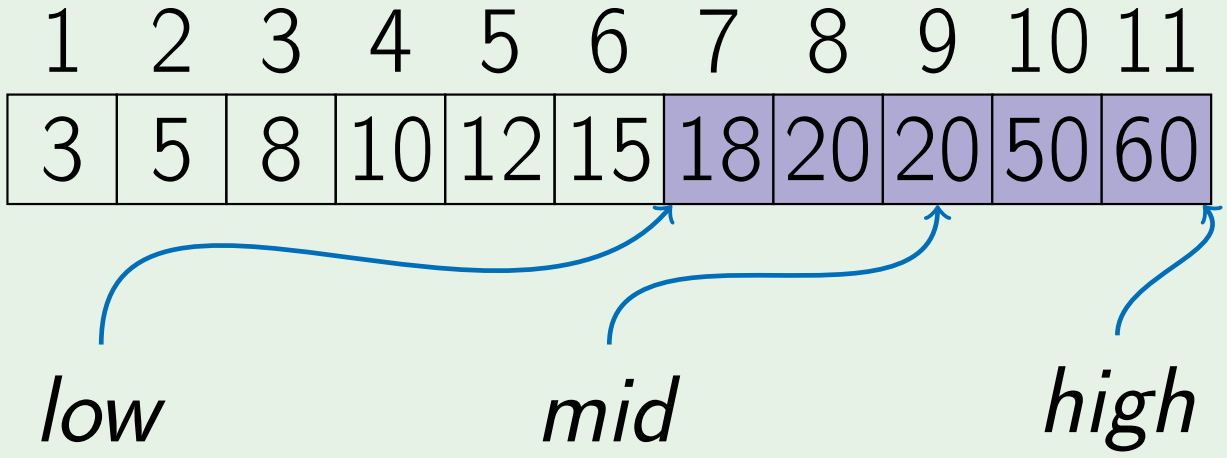
Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$
$\texttt{BinarySearch}(A, 7, 11, 50)$



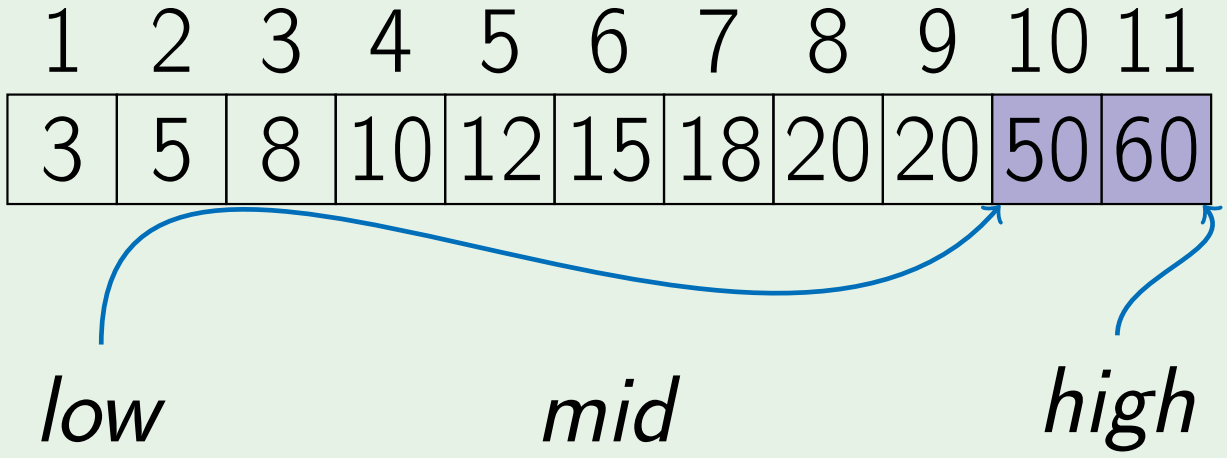
Example: Searching for the key 50

`BinarySearch`$(A, 1, 11, 50)$
`BinarySearch`$(A, 7, 11, 50)$



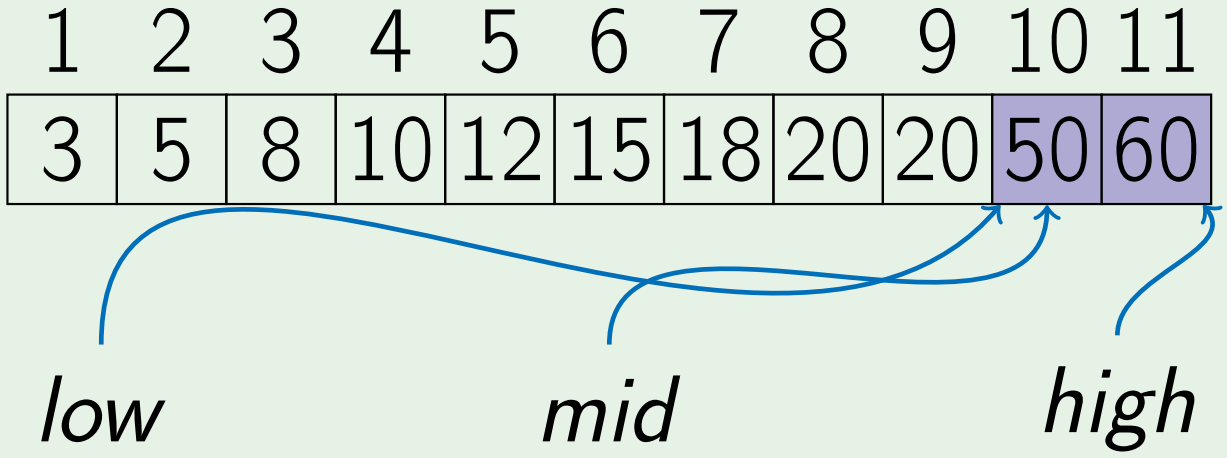
Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$
$\texttt{BinarySearch}(A, 7, 11, 50)$
$\texttt{BinarySearch}(A, 10, 11, 50)$



Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$
$\texttt{BinarySearch}(A, 7, 11, 50)$
$\texttt{BinarySearch}(A, 10, 11, 50)$



Example: Searching for the key 50

$\texttt{BinarySearch}(A, 1, 11, 50)$
$\texttt{BinarySearch}(A, 7, 11, 50)$
$\texttt{BinarySearch}(A, 10, 11, 50) \rightarrow 10$

1	2	3	4	5	6	7	8	9	10	11
3	5	8	10	12	15	18	20	20	50	60

Summary

- Break problem into non-overlapping subproblems of the same type.
- Recursively solve those subproblems.
- Combine results of subproblems.

BinarySearch(A, low, $high$, key)

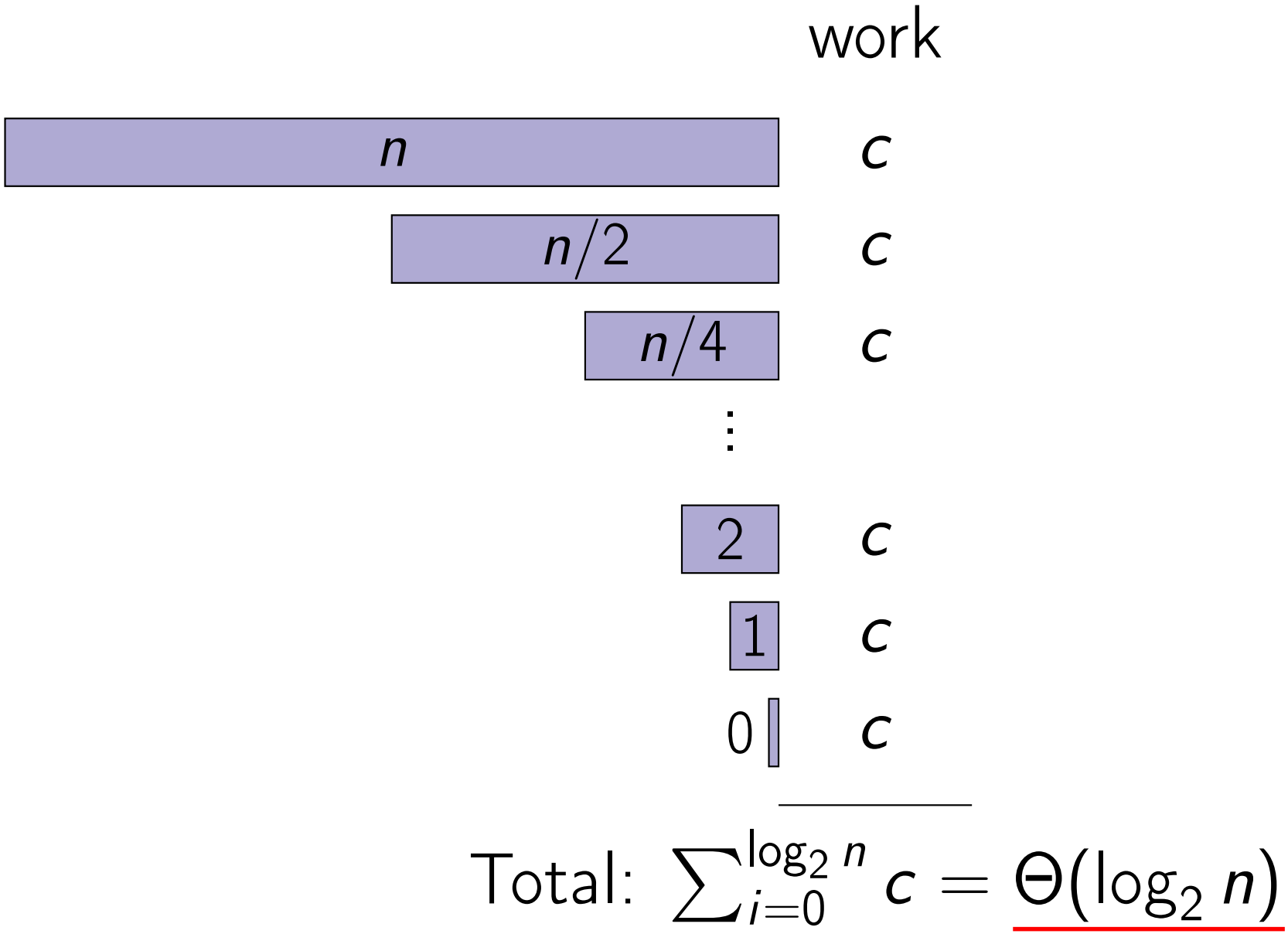
```
if high < low:
    return low − 1
mid ← ⌊low + (high − low)/2⌋
if key = A[mid]:
    return mid
else if key < A[mid]:
    return BinarySearch(A, low, mid − 1, key)
else:
    return BinarySearch(A, mid + 1, high, key)
```

Binary Search Recurrence Relation

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c$$

$$T(0) = c$$

Runtime of Binary Search



Total: $\sum_{i=0}^{\log_2 n} c = \Theta(\log_2 n)$

Iterative Version

BinarySearchIt(A, low, $high$, key)

```
while low ≤ high:
    mid ← ⌊low + (high − low)/2⌋
    if key = A[mid]:
        return mid
    else if key < A[mid]:
        high = mid − 1
    else:
        low = mid + 1
return low − 1
```

Real-life Example

english	**french**	**italian**	**german**	**spanish**
house	maison	casa	Haus	casa
chair	chaise	sedia	Sessel	silla
pimple	bouton	foruncolo	Pickel	espenilla

english
sorted

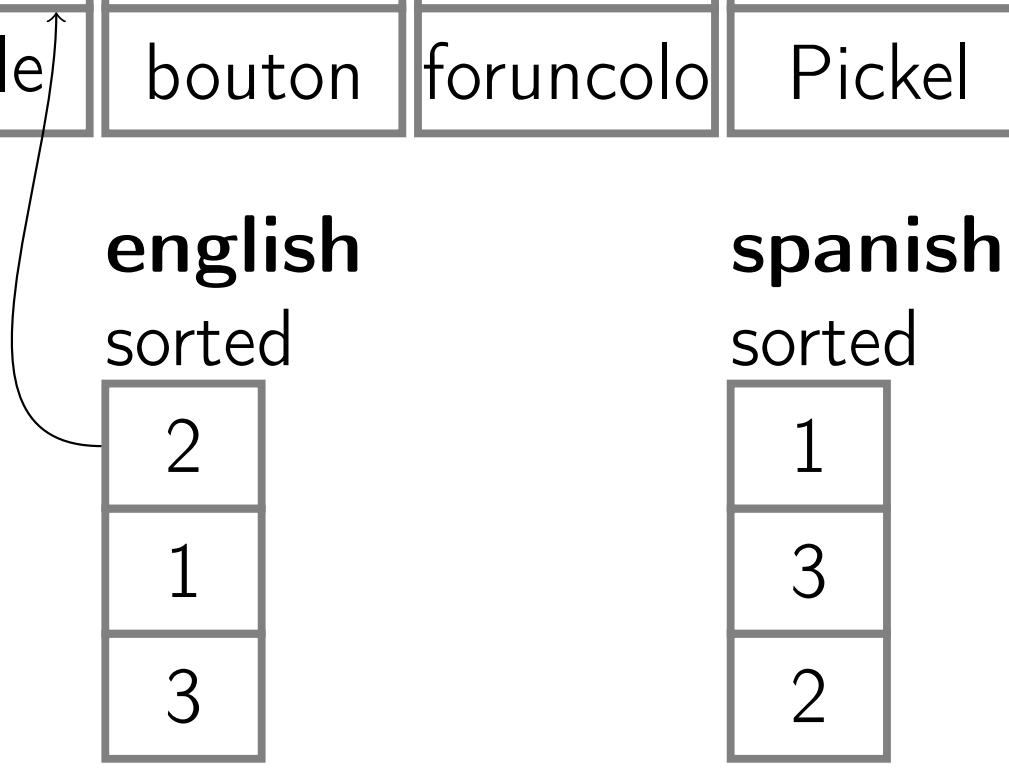
2
1
3

spanish
sorted

1
3
2

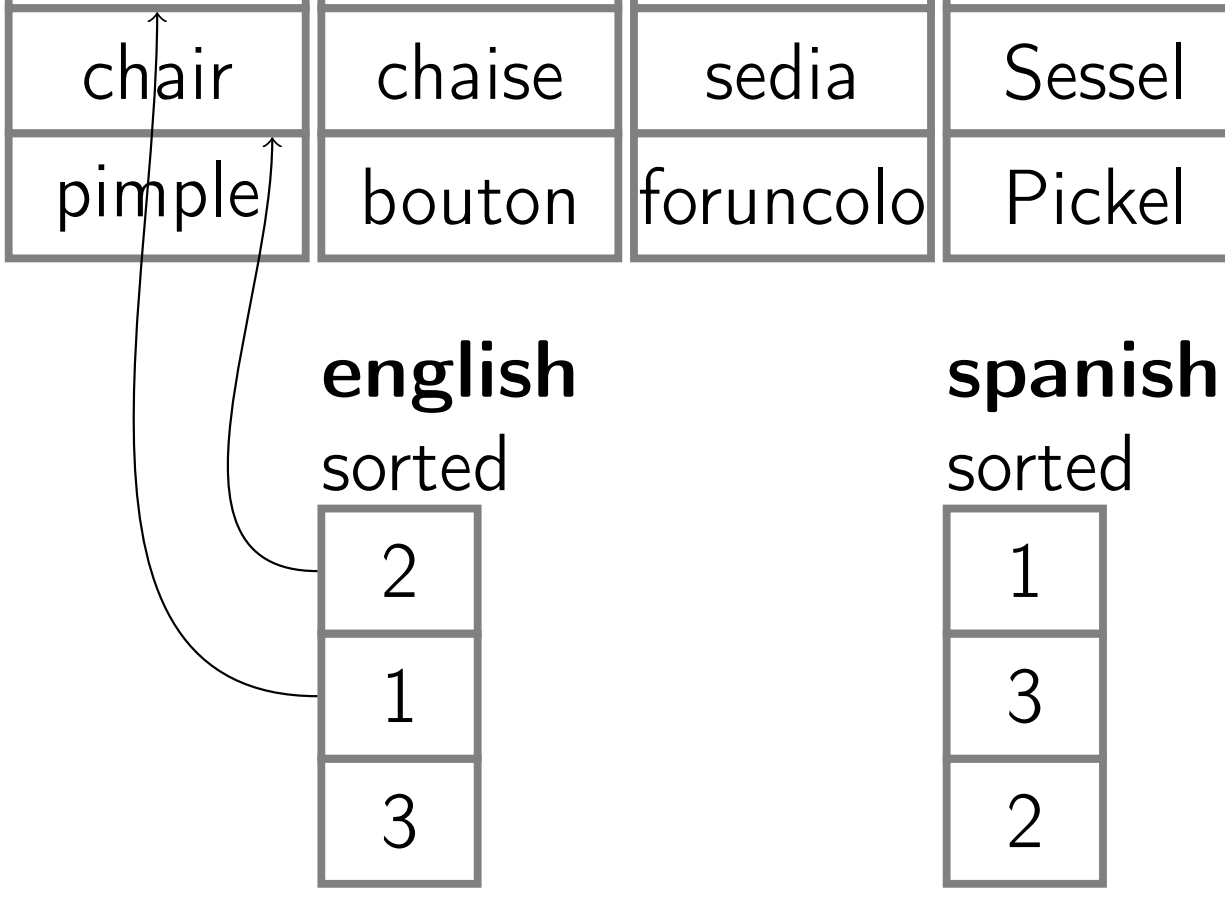
Real-life Example

english	**french**	**italian**	**german**	**spanish**
house	maison	casa	Haus	casa
chair	chaise	sedia	Sessel	silla
pimple	bouton	foruncolo	Pickel	espenilla



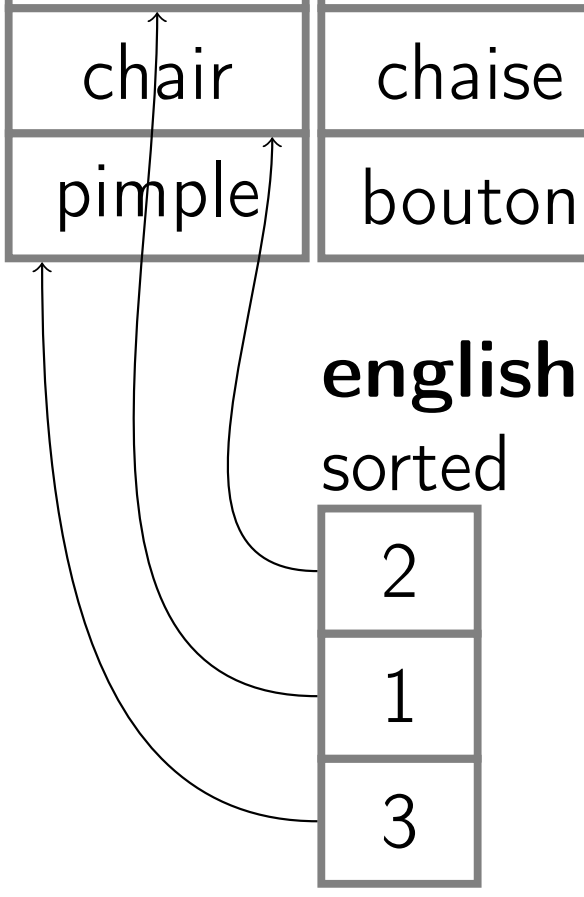
Real-life Example

english	**french**	**italian**	**german**	**spanish**
house	maison	casa	Haus	casa
chair	chaise	sedia	Sessel	silla
pimple	bouton	foruncolo	Pickel	espenilla



Real-life Example

english	**french**	**italian**	**german**	**spanish**
house	maison	casa	Haus	casa
chair	chaise	sedia	Sessel	silla
pimple	bouton	foruncolo	Pickel	espenilla



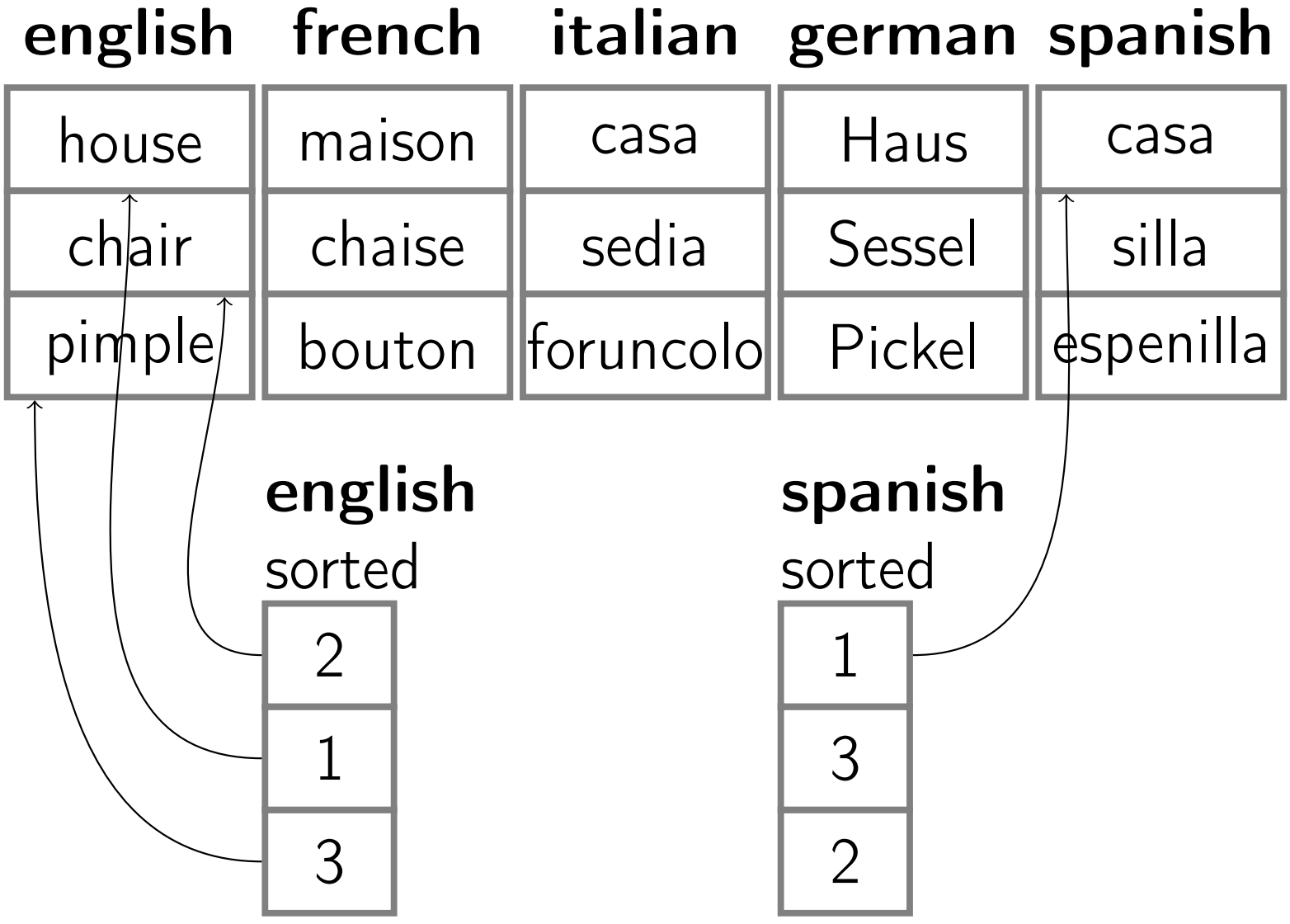
english
sorted

2
1
3

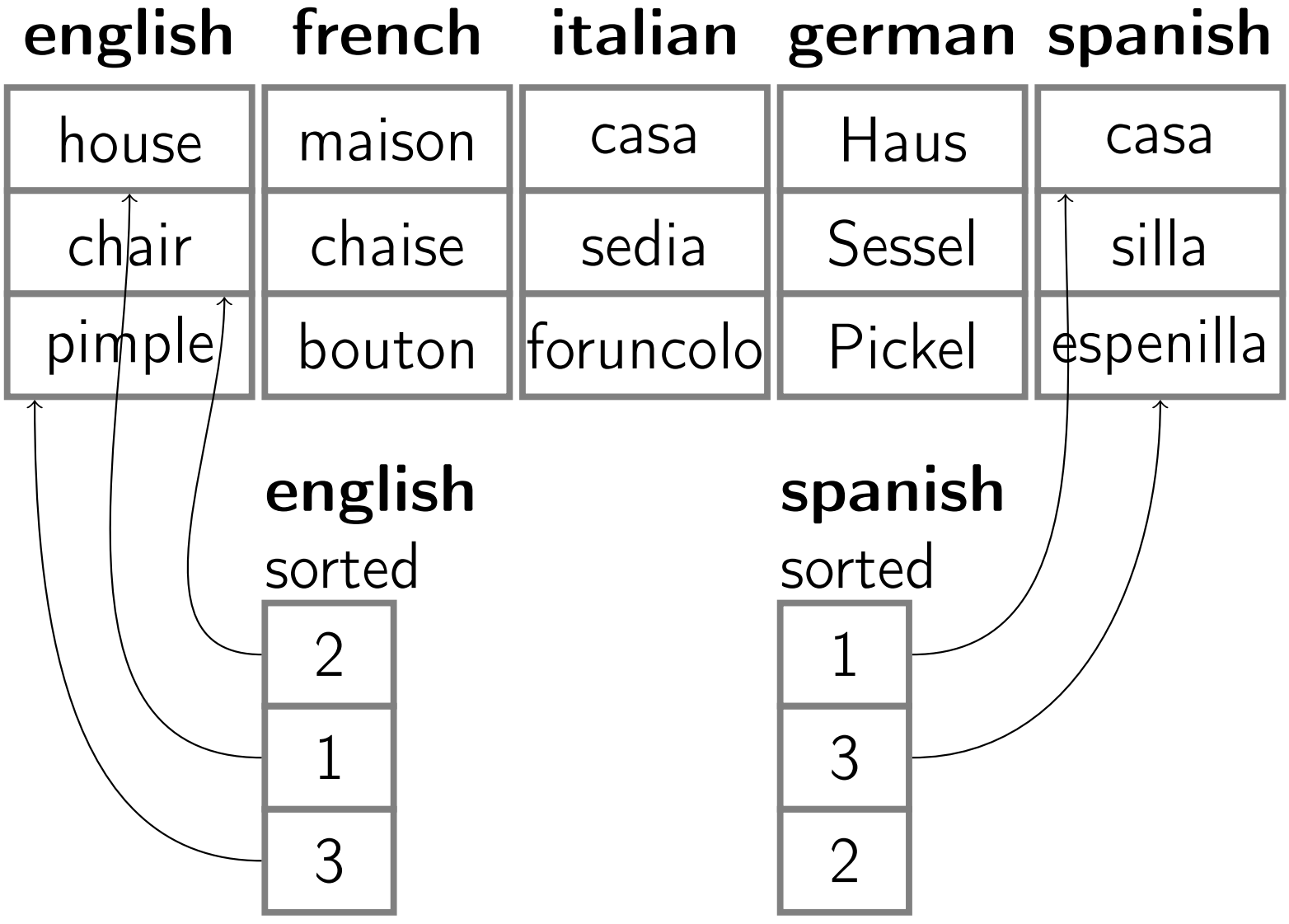
spanish
sorted

1
3
2

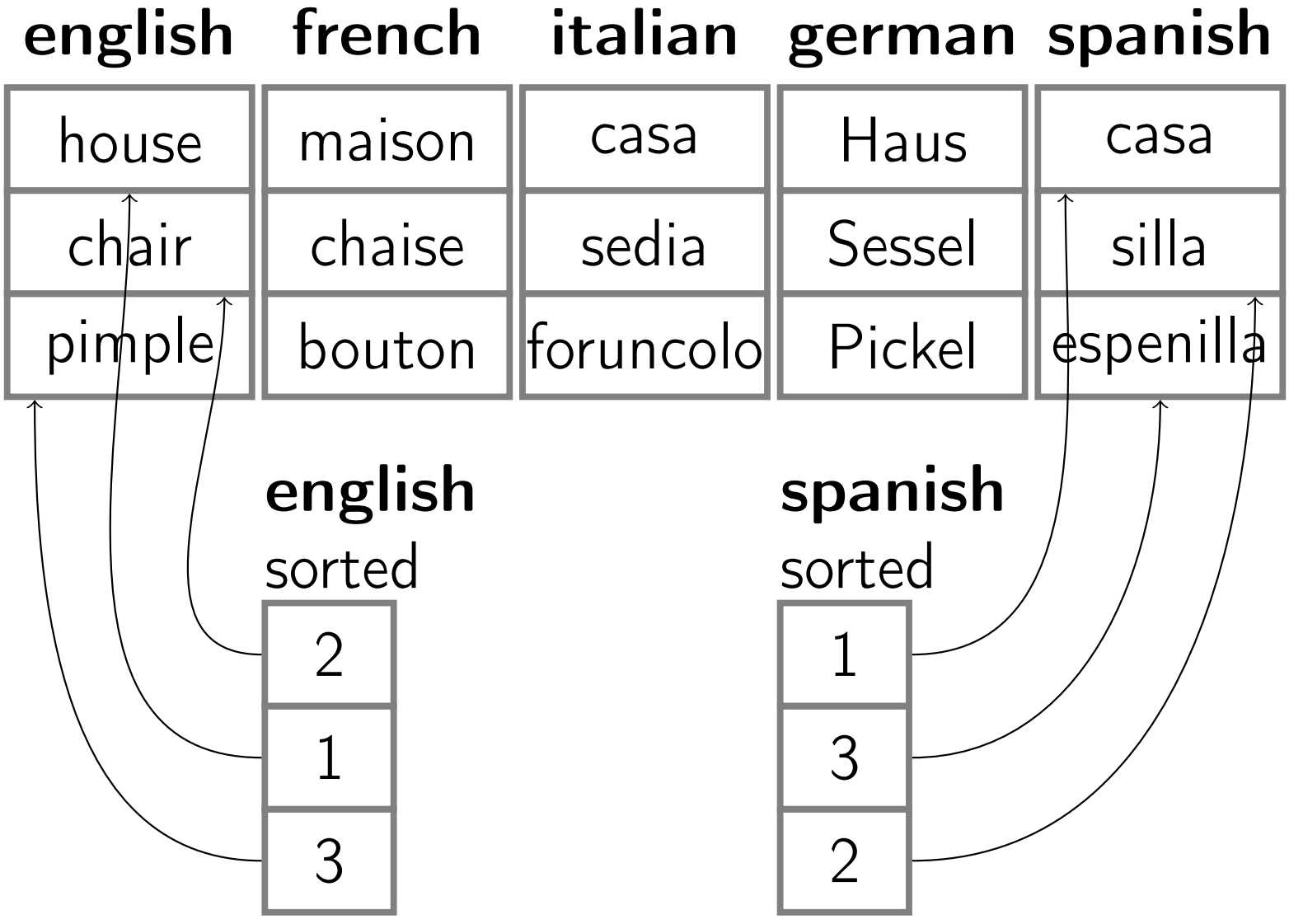
Real-life Example



Real-life Example



Real-life Example



Summary

The runtime of binary search is $\Theta(\log n)$.

Divide-and-Conquer: Polynomial Multiplication

Neil Rhodes

Department of Computer Science and Engineering
University of California, San Diego

Data Structures and Algorithms
Algorithmic Toolbox

Outline

1. Problem Overview
2. Naïve Algorithm
3. Naïve Divide and Conquer Algorithm
4. Faster Divide and Conquer

Uses of multiplying polynomials

- Error-correcting codes
- Large-integer multiplication
- Generating functions
- Convolution in signal processing

Multiplying Polynomials

Example

$$A(x) = 3x^2 + 2x + 5$$
$$B(x) = 5x^2 + x + 2$$
$$A(x)B(x) = 15x^4 + 13x^3 + 33x^2 + 9x + 10$$

Multiplying polynomials

Input: Two $n-1$ degree polynomials:

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

$$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0$$

Output: The product polynomial:

$$c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \cdots + c_1 x + c_0$$

where:

$$c_{2n-2} = a_{n-1}b_{n-1}$$

$$c_{2n-3} = a_{n-1}b_{n-2} + a_{n-2}b_{n-1}$$

...

$$c_2 = a_2 b_0 + a_1 b_1 + a_0 b_2$$

$$c_1 = a_1 b_0 + a_0 b_1$$

$$c_0 = a_0 b_0$$

Multiplying Polynomials

Example

Input: $n = 3, A = (3, 2, 5), B = (5, 1, 2)$

$$A(x) = 3x^2 + 2x + 5$$
$$B(x) = 5x^2 + x + 2$$
$$A(x)B(x) = 15x^4 + 13x^3 + 33x^2 + 9x + 10$$

Output: $C = (15, 13, 33, 9, 10)$

Outline

1. Problem Overview
2. Naïve Algorithm
3. Naïve Divide and Conquer Algorithm
4. Faster Divide and Conquer

MultPoly(A, B, n)

```
pair ← Array[n][n]
for i from 0 to n − 1:
  for j from 0 to n − 1:
    pair[i][j] ← A[i] * B[j]
product ← Array[2n − 1]
for i from 0 to 2n − 1:
  product[i] ← 0
for i from 0 to n − 1:
  for j from 0 to n − 1:
    product[i + j] ← product[i + j] + pair[i][j]
return product
```

Multiplying Polynomials

Naïve Solution: $O(n^2)$

- Multiply all $a_i b_j$ pairs (n^2 multiplications)
- Sum needed pairs (n^2 additions)

Outline

1. Problem Overview
2. Naïve Algorithm
3. Naïve Divide and Conquer Algorithm
4. Faster Divide and Conquer

Multiplying Polynomials

- Let $A(x) = D_1(x)x^{\frac{n}{2}} + D_0(x)$ where
  $D_1(x) = a_{n-1}x^{\frac{n}{2}-1} + a_{n-2}x^{\frac{n}{2}-2} + \ldots + a_{\frac{n}{2}}$
  $D_0(x) = a_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + a_{\frac{n}{2}-2}x^{\frac{n}{2}-2} + \ldots + a_0$
- Let $B(x) = E_1(x)x^{\frac{n}{2}} + E_0(x)$ where
  $E_1(x) = b_{n-1}x^{\frac{n}{2}-1} + b_{n-2}x^{\frac{n}{2}-2} + \ldots + b_{\frac{n}{2}}$
  $E_0(x) = b_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + b_{\frac{n}{2}-2}x^{\frac{n}{2}-2} + \ldots + b_0$
- $AB = (D_1x^{\frac{n}{2}} + D_0)(E_1x^{\frac{n}{2}} + E_0)$
  $= (D_1E_1)x^n + (D_1E_0 + D_0E_1)x^{\frac{n}{2}} + D_0E_0$
- Calculate D_1E_1, D_1E_0, D_0E_1, and D_0E_0

Recurrence: $T(n) = 4T(\frac{n}{2}) + kn$.

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2 \qquad D_0E_0 = 6x^2 + 11x + 4$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$AB =$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

$$(12x^2 + 25x + 12 \qquad)x^2 +$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

$$(12x^2 + 25x + 12 + 2x^2 + 5x + 2)x^2 +$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$AB = (4x^2 + 11x + 6)x^4 +$$
$$(12x^2 + 25x + 12 + 2x^2 + 5x + 2)x^2 +$$
$$6x^2 + 11x + 4$$

Polynomial Mult: Divide & Conquer

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_1E_0 = 12x^2 + 25x + 12$$

$$D_0E_1 = 2x^2 + 5x + 2 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$\begin{aligned} AB = &(4x^2 + 11x + 6)x^4 + \\ &(12x^2 + 25x + 12 + 2x^2 + 5x + 2)x^2 + \\ &6x^2 + 11x + 4 \\ =&4x^6 + 11x^5 + 20x^4 + 30x^3 + 20x^2 + 11x + 4 \end{aligned}$$

Function Mult2(A, B, n, a_l, b_l)

$R =$ array$[0..2n-2]$

if $n = 1$:

$R[0] = A[a_l] * B[b_l]$; return R

$R[0..n-2]$ = Mult2$(A, B, \frac{n}{2}, a_l, b_l)$

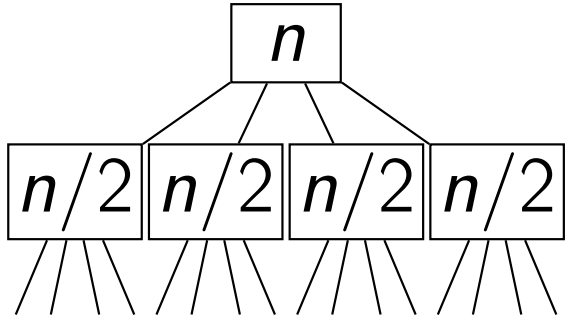
$R[n..2n-2]$ = Mult2$(A, B, \frac{n}{2}, a_l + \frac{n}{2}, b_l + \frac{n}{2})$

D_0E_1 = Mult2$(A, B, \frac{n}{2}, a_l, b_l + \frac{n}{2})$

D_1E_0 = Mult2$(A, B, \frac{n}{2}, a_l + \frac{n}{2}, b_l)$

$R[\frac{n}{2}...n + \frac{n}{2} - 2]$ += $D_1E0 + D_0E_1$

return R

level		#	work
0		1	kn
1		4	$4k\frac{n}{2} = k2n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
i	$\cdots$ $n/2^i$ $\cdots$	4^i	$4^i k\frac{n}{2^i} = k2^i n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\log_2 n$	1 $\cdots$ 1	$4^{\log_2 n}$	$k4^{\log_2 n} = kn^2$

Total: $\sum_{i=0}^{\log_2 n} 4^i k\frac{n}{2^i} = \Theta(n^2)$

Outline

1. Problem Overview
2. Naïve Algorithm
3. Naïve Divide and Conquer Algorithm
4. Faster Divide and Conquer

Karatsuba approach

$$A(x) = a_1 x + a_0$$

$$B(x) = b_1 x + b_0$$

$$C(x) = a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0$$

Needs 4 multiplications

Karatsuba approach

$$A(x) = a_1 x + a_0$$

$$B(x) = b_1 x + b_0$$

$$C(x) = a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0$$

Needs 4 multiplications

Rewrite as:

$$C(x) = a_1 b_1 x^2 + ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0)x + a_0 b_0$$

Karatsuba approach

$$A(x) = a_1 x + a_0$$

$$B(x) = b_1 x + b_0$$

$$C(x) = a_1 b_1 x^2 + (a_1 b_0 + a_0 b_1)x + a_0 b_0$$

Needs 4 multiplications

Rewrite as:

$$C(x) = a_1 b_1 x^2 + ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0)x + a_0 b_0$$

Needs 3 multiplications

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) =$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6)$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$\begin{aligned}(D_1 + D_0)(E_1 + E_0) &= (6x + 4)(4x + 6) \\ &= 24x^2 + 52x + 24\end{aligned}$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6)$$

$$= 24x^2 + 52x + 24$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6)$$

$$= 24x^2 + 52x + 24$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

$$(24x^2 + 52x + 24$$

$$)x^2 +$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6)$$

$$= 24x^2 + 52x + 24$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

$$(24x^2 + 52x + 24 - (4x^2 + 11x + 6)$$

$$)x^2 +$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6)$$

$$= 24x^2 + 52x + 24$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

$$(24x^2 + 52x + 24 - (4x^2 + 11x + 6)$$

$$- (6x^2 + 11x + 4))x^2 +$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$(D_1 + D_0)(E_1 + E_0) = (6x + 4)(4x + 6)$$

$$= 24x^2 + 52x + 24$$

$$AB = (4x^2 + 11x + 6)x^4 +$$

$$(24x^2 + 52x + 24 - (4x^2 + 11x + 6)$$

$$- (6x^2 + 11x + 4))x^2 +$$

$$6x^2 + 11x + 4$$

Karatsuba Example

$$A(x) = 4x^3 + 3x^2 + 2x + 1$$

$$B(x) = x^3 + 2x^2 + 3x + 4$$

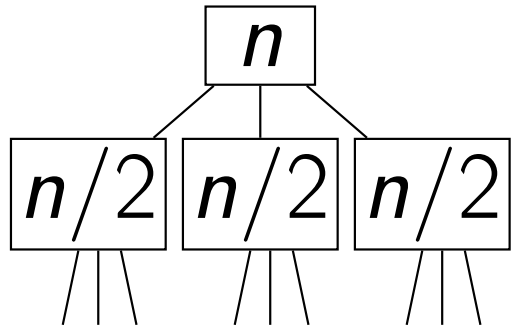
$$D_1(x) = 4x + 3 \qquad D_0(x) = 2x + 1$$

$$E_1(x) = x + 2 \qquad E_0(x) = 3x + 4$$

$$D_1E_1 = 4x^2 + 11x + 6 \qquad D_0E_0 = 6x^2 + 11x + 4$$

$$\begin{aligned}(D_1 + D_0)(E_1 + E_0) &= (6x + 4)(4x + 6) \\ &= 24x^2 + 52x + 24\end{aligned}$$

$$\begin{aligned}AB = &(4x^2 + 11x + 6)x^4 + \\ &(24x^2 + 52x + 24 - (4x^2 + 11x + 6) \\ &\qquad - (6x^2 + 11x + 4))x^2 + \\ &6x^2 + 11x + 4 \\ =&4x^6 + 11x^5 + 20x^4 + 30x^3 + 20x^2 + 11x + 4\end{aligned}$$

level		#	work
0		1	kn
1		3	$3k\frac{n}{2} = k\frac{3}{2}n$
⋮	⋮	⋮	⋮
i	$\cdots$ $n/2^i$ $\cdots$	3^i	$3^i k\frac{n}{2^i} = k(\frac{3}{2})^i n$
⋮	⋮	⋮	⋮
$\log_2 n$	1 $\cdots$ 1	$3^{\log_2 n}$	$k3^{\log_2 n} = kn^{\log_2 3}$

Total: $\sum_{i=0}^{\log_2 n} 3^i k\frac{n}{2^i} = \Theta(n^{\log_2 3})$

$= \Theta(n^{1.58})$

Divide-and-Conquer: Master Theorem

Neil Rhodes

Department of Computer Science and Engineering
University of California, San Diego

Data Structures and Algorithms
Algorithmic Toolbox

Outline

1. What is the Master Theorem
2. Proof of Master Theorem

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

$$T(n) = O(\log n)$$

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = O(n^2)$$

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = O(n^{\log_2 3})$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = O(n \log n)$$

Master Theorem

Theorem

If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ (for constants $a > 0, b > 1, d \geq 0$), then:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Master Theorem Example 1

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

$$a = 4$$

$$b = 2$$

$$d = 1$$

Since $d < \log_b a$, $T(n) = O(n^{\log_b a}) = O(n^2)$

Master Theorem Example 2

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

$$a = 3$$

$$b = 2$$

$$d = 1$$

Since $d < \log_b a$,
$T(n) = O(n^{\log_b a}) = O(n^{\log_2 3})$

Master Theorem Example 3

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$a = 2$$

$$b = 2$$

$$d = 1$$

Since $d = \log_b a$,
$T(n) = O(n^d \log n) = O(n \log n)$

Master Theorem Example 4

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

$$a = 1$$

$$b = 2$$

$$d = 0$$

Since $d = \log_b a$, $T(n) = O(n^d \log n) = O(n^0 \log n) = O(\log n)$

Master Theorem Example 5

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^2)$$

$$a = 2$$

$$b = 2$$

$$d = 2$$

Since $d > \log_b a$, $T(n) = O(n^d) = O(n^2)$

Outline

1. What is the Master Theorem
2. Proof of Master Theorem

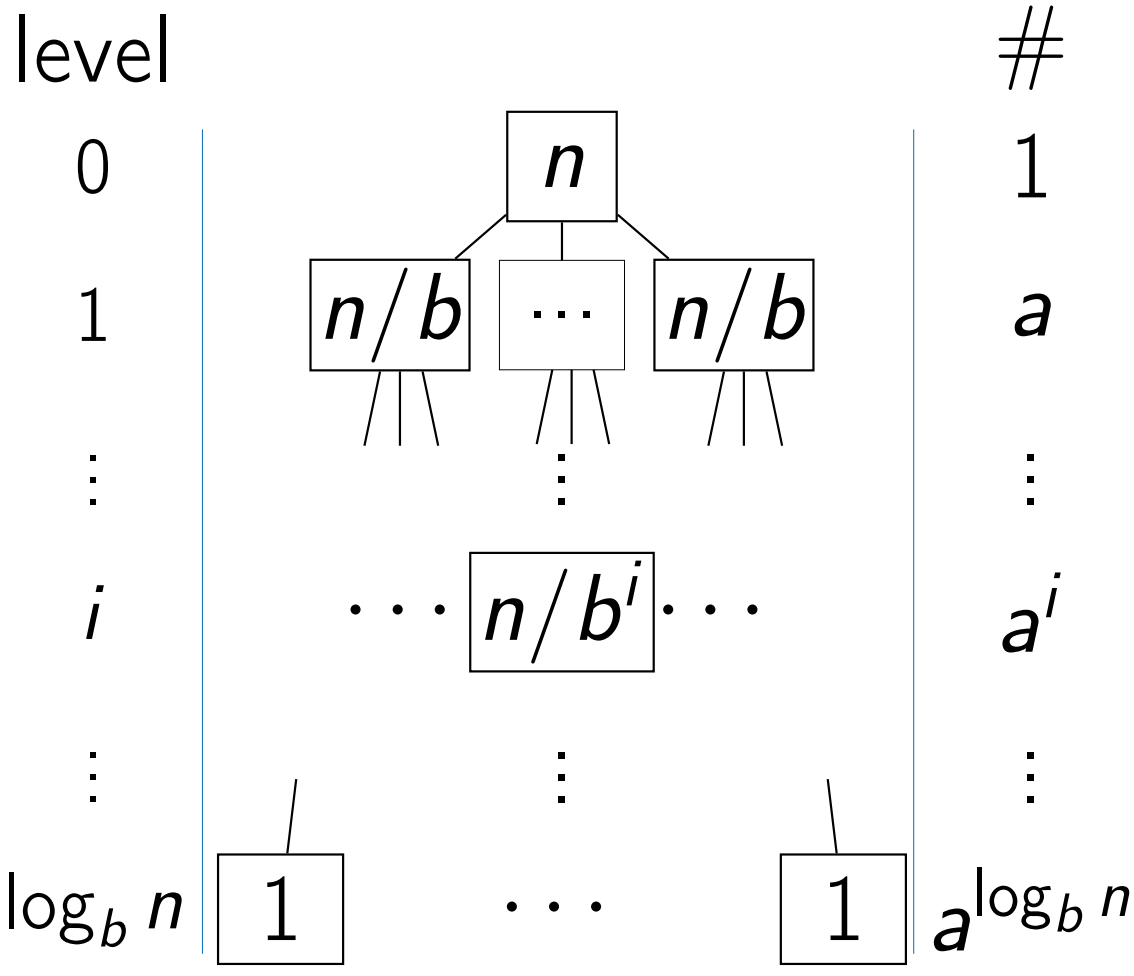
Master Theorem

Theorem

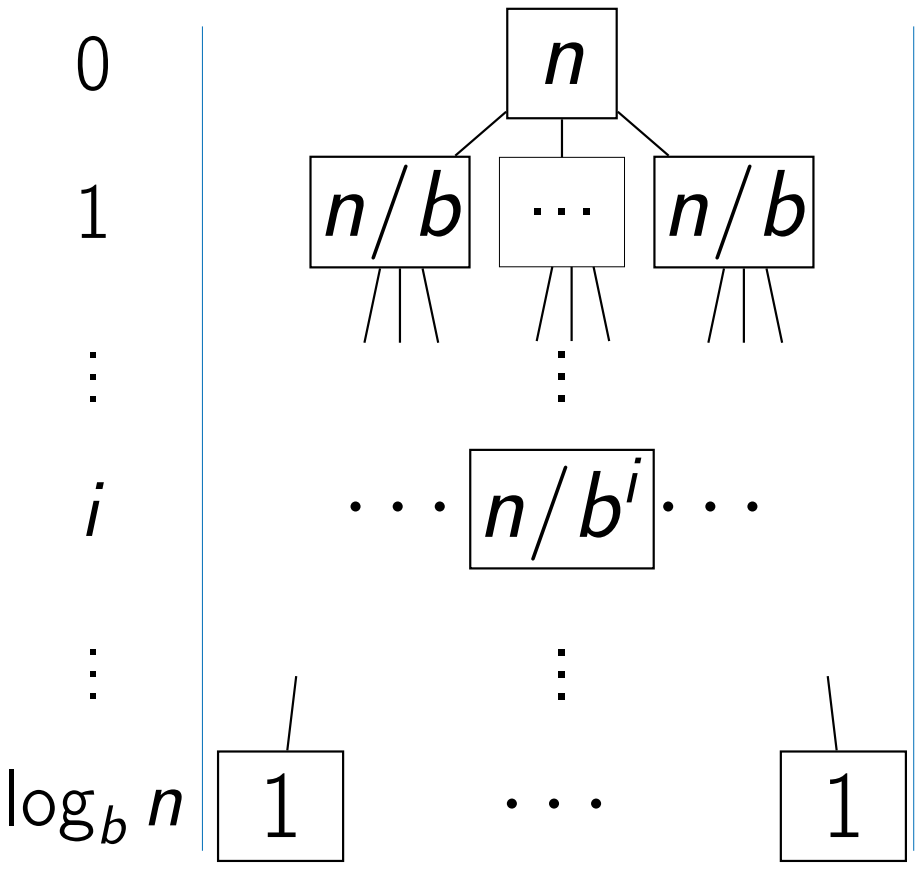
If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ (for constants $a > 0, b > 1, d \geq 0$), then:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$



$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$



Geometric Series

For $r \neq 1$:

$$
\begin{aligned}
& a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} \\
=& a\frac{1 - r^n}{1 - r} \\
=& \begin{cases} O(a) & \text{if } r < 1 \\ O(ar^{n-1}) & \text{if } r > 1 \end{cases}
\end{aligned}
$$

Case 1: $\frac{a}{b^d} < 1$ ($d > log_b a$)

$$\sum_{i=0}^{\log_b n} O(n^d)\left(\frac{a}{b^d}\right)^i$$

$$= O(n^d)$$

Case 2: $\frac{a}{b^d} = 1$ ($d = log_b a$)

$$\sum_{i=0}^{\log_b n} O(n^d)\left(\frac{a}{b^d}\right)^i$$

$$= \sum_{i=0}^{\log_b n} O(n^d)$$

$$= (1 + \log_b n) O(n^d)$$

$$= O(n^d \log n)$$

Case 3: $\frac{a}{b^d} > 1$ ($d < log_b a$)

$$\sum_{i=0}^{\log_b n} O(n^d)\left(\frac{a}{b^d}\right)^i$$

$$= O\left(O(n^d)\left(\frac{a}{b^d}\right)^{\log_b n}\right)$$

$$= O\left(O(n^d)\frac{a^{\log_b n}}{b^{d \log_b n}}\right)$$

$$= O\left(O(n^d)\frac{n^{\log_b a}}{n^d}\right)$$

$$= O(n^{\log_b a})$$

Summary

Master theorem is a shortcut:

Theorem

If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ (for constants $a > 0, b > 1, d \geq 0$), then:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Divide-and-Conquer: Sorting Problem

Alexander S. Kulikov

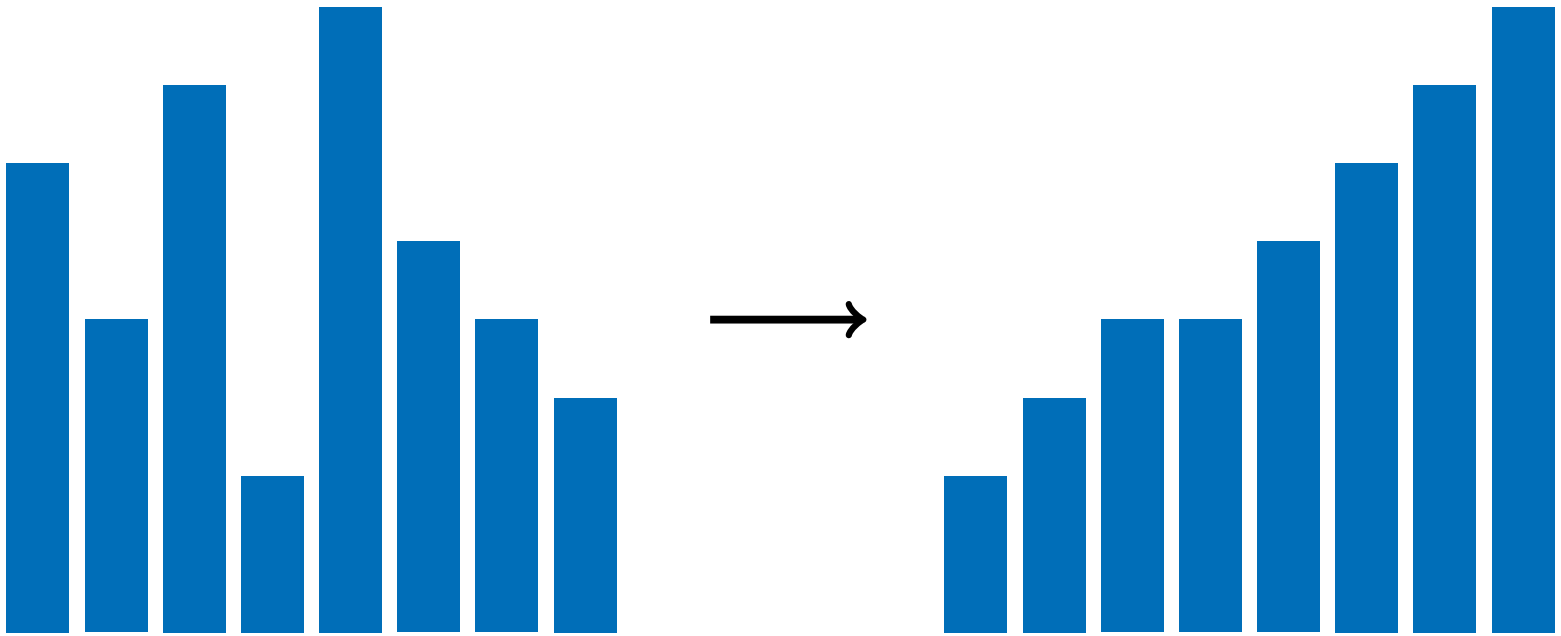
Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Data Structures and Algorithms
Algorithmic Toolbox

Outline

1. Problem Overview
2. Selection Sort
3. Merge Sort
4. Lower Bound for Comparison Based Sorting
5. Non-Comparison Based Sorting Algorithms

Sorting Problem



Sorting

Input: Sequence $A[1 \ldots n]$.

Output: Permutation $A'[1 \ldots n]$ of $A[1 \ldots n]$ in non-decreasing order.

Why Sorting?

- Sorting data is an important step of many efficient algorithms.
- Sorted data allows for more efficient queries.

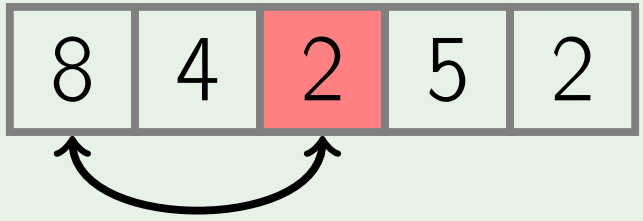
Outline

1. Problem Overview
2. Selection Sort
3. Merge Sort
4. Lower Bound for Comparison Based Sorting
5. Non-Comparison Based Sorting Algorithms

Selection sort: example

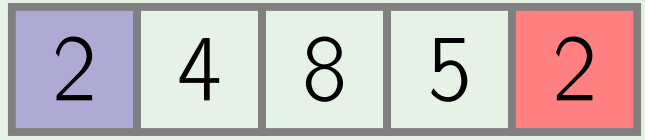
- Find a minimum by scanning the array

Selection sort: example



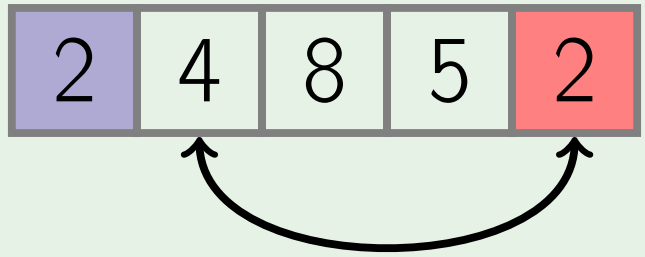
- Find a minimum by scanning the array
- Swap it with the first element

Selection sort: example



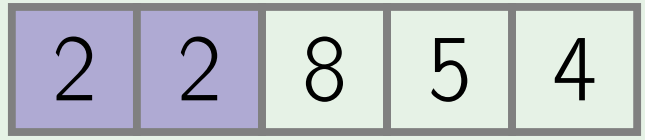
- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example



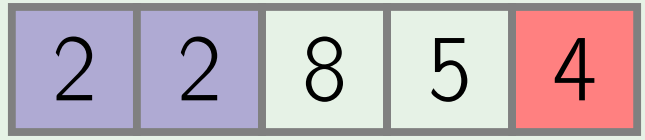
- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example



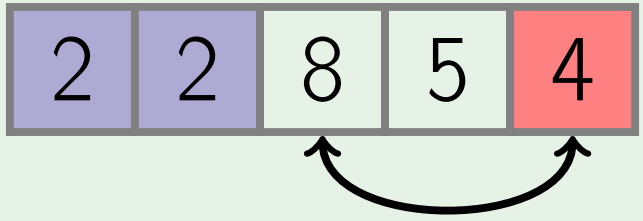
- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example



- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example



- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example

- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example

- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example

- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

Selection sort: example

- Find a minimum by scanning the array
- Swap it with the first element
- Repeat with the remaining part of the array

SelectionSort($A[1 \ldots n]$)

```
for i from 1 to n:
    minIndex ← i
    for j from i + 1 to n:
        if A[j] < A[minIndex]:
            minIndex ← j
    {A[minIndex] = min A[i ... n]}
    swap(A[i], A[minIndex])
    {A[1 ... i] is in final position}
```

Online visualization: selection sort

Lemma

The running time of `SelectionSort`$(A[1 \ldots n])$ is $O(n^2)$.

Proof

n iterations of outer loop, at most n iterations of inner loop. □

Too Pessimistic Estimate?

- As i grows, the number of iterations of the inner loop decreases: j iterates from $i + 1$ to n.
- A more accurate estimate for the total number of iterations of the inner loop is $(n - 1) + (n - 2) + \cdots + 1$.
- We will show that this sum is $\Theta(n^2)$ implying that our initial estimate is actually tight.

Arithmetic Series

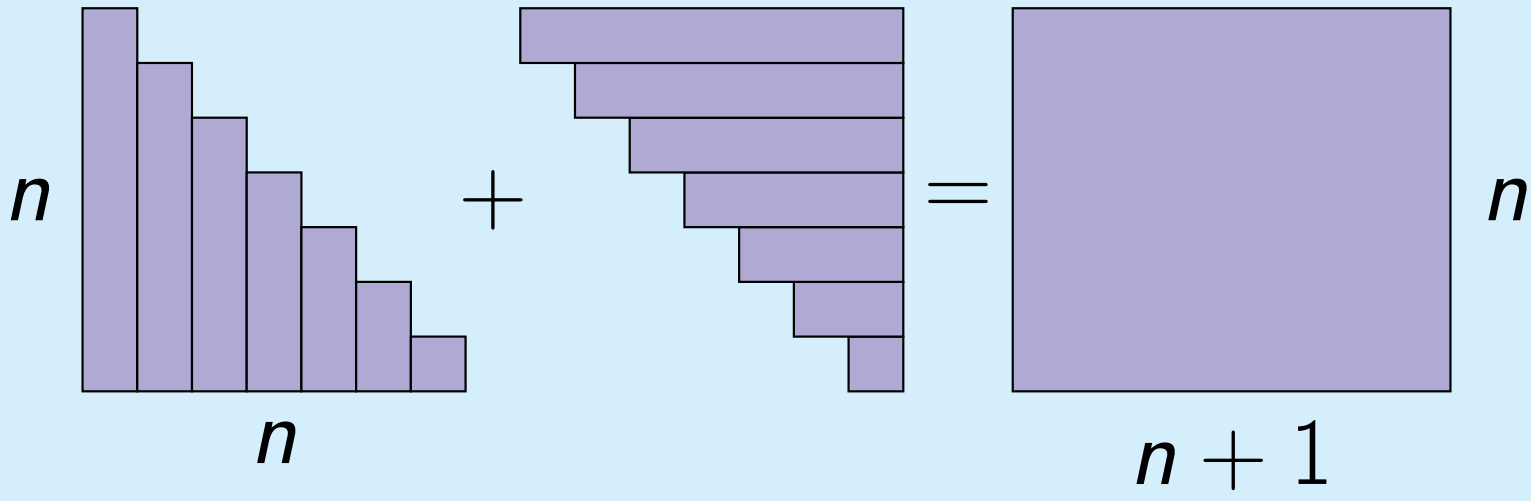
Lemma

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Proof

$$\begin{array}{cccc} 1 & 2 & \cdots & n \\ n & n-1 & \cdots & 1 \\ \hline n+1 & n+1 & \cdots & n+1 \end{array} = n(n+1) \quad \square$$

Alternative proof



□

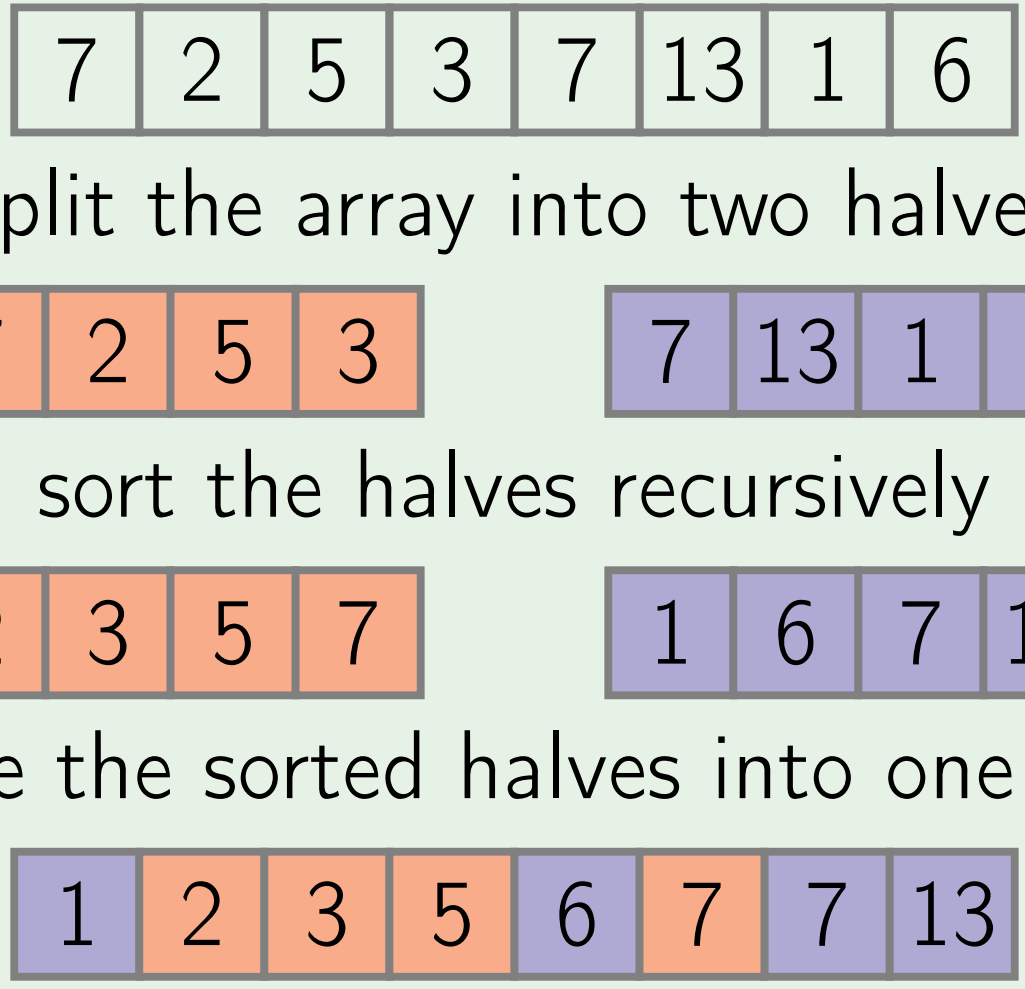
Selection Sort: Summary

- Selection sort is an easy to implement algorithm with running time $O(n^2)$.
- Sorts in place: requires a constant amount of extra memory.
- There are many other quadratic time sorting algorithms: e.g., insertion sort, bubble sort.

Outline

1. Problem Overview
2. Selection Sort
3. Merge Sort
4. Lower Bound for Comparison Based Sorting
5. Non-Comparison Based Sorting Algorithms

Example: merge sort



MergeSort($A[1 \ldots n]$)

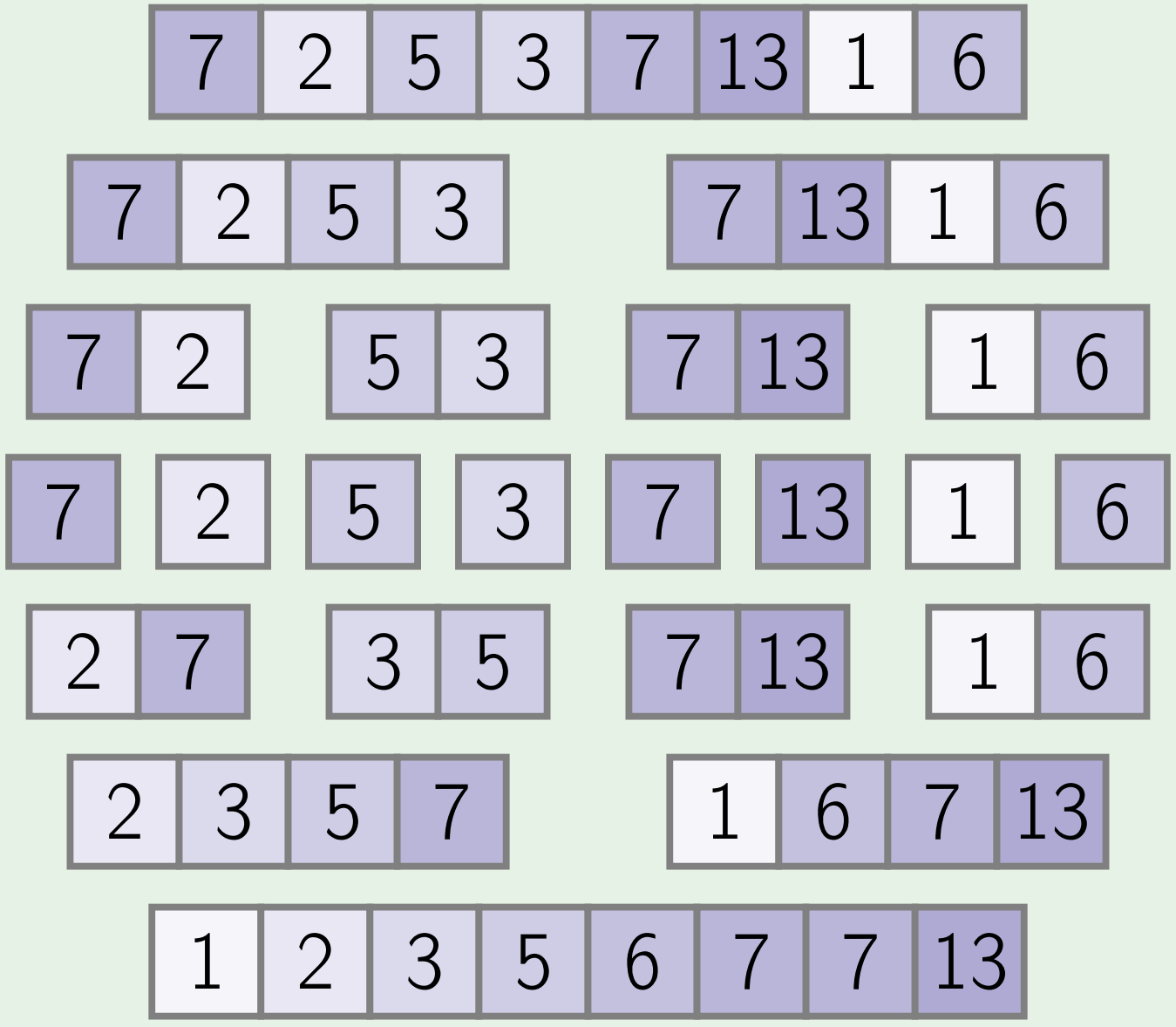
```
if n = 1:
  return A
m ← ⌊n/2⌋
B ← MergeSort(A[1 ... m])
C ← MergeSort(A[m + 1 ... n])
A' ← Merge(B, C)
return A'
```

Merging Two Sorted Arrays

Merge($B[1 \ldots p], C[1 \ldots q]$)

```
{B and C are sorted}
D ← empty array of size p + q
while B and C are both non-empty:
  b ← the first element of B
  c ← the first element of C
  if b ≤ c:
    move b from B to the end of D
  else:
    move c from C to the end of D
move the rest of B and C to the end of D
return D
```

Merge sort: example

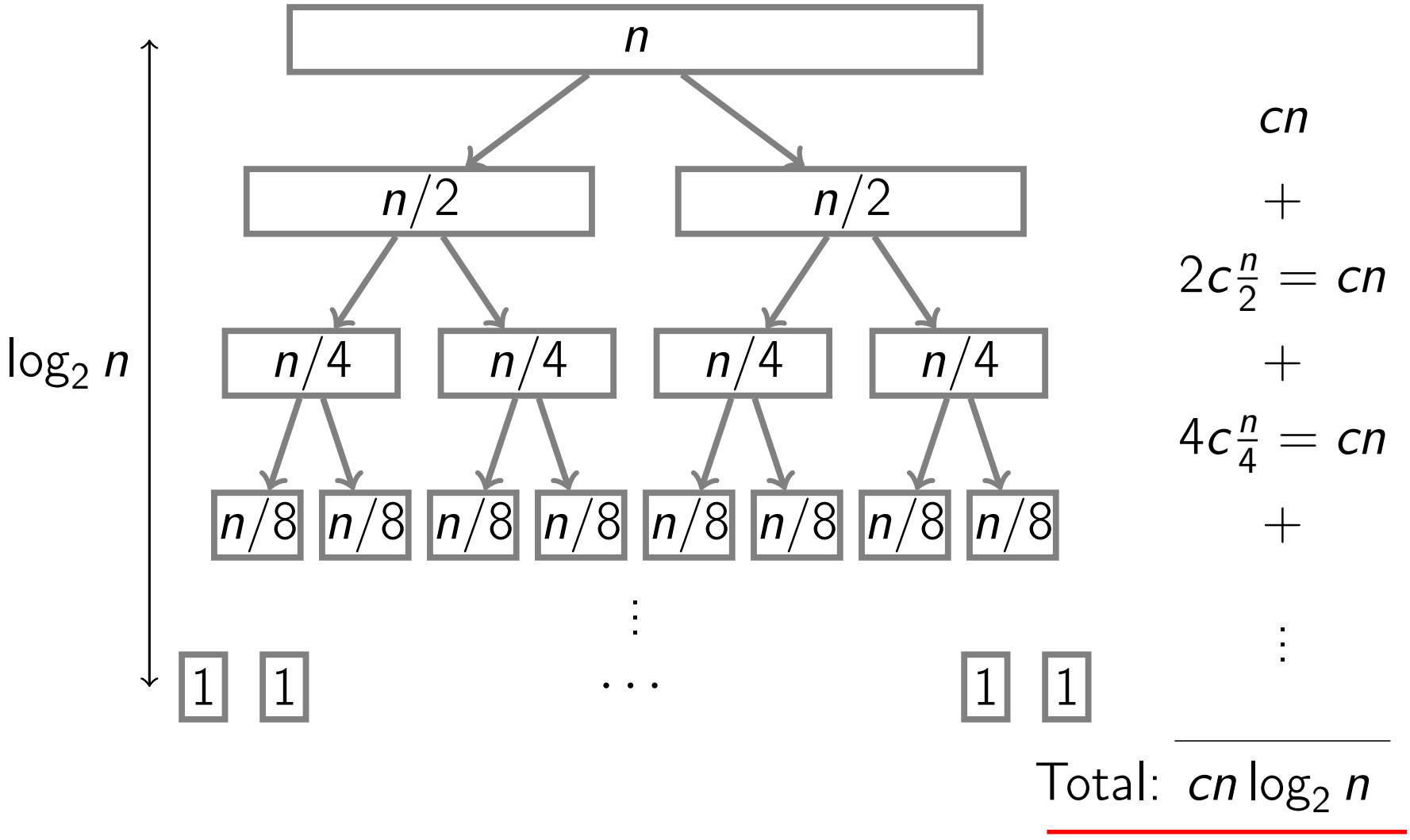


Lemma

The running time of `MergeSort`$(A[1 \ldots n])$ is $O(n \log n)$.

Proof

- The running time of merging B and C is $O(n)$.
- Hence the running time of `MergeSort`$(A[1 \ldots n])$ satisfies a recurrence $T(n) \leq 2T(n/2) + O(n)$.


work:
n
n/2
n/2
n/4
n/4
n/4
n/4
n/8
n/8
n/8
n/8
n/8
n/8
n/8
n/8
1
1
1
1
log2 n
cn
+
2c n/2 = cn
+
4c n/4 = cn
+
Total: cn log2 n

Outline

1. Problem Overview
2. Selection Sort
3. Merge Sort
4. Lower Bound for Comparison Based Sorting
5. Non-Comparison Based Sorting Algorithms

Definition

A comparison based sorting algorithm sorts objects by comparing pairs of them.

Example

Selection sort and merge sort are comparison based.

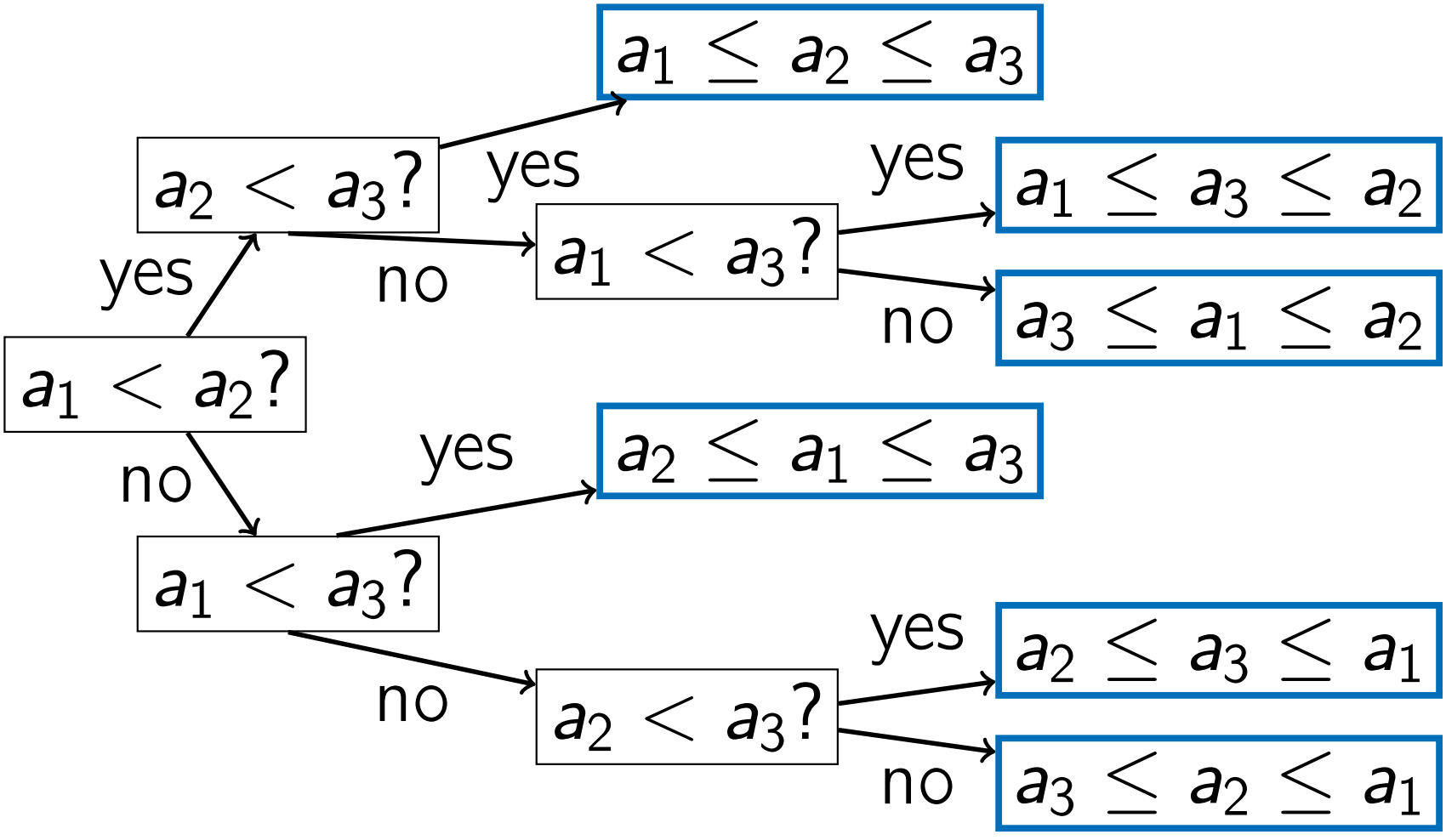
Lemma

Any comparison based sorting algorithm performs $\Omega(n \log n)$ comparisons in the worst case to sort n objects.

In other words

For any comparison based sorting algorithm, there exists an array $A[1 \ldots n]$ such that the algorithm performs at least $\Omega(n \log n)$ comparisons to sort A.

Decision Tree



Estimating Tree Depth

- the number of leaves ℓ in the tree must be at least $n!$ (the total number of permutations)
- the worst-case running time of the algorithm (the number of comparisons made) is at least the depth d
- $d \geq \log_2 \ell$ (or, equivalently, $2^d \geq \ell$)
- thus, the running time is at least

$$\log_2(n!) = \Omega(n \log n)$$

Lemma

$$\log_2(n!) = \Omega(n \log n)$$

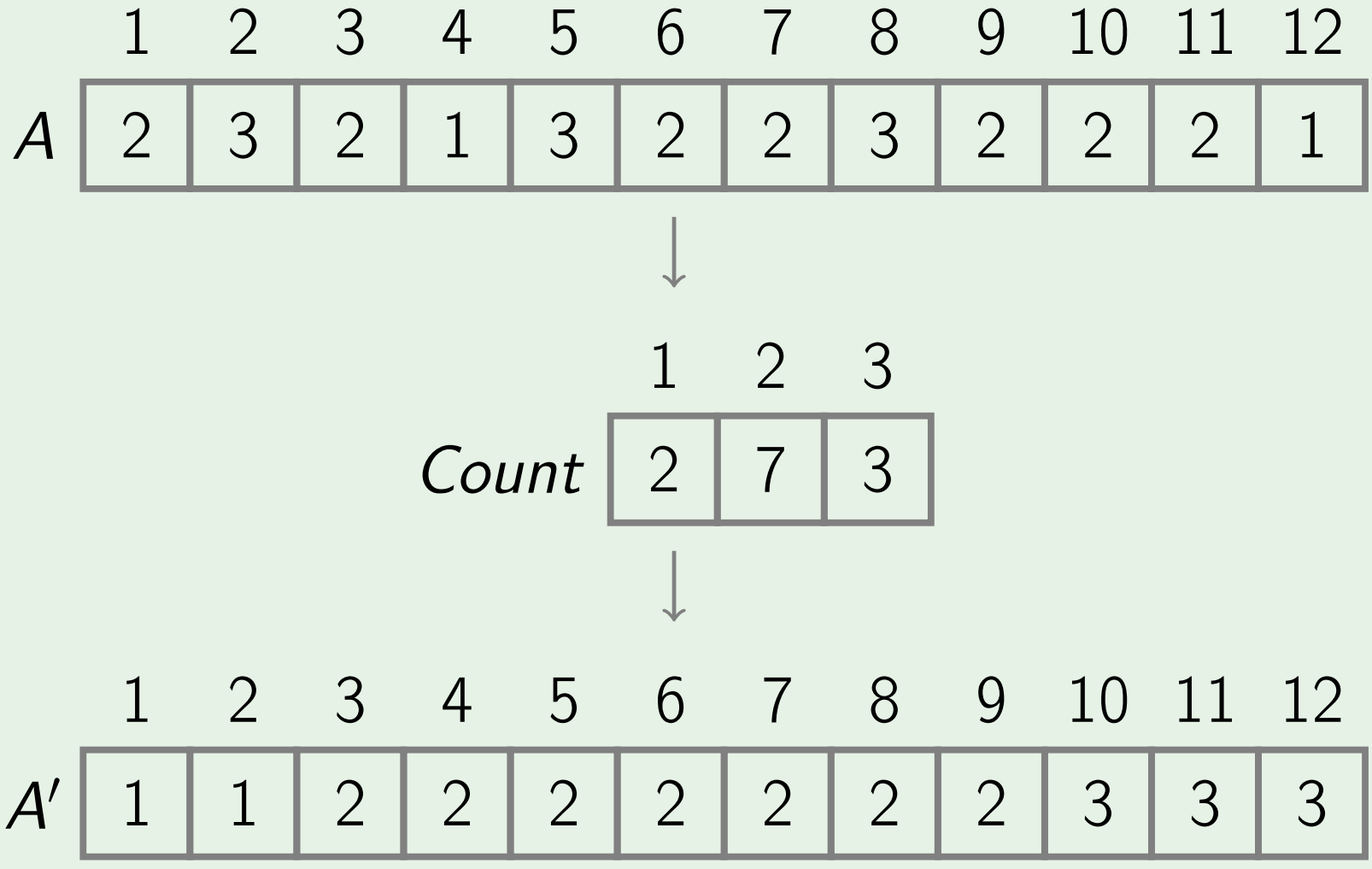
Proof

$$\begin{aligned}
\log_2(n!) &= \log_2(1 \cdot 2 \cdot \cdots \cdot n) \\
&= \log_2 1 + \log_2 2 + \cdots + \log_2 n \\
&\geq \log_2 \frac{n}{2} + \cdots + \log_2 n \\
&\geq \frac{n}{2} \log_2 \frac{n}{2} = \Omega(n \log n)
\end{aligned}$$

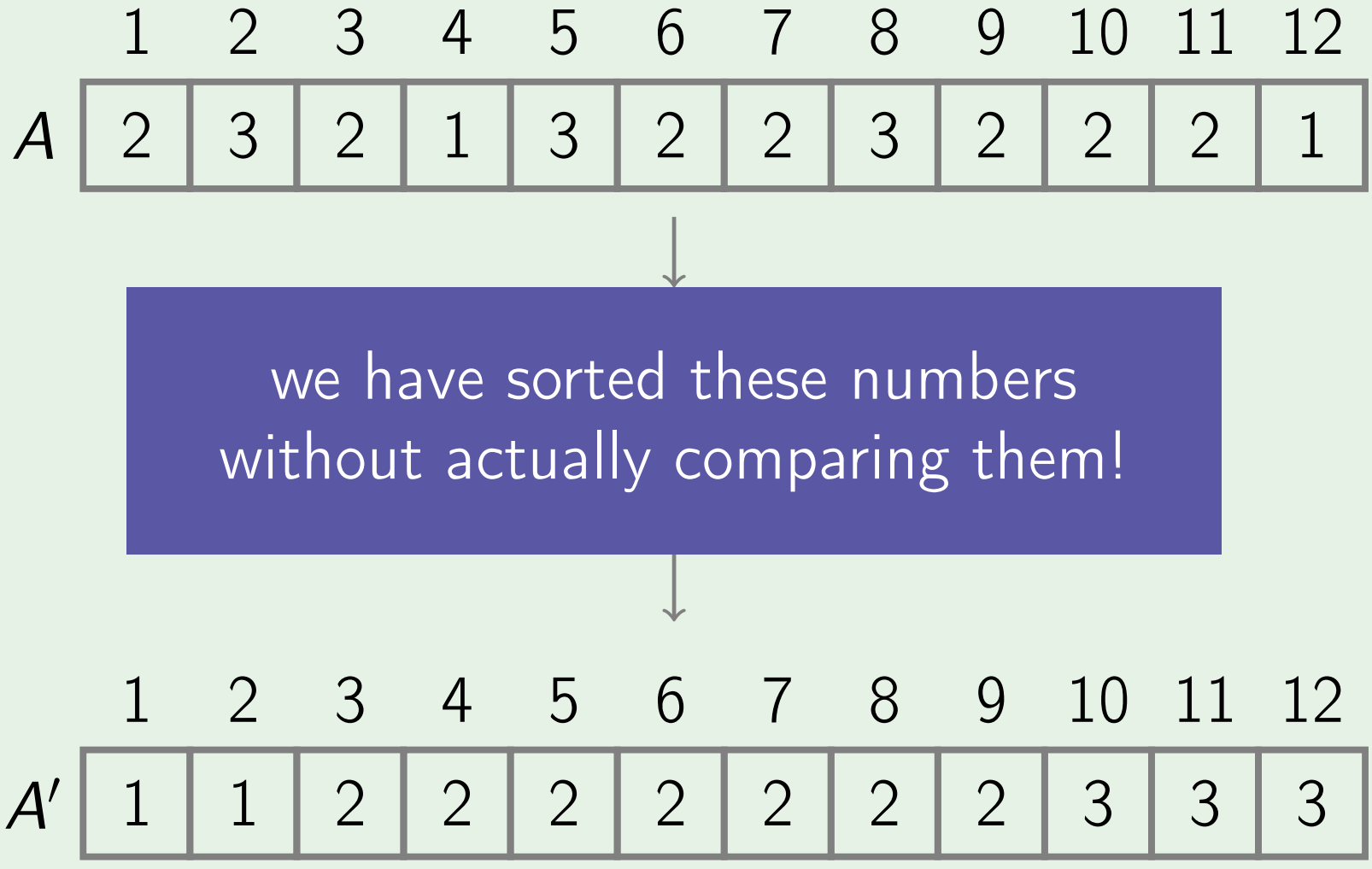
Outline

1. Problem Overview
2. Selection Sort
3. Merge Sort
4. Lower Bound for Comparison Based Sorting
5. Non-Comparison Based Sorting Algorithms

Example: sorting small integers



Example: sorting small integers



Counting Sort: Ideas

- Assume that all elements of $A[1 \ldots n]$ are integers from 1 to M.
- By a single scan of the array A, count the number of occurrences of each $1 \leq k \leq M$ in the array A and store it in $Count[k]$.
- Using this information, fill in the sorted array A'.

CountSort($A[1 \ldots n]$)

$Count[1 \ldots M] \leftarrow [0, \ldots, 0]$
`for` i `from 1 to` n`:`
  $Count[A[i]] \leftarrow Count[A[i]] + 1$
{k appears $Count[k]$ times in A}
$Pos[1 \ldots M] \leftarrow [0, \ldots, 0]$
$Pos[1] \leftarrow 1$
`for` j `from 2 to` M`:`
  $Pos[j] \leftarrow Pos[j-1] + Count[j-1]$
{k will occupy range $[Pos[k] \ldots Pos[k+1]-1]$}
`for` i `from 1 to` n`:`
  $A'[Pos[A[i]]] \leftarrow A[i]$
  $Pos[A[i]] \leftarrow Pos[A[i]] + 1$

Lemma

Provided that all elements of $A[1 \ldots n]$ are integers from 1 to M, $\texttt{CountSort}(A)$ sorts A in time $O(n + M)$.

Remark

If $M = O(n)$, then the running time is $O(n)$.

Summary 

- Merge sort uses the divide-and-conquer strategy to sort an n-element array in time $O(n \log n)$.
- No comparison based algorithm can do this (asymptotically) faster.
- One can do faster if something is known about the input array in advance (e.g., it contains small integers).

Divide-and-Conquer: Quick Sort

Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Algorithmic Toolbox
Data Structures and Algorithms

Outline

1. Overview
2. Algorithm
3. Random Pivot
4. Running Time Analysis
5. Equal Elements
6. Final Remarks

Quick Sort

- comparison based algorithm
- running time: $O(n \log n)$ (on average)
- efficient in practice

Example: quick sort

6	4	8	2	9	3	9	4	7	6	1

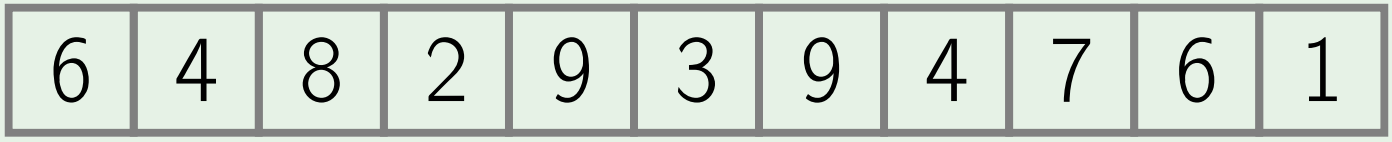
partition with respect to $x = A[1]$

in particular, x is in its final position

1	4	2	3	4	6	6	9	7	8	9

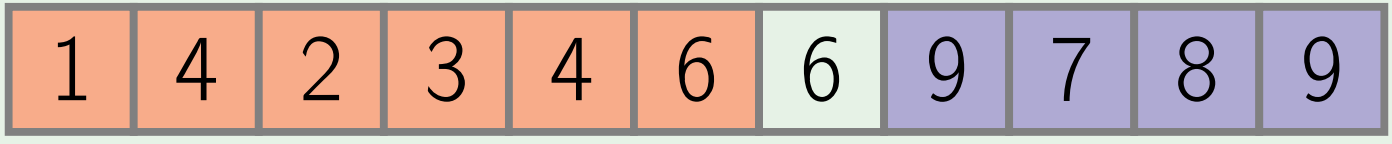
≤ 6 $\qquad\qquad$ > 6

Example: quick sort

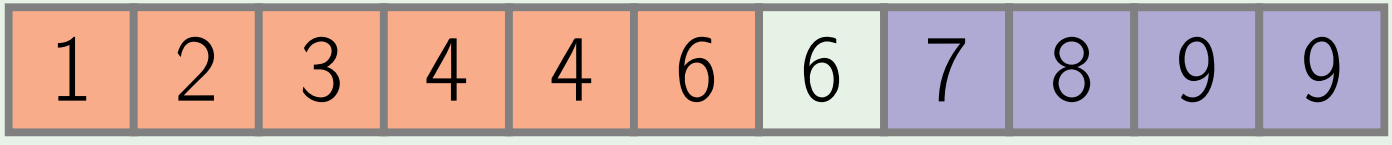


partition with respect to $x = A[1]$

in particular, x is in its final position



sort the two parts recursively



Outline

1. Overview
2. Algorithm
3. Random Pivot
4. Running Time Analysis
5. Equal Elements
6. Final Remarks

QuickSort(A, ℓ, r)

if $\ell \geq r$:

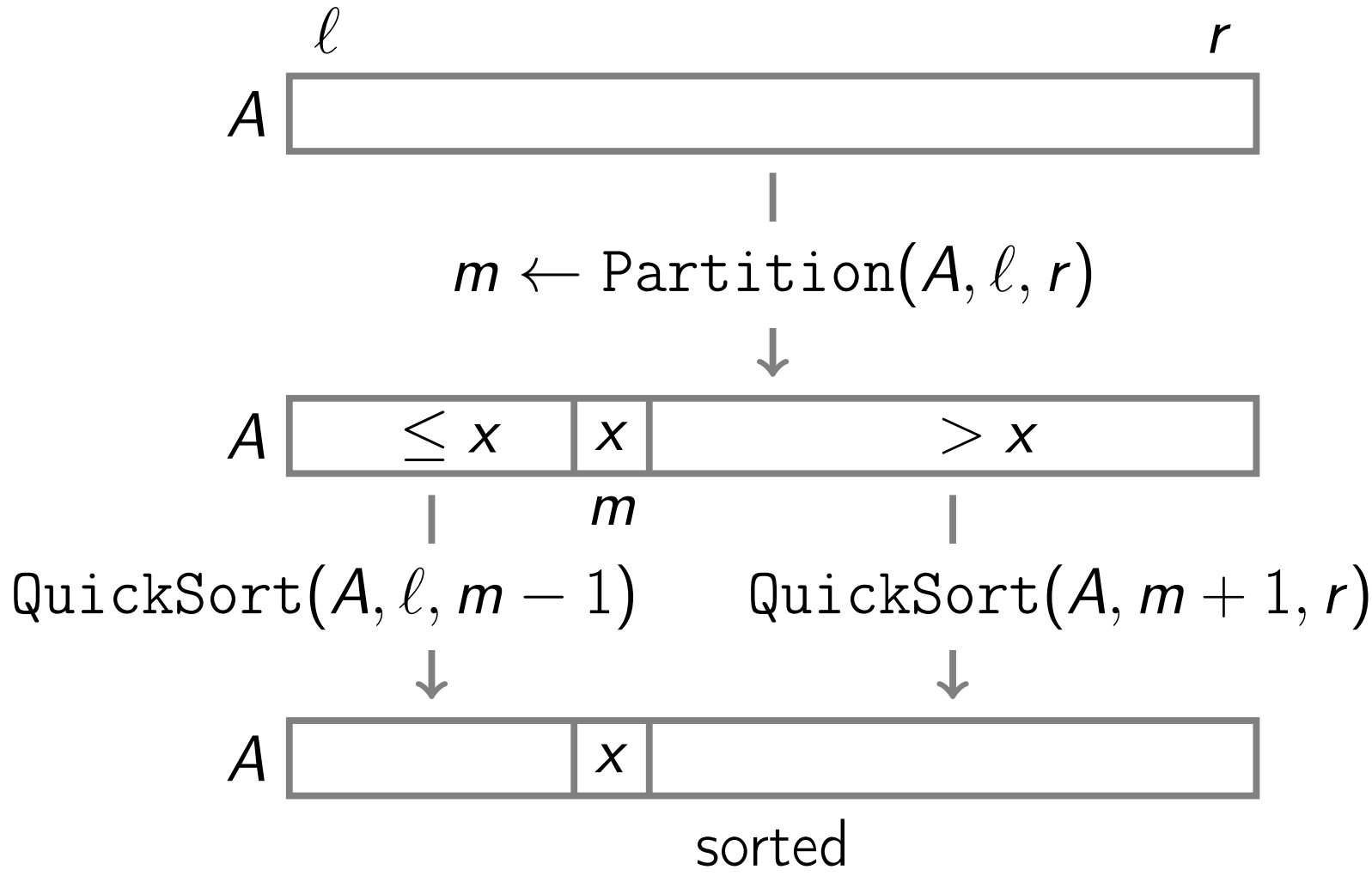
  return

$m \leftarrow$ Partition(A, ℓ, r)

{$A[m]$ is in the final position}

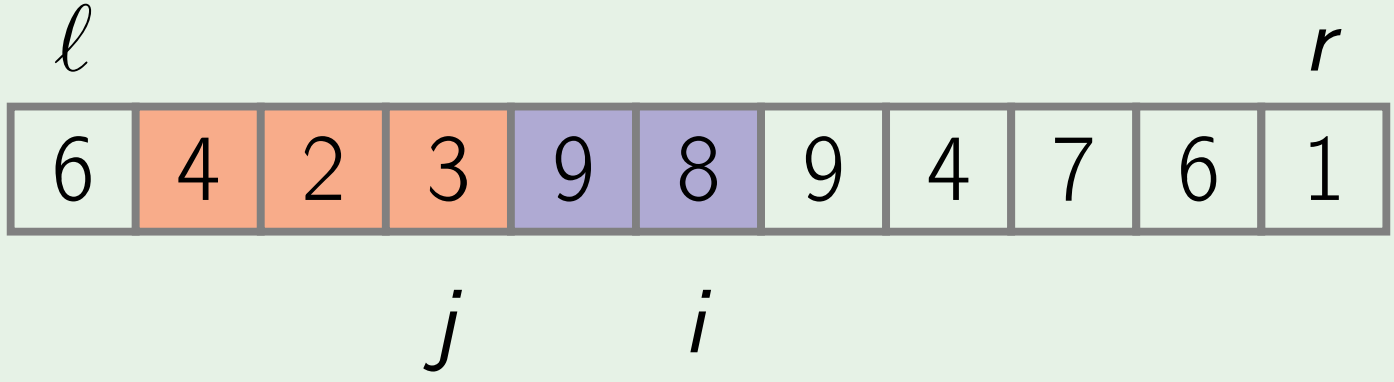
QuickSort$(A, \ell, m-1)$

QuickSort$(A, m+1, r)$


ℓ
r
A
$m \leftarrow \texttt{Partition}(A, \ell, r)$
A
$\leq x$
x
$> x$
m
$\texttt{QuickSort}(A, \ell, m-1)$
$\texttt{QuickSort}(A, m+1, r)$
A
x
sorted

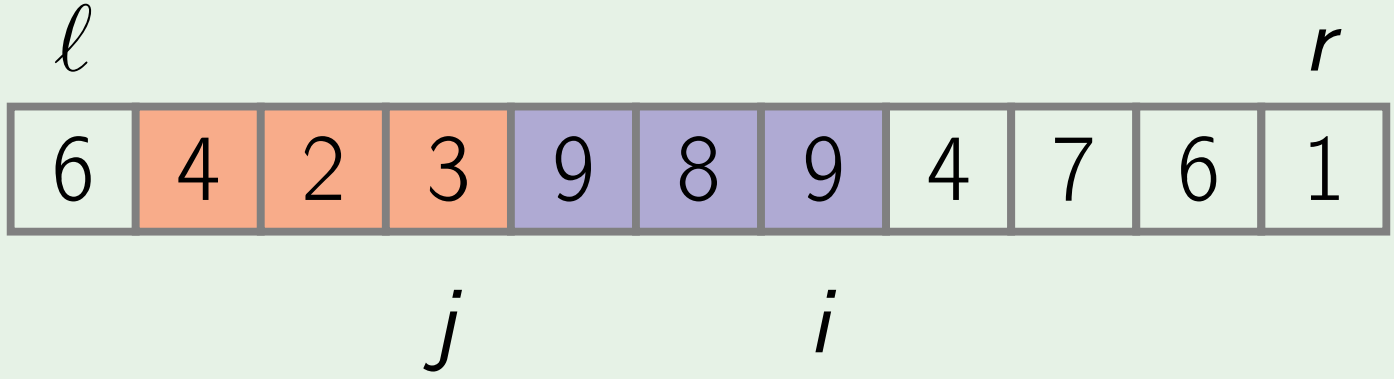
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

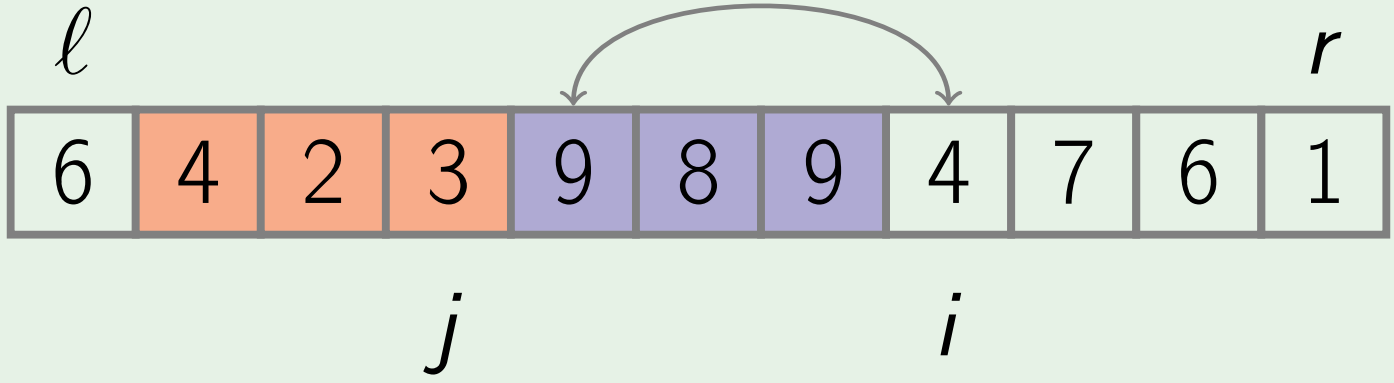


Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$

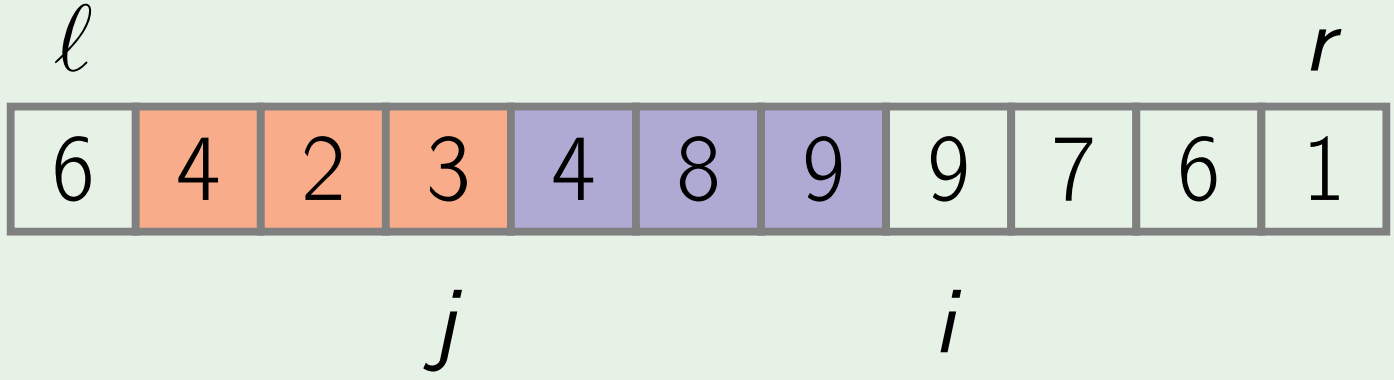
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



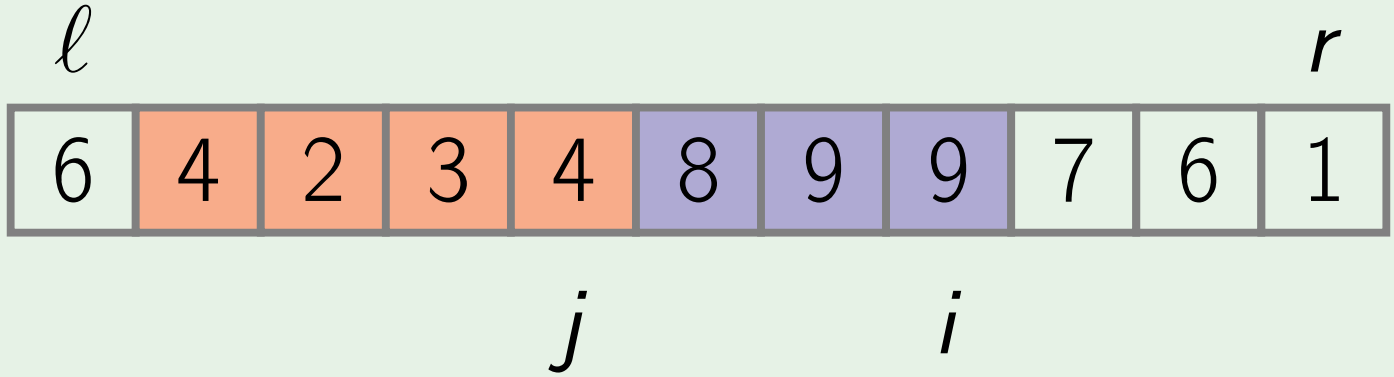
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



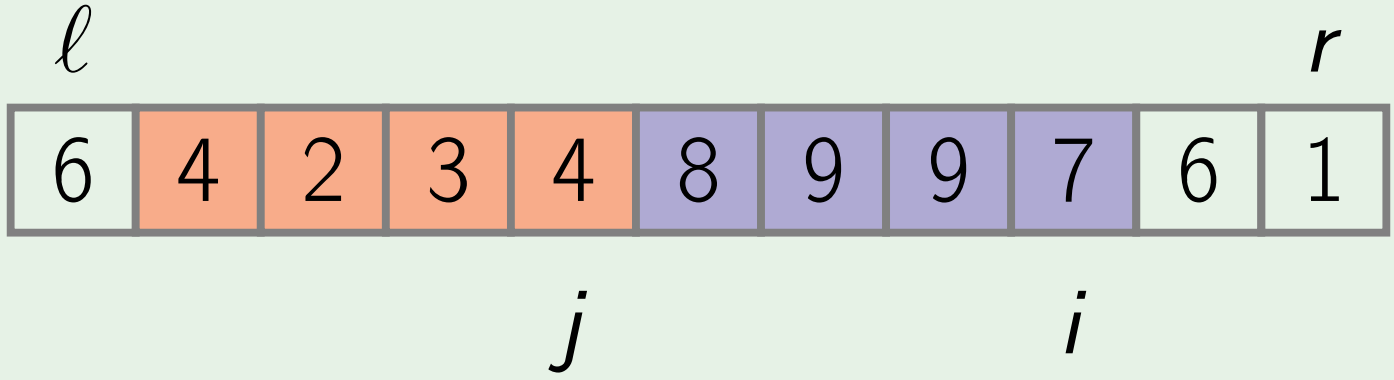
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



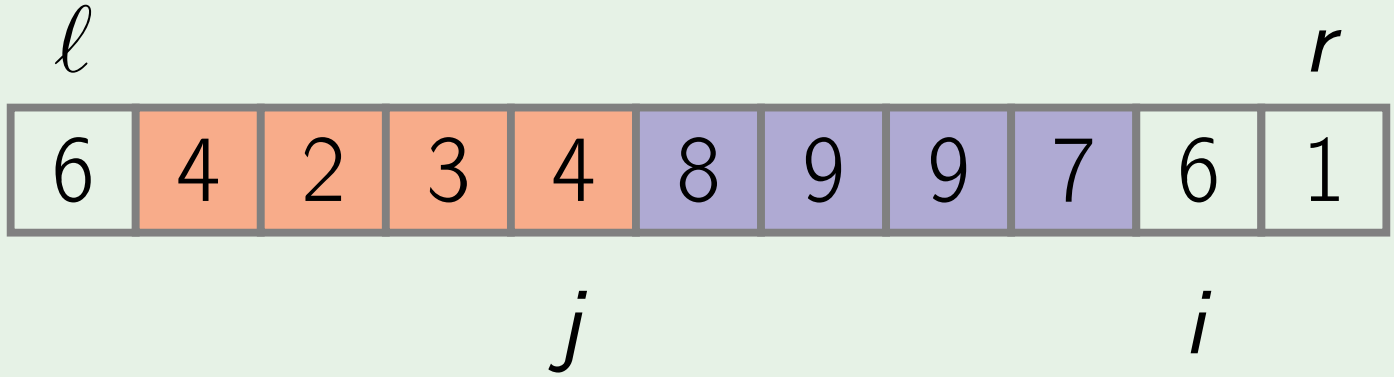
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



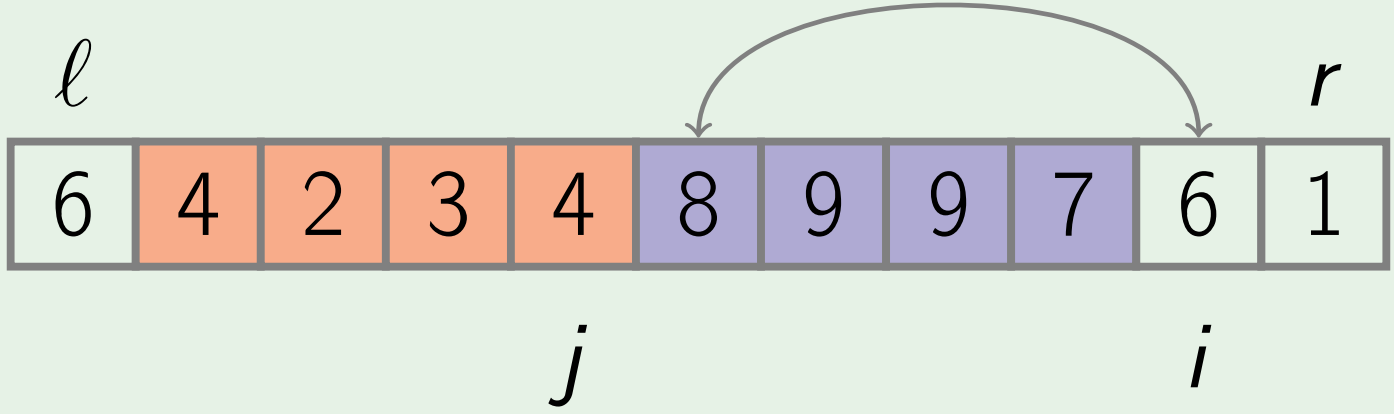
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



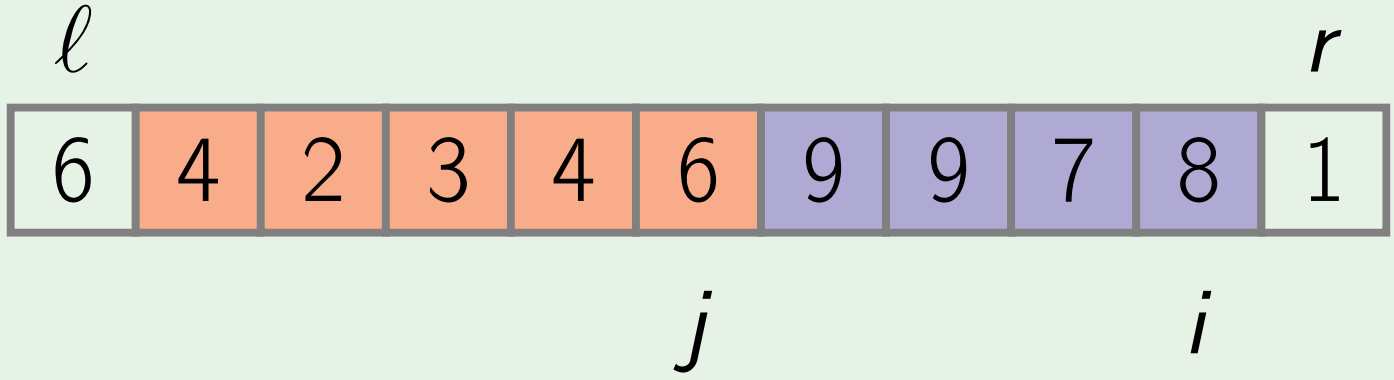
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



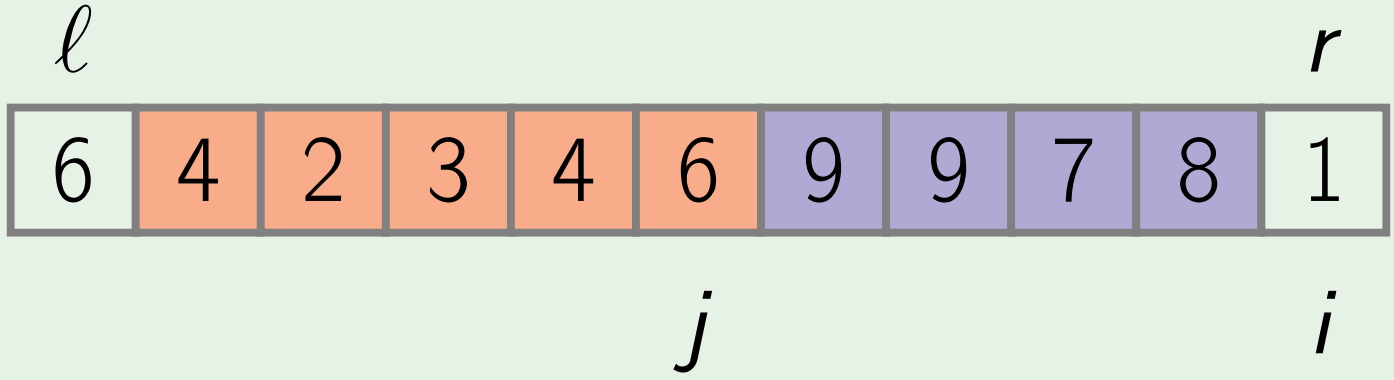
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



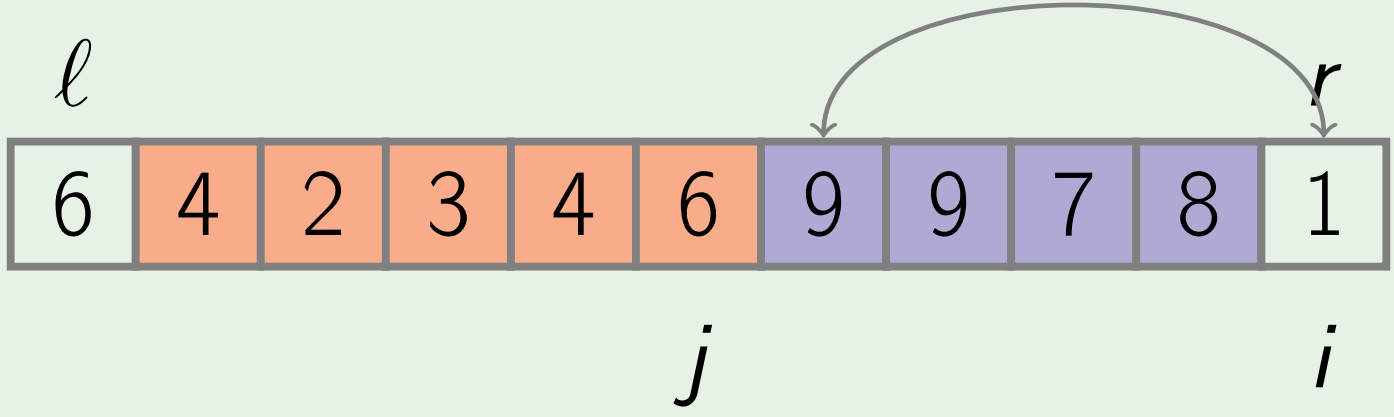
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



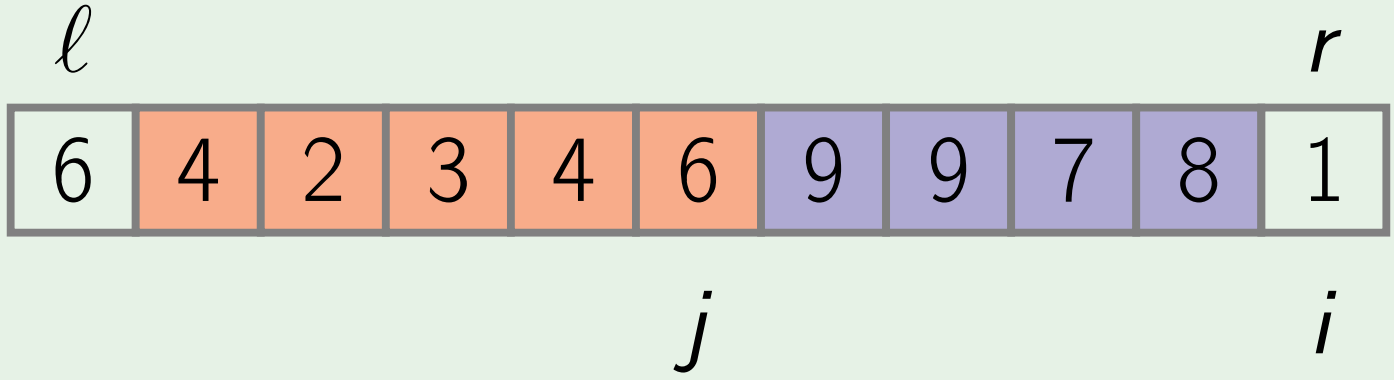
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



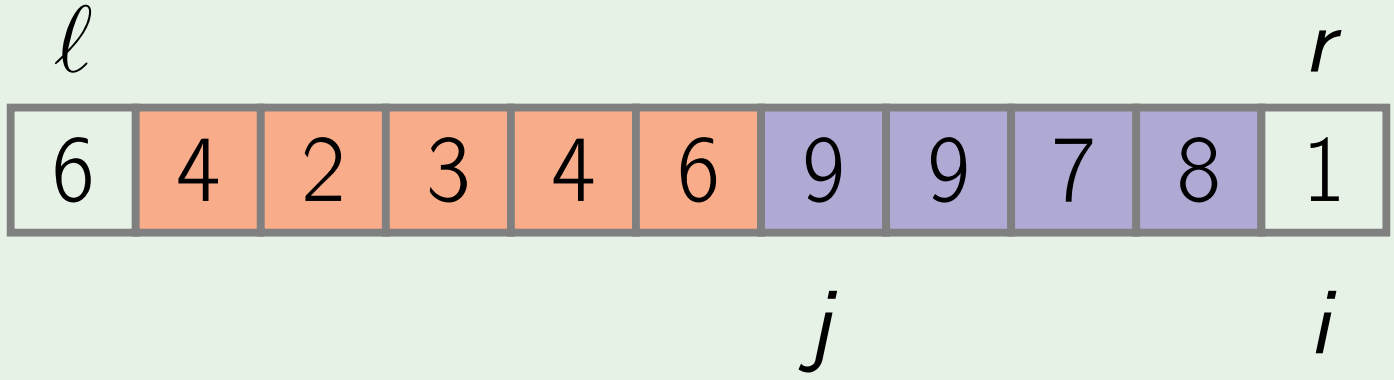
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



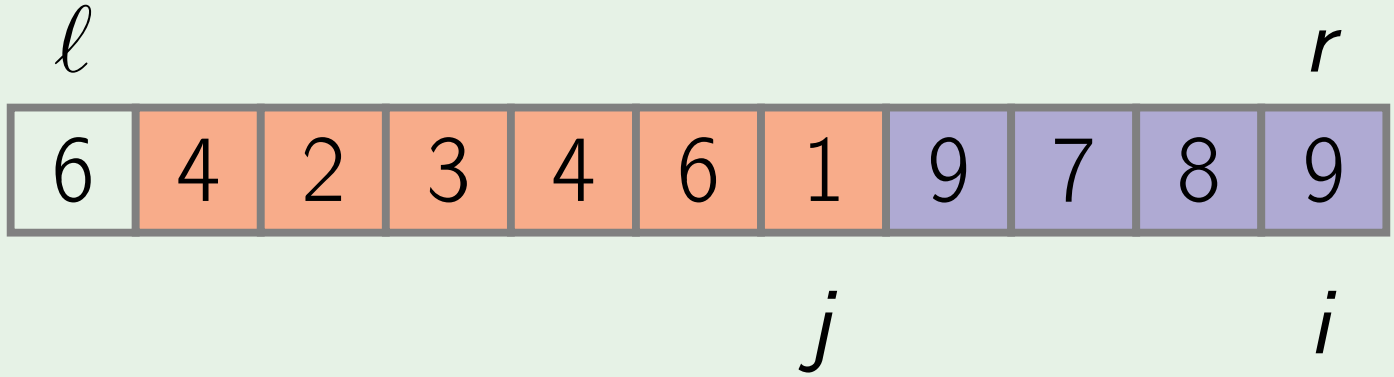
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$



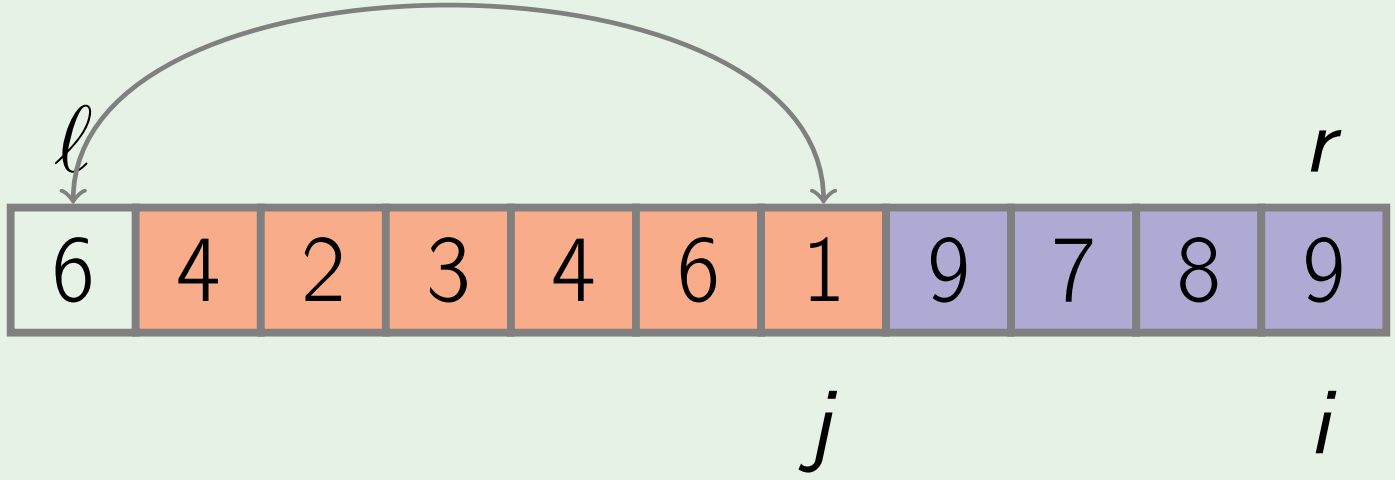
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$
- in the end, move $A[\ell]$ to its final place



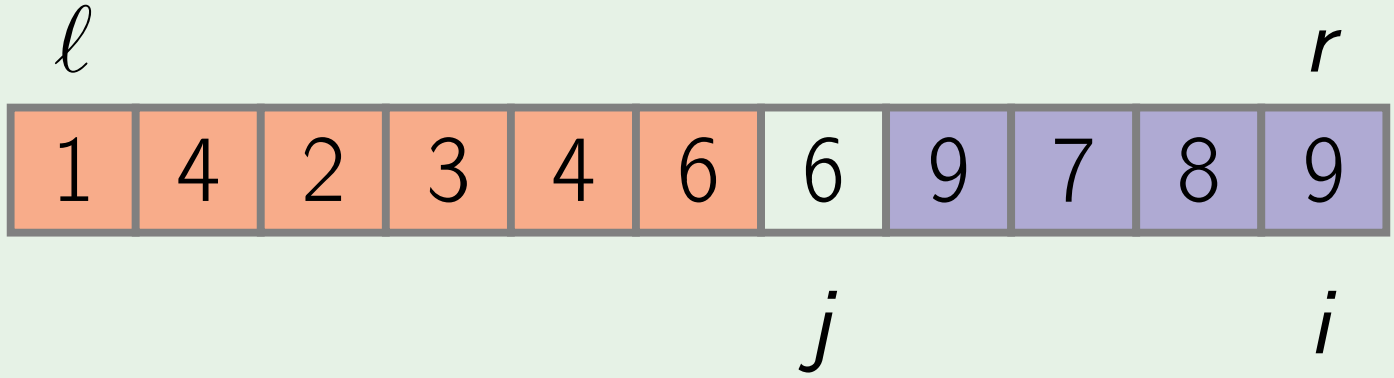
Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$
- in the end, move $A[\ell]$ to its final place



Partitioning: example

- the pivot is $x = A[\ell]$
- move i from $\ell + 1$ to r maintaining the following invariant:
  - $A[k] \leq x$ for all $\ell + 1 \leq k \leq j$
  - $A[k] > x$ for all $j + 1 \leq k \leq i$
- in the end, move $A[\ell]$ to its final place



Partition(A, ℓ, r)

$x \leftarrow A[\ell]$ {pivot}

$j \leftarrow \ell$

for i from $\ell + 1$ to r:

  if $A[i] \leq x$:

    $j \leftarrow j + 1$

    swap $A[j]$ and $A[i]$

  $\{A[\ell + 1 \ldots j] \leq x,\ A[j + 1 \ldots i] > x\}$

swap $A[\ell]$ and $A[j]$

return j

Outline

1. Overview
2. Algorithm
3. Random Pivot
4. Running Time Analysis
5. Equal Elements
6. Final Remarks

Unbalanced Partitions

- $T(n) = n + T(n-1)$:

  $T(n) = n+(n-1)+(n-2)+\cdots = \Theta(n^2)$

- $T(n) = n + T(n-5) + T(4)$:

  $T(n) \geq n+(n-5)+(n-10)+\cdots = \Theta(n^2)$

Balanced Partitions

- $T(n) = 2T(n/2) + n$:

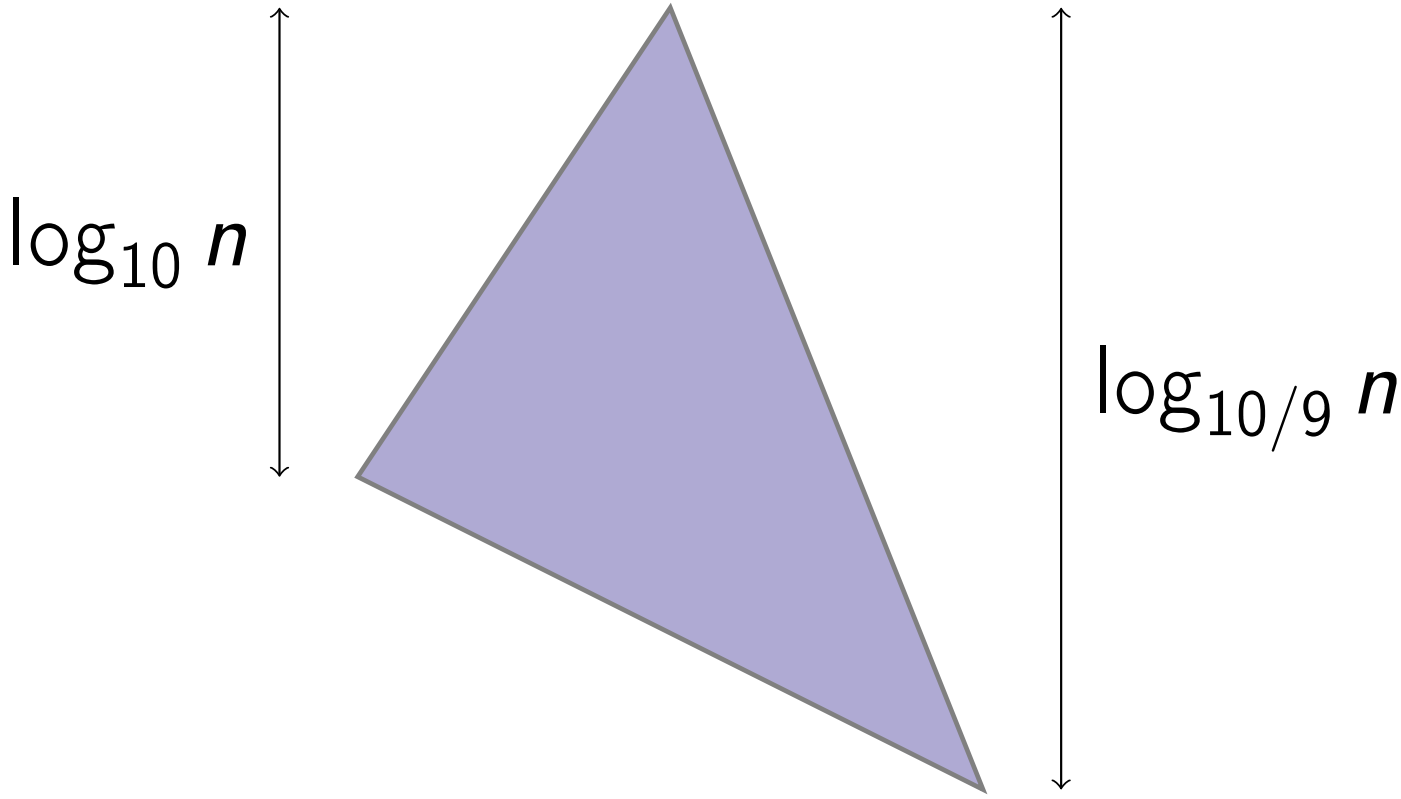
  $$T(n) = \Theta(n \log n)$$

- $T(n) = T(n/10) + T(9n/10) + n$:

  $$T(n) = \Theta(n \log n)$$

Balanced Partitions

$$T(n) = T(n/10) + T(9n/10) + O(n)$$



$$T(n) = O(n \log n)$$

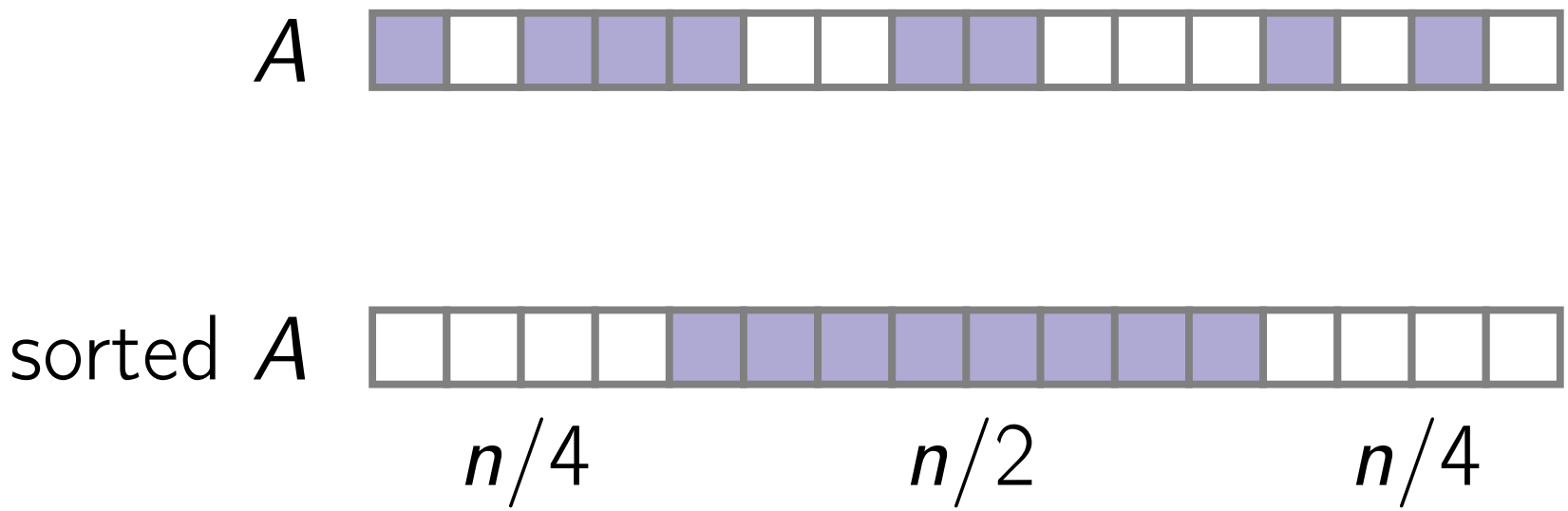
Random Pivot

RandomizedQuickSort(A, ℓ, r)

```
if ℓ ≥ r:
   return
k ← random number between ℓ and r
swap A[ℓ] and A[k]
m ← Partition(A, ℓ, r)
{A[m] is in the final position}
RandomizedQuickSort(A, ℓ, m − 1)
RandomizedQuickSort(A, m + 1, r)
```

Why Random?

half of the elements of A guarantees a balanced partition:



Theorem

Assume that all the elements of $A[1 \ldots n]$ are pairwise different. Then the average running time of `RandomizedQuickSort`(A) is $O(n \log n)$ while the worst case running time is $O(n^2)$.

Remark

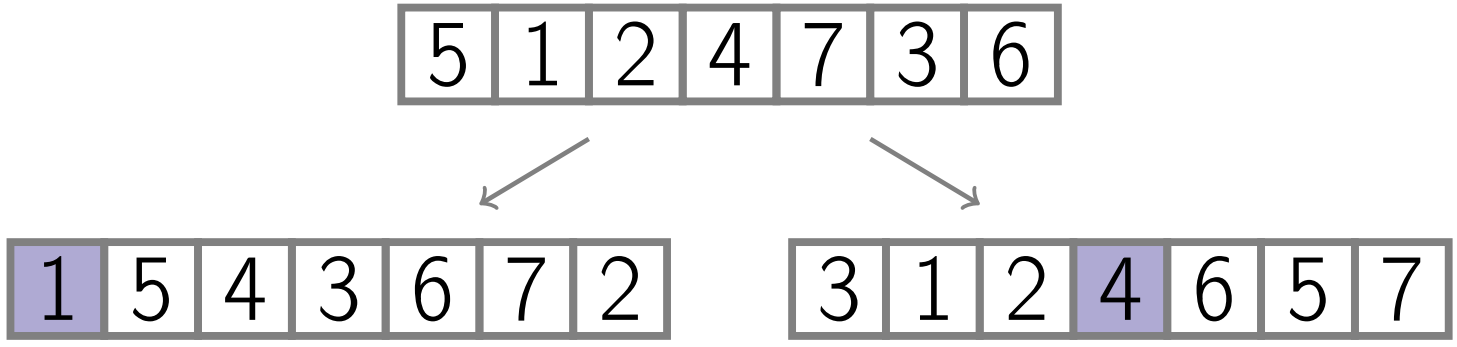
Averaging is over random numbers used by the algorithm, but not over the inputs.

Outline

1. Overview
2. Algorithm
3. Random Pivot
4. Running Time Analysis
5. Equal Elements
6. Final Remarks

Proof Ideas: Comparisons

- the running time is proportional to the number of comparisons made
- balanced partition are better since they reduce the number of comparisons needed:



Proof Ideas: Probability

A	5	1	8	9	2	4	7	3	6
A'	1	2	3	4	5	6	7	8	9

$$\text{Prob}\,(1 \text{ and } 9 \text{ are compared}) = \frac{2}{9}$$

$$\text{Prob}\,(3 \text{ and } 4 \text{ are compared}) = 1$$

Proof

- let, for $i < j$,

$$\chi_{ij} = \begin{cases} 1 & A'[i] \text{ and } A'[j] \text{ are compared} \\ 0 & \text{otherwise} \end{cases}$$

- for all $i < j$, $A'[i]$ and $A'[j]$ are either compared exactly once or not compared at all (as we compare with a pivot)
- this, in particular, implies that the worst case running time is $O(n^2)$

Proof (continued)

- crucial observation: $\chi_{ij} = 1$ iff the first selected pivot in $A'[i \ldots j]$ is $A'[i]$ or $A'[j]$
- then $\text{Prob}(\chi_{ij}) = \frac{2}{j-i+1}$ and $\text{E}(\chi_{ij}) = \frac{2}{j-i+1}$

Proof (continued)

Then (the expected value of) the running time is

$$\begin{aligned}
\mathrm{E}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\chi_{ij} &= \sum_{i=1}^{n}\sum_{j=i+1}^{n}\mathrm{E}(\chi_{ij}) \\
&= \sum_{i<j}\frac{2}{j-i+1} \\
&\leq 2n\cdot\left(\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}\right) \\
&= \Theta(n\log n)
\end{aligned}$$

Outline

1. Overview
2. Algorithm
3. Random Pivot
4. Running Time Analysis
5. Equal Elements
6. Final Remarks

Equal Elements

- what if all the elements of the given array are equal to each other?
- quick sort visualization
- the array is always split into two parts of size 0 and $n - 1$
- $T(n) = n + T(n-1) + T(0)$ and hence $T(n) = \Theta(n^2)$!

To handle equal elements, we replace the line

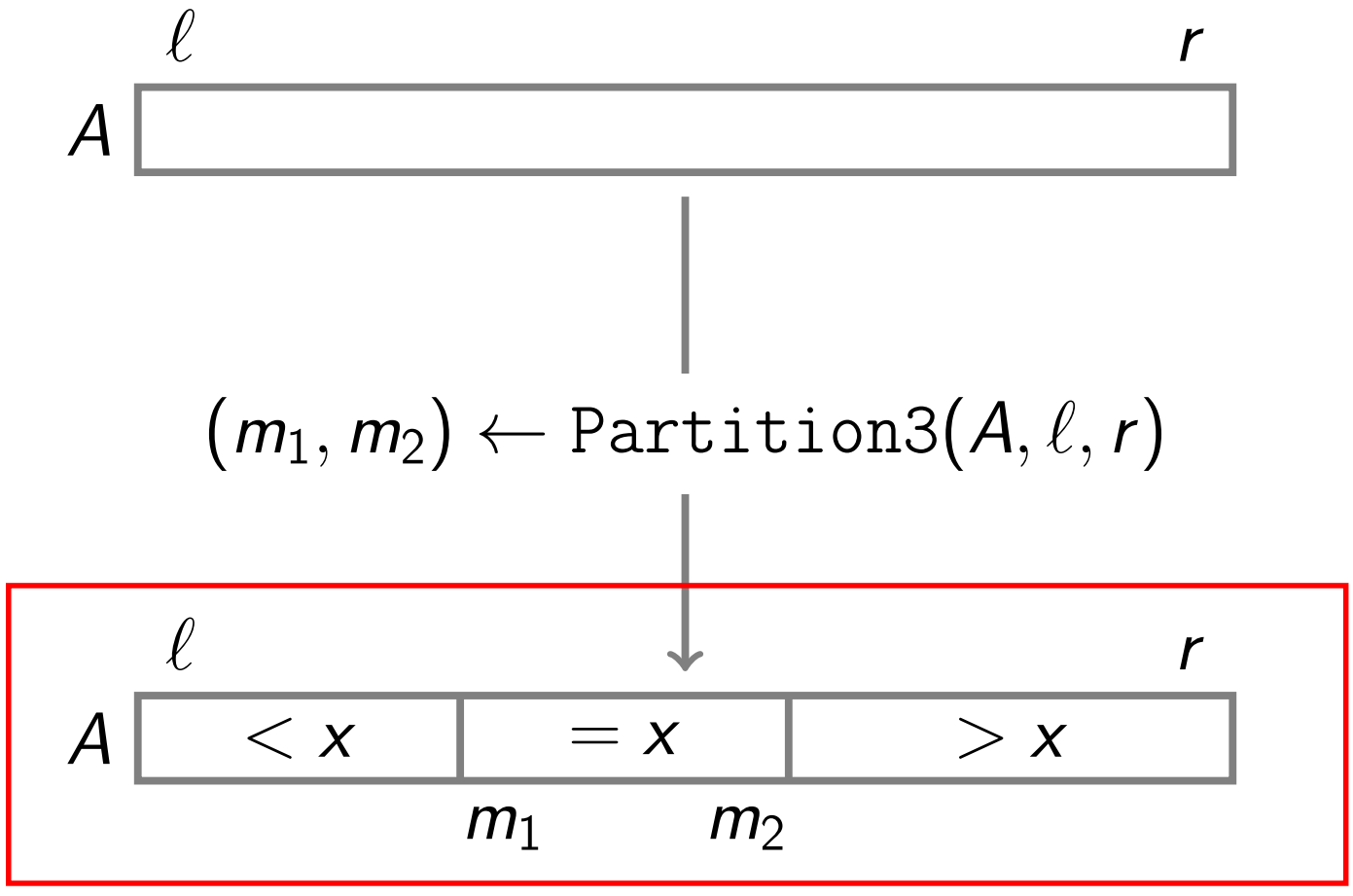
$$m \leftarrow \texttt{Partition}(A, \ell, r)$$

with the line

$$(m_1, m_2) \leftarrow \texttt{Partition3}(A, \ell, r)$$

such that

- for all $\ell \leq k \leq m_1 - 1$, $A[k] < x$
- for all $m_1 \leq k \leq m_2$, $A[k] = x$
- for all $m_2 + 1 \leq k \leq r$, $A[k] > x$


ℓ
r
A
$(m_1, m_2) \leftarrow \texttt{Partition3}(A, \ell, r)$
ℓ
r
A
$< x$
$= x$
$> x$
m_1
m_2

RandomizedQuickSort(A, ℓ, r)

if $\ell \geq r$:

  return

$k \leftarrow$ random number between ℓ and r

swap $A[\ell]$ and $A[k]$

$(m_1, m_2) \leftarrow$ Partition3(A, ℓ, r)

{$A[m_1 \ldots m_2]$ is in final position}

RandomizedQuickSort$(A, \ell, m_1 - 1)$

RandomizedQuickSort$(A, m_2 + 1, r)$

Outline

1. Overview
2. Algorithm
3. Random Pivot
4. Running Time Analysis
5. Equal Elements
6. Final Remarks

Tail Recursion Elimination

QuickSort(A, ℓ, r)

```
while ℓ < r:
    m ← Partition(A, ℓ, r)
    QuickSort(A, ℓ, m − 1)
    ℓ ← m + 1
```

QuickSort(A, ℓ, r)

```
while ℓ < r:
    m ← Partition(A, ℓ, r)
    if (m − ℓ) < (r − m):
        QuickSort(A, ℓ, m − 1)
        ℓ ← m + 1
    else:
        QuickSort(A, m + 1, r)
        r ← m − 1
```

Worst-case space requirement: $O(\log n)$

Intro Sort

- runs quick sort with a simple deterministic pivot selection heuristic (say, median of the first, middle, and last element)
- if the recursion depth exceeds a certain threshold $c \log n$ the algorithm switches to heap sort
- the running time is $O(n \log n)$ in the worst case

Conclusion 

- Quick sort is a comparison based algorithm
- Running time: $O(n \log n)$ on average, $O(n^2)$ in the worst case
- Efficient in practice