Lecture 3

Divide and Conquer I: Introduction, Merge-sort and Master Theorem

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

Recursion

- Definition:
  - Solving a task where the solution depends on solutions to smaller instances of the same problem, by using functions/algorithms that call themselves.

Recursion

- Definition:
  - Solving a task where the solution depends on solutions to smaller instances of the same problem, by using functions/algorithms that call themselves.

- Example:

$$\text{Factorial}(n)$$

$$\textbf{If } n == 0 \textbf{ then return } 1$$

$$\textbf{return } n \cdot \text{Factorial}(n-1)$$

Recursion

- Definition:
  – Solving a task where the solution depends on solutions to smaller instances of the same problem, by using functions/algorithms that call themselves.

- Example:

$$\text{Factorial}(n)$$

$$\textbf{If } n == 0 \textbf{ then return } 1$$

Base case

$$\textbf{return } n \cdot \text{Factorial}(n-1)$$

Recursion

Recursion

- Definition:
  – Solving a task where the solution depends on solutions to smaller instances of the same problem, by using functions/algorithms that call themselves.

Running time: $T(n) = T(n-1) + \Theta(1)$

- Example:

$$\text{Factorial}(n)$$

$$\textbf{If } n == 0 \textbf{ then return } 1$$

Base case

$$\textbf{return } n \cdot \text{Factorial}(n-1)$$

Recursion

Recursion

Exercise: Let $T(n) = T(n-1) + 1$, with $T(1) = 1$.

Find $T(n)$.

Recursion

Exercise: Let $T(n) = T(n-1) + 1$, with $T(1) = 1$.

Find $T(n)$.

Solution:

Since $T(n-1) = T(n-2) + 1$, by substitution we have

$$T(n) = T(n-2) + 2.$$

Recursion

Exercise: Let $T(n) = T(n-1) + 1$, with $T(1) = 1$.

Find $T(n)$.

Solution:

Since $T(n-1) = T(n-2) + 1$, by substitution we have

$$T(n) = T(n-2) + 2.$$

Since $T(n-2) = T(n-3) + 1$, by substitution we have

$$T(n) = T(n-3) + 3.$$

Recursion

Exercise: Let $T(n) = T(n-1) + 1$, with $T(1) = 1$.

Find $T(n)$.

Solution:

Since $T(n-1) = T(n-2) + 1$, by substitution we have

$$T(n) = T(n-2) + 2.$$

Since $T(n-2) = T(n-3) + 1$, by substitution we have

$$T(n) = T(n-3) + 3.$$

$$\vdots$$

Since $T(n-i) = T(n-i-1) + 1$, by substitution we have

$$T(n) = T(n-i) + i.$$

Recursion

Exercise: Let $T(n) = T(n-1) + 1$, with $T(1) = 1$.

Find $T(n)$.

Solution:

Since $T(n-i) = T(n-i-1) + 1$, by substitution we have

$$T(n) = T(n-i) + i.$$

By setting $i = n - 1$ we have

$$T(n) = T(1) + n - 1 = n \text{ which is } \Theta(n).$$

Mergesort - A fast sorting recursive Algorithm

Have seen algorithms like insertion sort that have running time (worst case) $\Theta(n^2)$.

- Key idea:

  Divide input into two parts of equal size

  Sort each part recursively

  Merge the two sorted parts to obtain the solution.

Mergesort - A fast sorting recursive Algorithm

- Key idea:

  Divide input into two parts of equal size

  Sort each part recursively

  Merge the two sorted parts to obtain the solution.

Example: Sort the following 11 numbers

9 3 4 220 1 3 10 5 8 7 2

Divide

Recursion

Merge

Mergesort - A fast sorting recursive Algorithm

- Key idea:
  Divide input into two parts of equal size
  Sort each part recursively
  Merge the two sorted parts to obtain the solution.

Example: Sort the following 11 numbers

9 3 4 220 1 3 10 5 8 7 2

9 3 4 220 1 3 10 5 8 7 2

Divide

Recursion

Merge

Mergesort - A fast sorting recursive Algorithm

- Key idea:

  Divide input into two parts of equal size

  Sort each part recursively

  Merge the two sorted parts to obtain the solution.

Example: Sort the following 11 numbers

9 3 4 220 1 3 10 5 8 7 2

9 3 4 220 1   3 10 5 8 7 2   **Divide**

1 3 4 9 220   2 3 5 7 8 10   **Recursion**

Merge

Mergesort - A fast sorting recursive Algorithm

- Key idea:
  Divide input into two parts of equal size
  Sort each part recursively
  Merge the two sorted parts to obtain the solution.

Example: Sort the following 11 numbers

9 3 4 220 1 3 10 5 8 7 2

9 3 4 220 1   3 10 5 8 7 2   Divide

1 3 4 9 220   2 3 5 7 8 10   Recursion

1 2 3 3 4 5 7 8 9 10 220   Merge

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$\begin{array}{l}
\textbf{While } k < \mathrm{len(A)} + \mathrm{len(B)} \textbf{ do} \\
\quad \textbf{If } A[i] <= B[j] \textbf{ then} \\
\quad\quad C[k] \leftarrow A[i] \\
\quad\quad i = i + 1, k = k + 1 \\
\quad \textbf{else} \\
\quad\quad C[k] \leftarrow B[j] \\
\quad\quad j = j + 1, k = k + 1
\end{array}$$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$\begin{aligned}
&\textbf{While } k < \text{len(A)} + \text{len(B)} \textbf{ do}\\
&\quad \textbf{If } A[i] <= B[j] \textbf{ then}\\
&\qquad C[k] \leftarrow A[i]\\
&\qquad i = i + 1, k = k + 1\\
&\quad \textbf{else}\\
&\qquad C[k] \leftarrow B[j]\\
&\qquad j = j + 1, k = k + 1
\end{aligned}$$

$A: 1\ 3\ 4\ 9\ 220$ $\quad$ $B: 2\ 3\ 5\ 7\ 8\ 10$

i $\quad$ j

$C:$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

While $k < \text{len(A)} + \text{len(B)}$ **do**

  If $A[i] <= B[j]$ **then**

    $C[k] \leftarrow A[i]$

    $i = i + 1, k = k + 1$

  else

    $C[k] \leftarrow B[j]$

    $j = j + 1, k = k + 1$

$A : 1\ 3\ 4\ 9\ 220$ $B : 2\ 3\ 5\ 7\ 8\ 10$

i j

$C : 1$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

While $k < \text{len(A)} + \text{len(B)}$ **do**

  If $A[i] <= B[j]$ **then**

    $C[k] \leftarrow A[i]$

    $i = i + 1, k = k + 1$

  else

    $C[k] \leftarrow B[j]$

    $j = j + 1, k = k + 1$

$A : 1\ 3\ 4\ 9\ 220$ $\qquad$ $B : 2\ 3\ 5\ 7\ 8\ 10$

i $\qquad$ j

$C : 1\ 2$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

While $k < \text{len(A)} + \text{len(B)}$ **do**

    If $A[i] <= B[j]$ **then**

        $C[k] \leftarrow A[i]$

        $i = i + 1, k = k + 1$

    else

        $C[k] \leftarrow B[j]$

        $j = j + 1, k = k + 1$

$A : 1\ 3\ 4\ 9\ 220$ $\quad$ $B : 2\ 3\ 5\ 7\ 8\ 10$

i $\quad$ j

$C : 1\ 2\ 3$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

While $k < \text{len(A)} + \text{len(B)}$ **do**

  If $A[i] <= B[j]$ **then**

    $C[k] \leftarrow A[i]$

    $i = i + 1, k = k + 1$

  else

    $C[k] \leftarrow B[j]$

    $j = j + 1, k = k + 1$

$A : 1\ 3\ 4\ 9\ 220$ $B : 2\ 3\ 5\ 7\ 8\ 10$

i j

$C : 1\ 2\ 3\ 3$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$
\begin{array}{l}
\textbf{While } k < \text{len(A)} + \text{len(B)} \textbf{ do} \\
\quad \textbf{If } A[i] <= B[j] \textbf{ then} \\
\quad\quad C[k] \leftarrow A[i] \\
\quad\quad i = i+1, k = k+1 \\
\quad \textbf{else} \\
\quad\quad C[k] \leftarrow B[j] \\
\quad\quad j = j+1, k = k+1
\end{array}
$$

$A : 1\ 3\ 4\ 9\ 220$ $\quad$ $B : 2\ 3\ 5\ 7\ 8\ 10$

i $\quad$ j

$C : 1\ 2\ 3\ 3\ 4$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$\textbf{While } k < \mathrm{len(A)} + \mathrm{len(B)} \textbf{ do}$$
$$\quad \textbf{If } A[i] <= B[j] \textbf{ then}$$
$$\qquad C[k] \leftarrow A[i]$$
$$\qquad i = i + 1, k = k + 1$$
$$\quad \textbf{else}$$
$$\qquad C[k] \leftarrow B[j]$$
$$\qquad j = j + 1, k = k + 1$$

$A : 1\ 3\ 4\ 9\ 220$ $\quad B : 2\ 3\ 5\ 7\ 8\ 10$

i $\quad j$

$C : 1\ 2\ 3\ 3\ 4\ 5$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$\textbf{While } k < \text{len(A)} + \text{len(B)} \textbf{ do}$$
$$\quad \textbf{If } A[i] <= B[j] \textbf{ then}$$
$$\qquad C[k] \leftarrow A[i]$$
$$\qquad i = i + 1, k = k + 1$$
$$\quad \textbf{else}$$
$$\qquad C[k] \leftarrow B[j]$$
$$\qquad j = j + 1, k = k + 1$$

$A : 1\ 3\ 4\ 9\ 220$ $\quad$ $B : 2\ 3\ 5\ 7\ 8\ 10$

$\qquad\quad\ \ i \qquad\qquad\qquad\qquad\qquad j$

$C : 1\ 2\ 3\ 3\ 4\ 5\ 7$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

While $k < \mathrm{len(A)} + \mathrm{len(B)}$ **do**

  If $A[i] <= B[j]$ **then**

    $C[k] \leftarrow A[i]$

    $i = i + 1, k = k + 1$

  else

    $C[k] \leftarrow B[j]$

    $j = j + 1, k = k + 1$

$A: 1\ 3\ 4\ 9\ 220$ $B: 2\ 3\ 5\ 7\ 8\ 10$

i j

$C: 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$
\begin{aligned}
&\textbf{While } k < \mathrm{len(A)} + \mathrm{len(B)} \textbf{ do}\\
&\quad \textbf{If } A[i] <= B[j] \textbf{ then}\\
&\quad\quad C[k] \leftarrow A[i]\\
&\quad\quad i = i + 1, k = k + 1\\
&\quad \textbf{else}\\
&\quad\quad C[k] \leftarrow B[j]\\
&\quad\quad j = j + 1, k = k + 1
\end{aligned}
$$

A : 1 3 4 9 220 $\qquad$ B : 2 3 5 7 8 10

i $\qquad$ j

C : 1 2 3 3 4 5 7 8 9

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

While $k < \text{len(A)} + \text{len(B)}$ **do**

  If $A[i] <= B[j]$ **then**

    $C[k] \leftarrow A[i]$

    $i = i + 1, k = k + 1$

  else

    $C[k] \leftarrow B[j]$

    $j = j + 1, k = k + 1$

$A : 1\ 3\ 4\ 9\ 220$ $B : 2\ 3\ 5\ 7\ 8\ 10$

i j

$C : 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9\ 10$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$
\begin{aligned}
&\textbf{While } k < \text{len(A)} + \text{len(B)} \textbf{ do}\\
&\quad \textbf{If } A[i] <= B[j] \textbf{ then}\\
&\qquad C[k] \leftarrow A[i]\\
&\qquad i = i + 1, k = k + 1\\
&\quad \textbf{else}\\
&\qquad C[k] \leftarrow B[j]\\
&\qquad j = j + 1, k = k + 1
\end{aligned}
$$

$A: 1\ 3\ 4\ 9\ 220$ $\quad$ $B: 2\ 3\ 5\ 7\ 8\ 10$

i $\quad$ j

$C: 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9\ 10\ 220$

Mergesort - A fast sorting recursive Algorithm

- Tricky part: Merge

Problem: Given two sorted arrays A, B, merge them to a sorted array C.

Solution: Index i for A, index j for B, index k for C.

$$
\begin{array}{l}
\textbf{While } k < \text{len(A)} + \text{len(B)} \textbf{ do} \\
\quad \textbf{If } A[i] <= B[j] \textbf{ then} \\
\qquad C[k] \leftarrow A[i] \\
\qquad i = i + 1, k = k + 1 \\
\quad \textbf{else} \\
\qquad C[k] \leftarrow B[j] \\
\qquad j = j + 1, k = k + 1
\end{array}
$$

Running time: $\Theta(n)$

$A: 1\ 3\ 4\ 9\ 220$ $\quad$ $B: 2\ 3\ 5\ 7\ 8\ 10$

i $\quad$ j

$C: 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9\ 10\ 220$

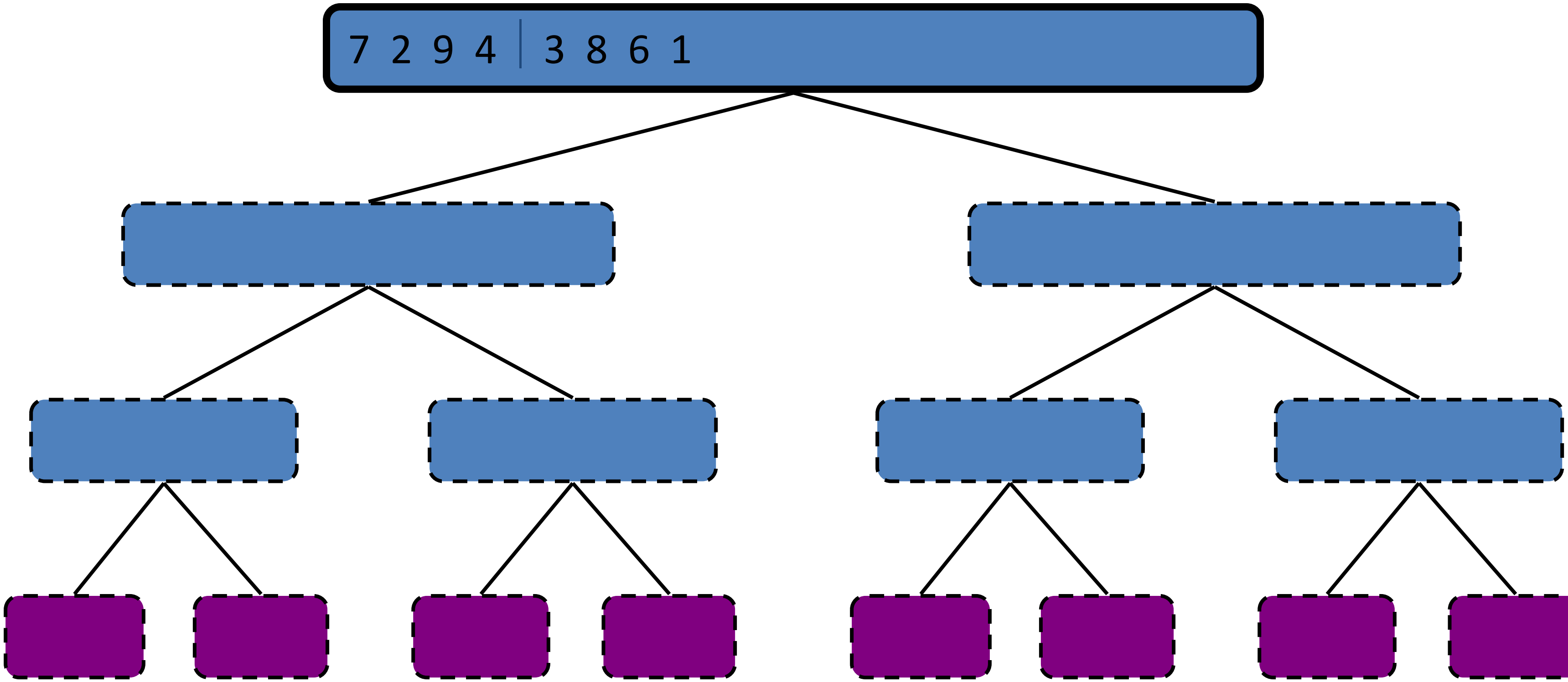
Mergesort

- Pseudocode:

$$
\begin{aligned}
&\text{Mergesort}(A[1:n])\\
&\quad \textbf{If } n == 1 \textbf{ then}\\
&\qquad \textbf{return } A\\
&\quad \text{Mergesort }(A[1:\tfrac{n}{2}])\\
&\quad \text{Mergesort }(A[\tfrac{n}{2}+1:n])\\
&\quad C \leftarrow \text{Merge}(A[1:\tfrac{n}{2}], A[\tfrac{n}{2}+1:n])\\
&\quad \textbf{return } C
\end{aligned}
$$

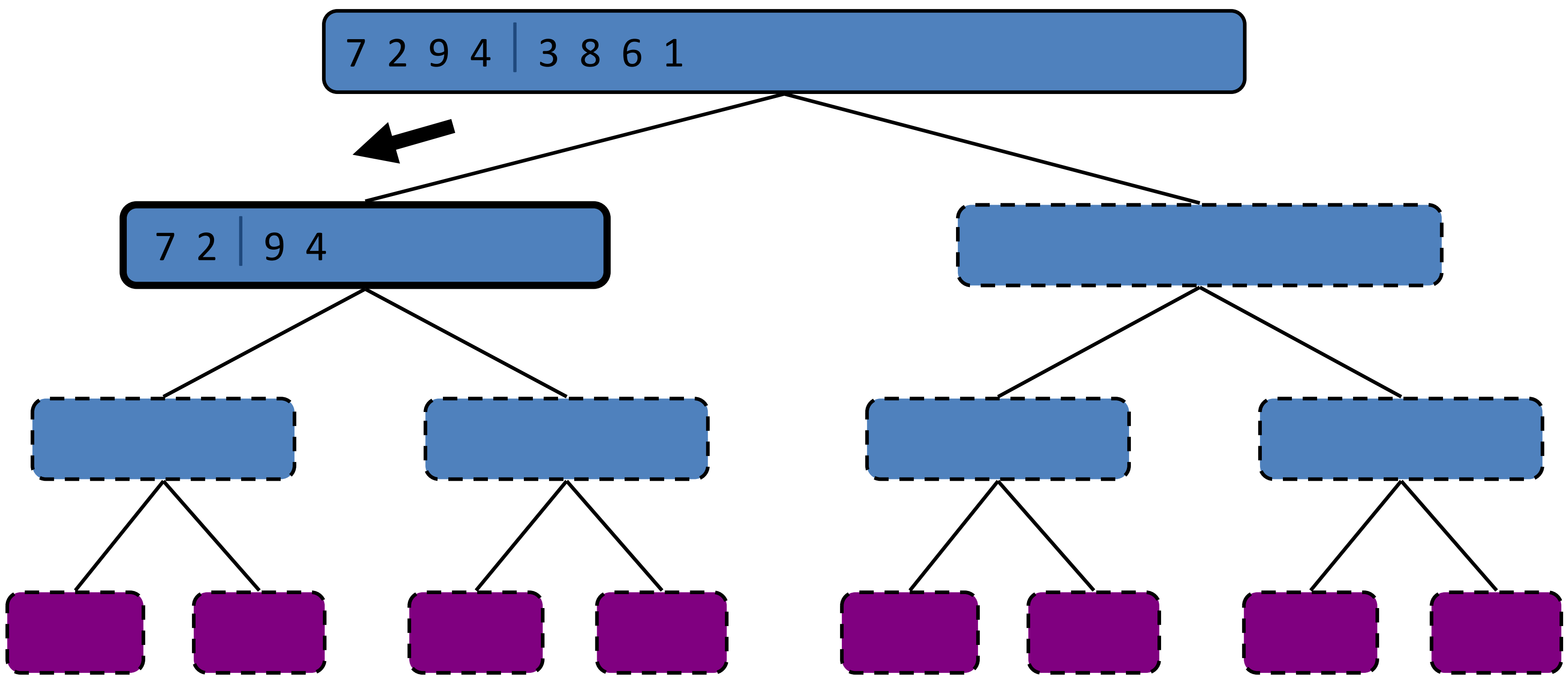
Mergesort (Example)

Example: Sort **7 2 9 4 3 8 6 1**



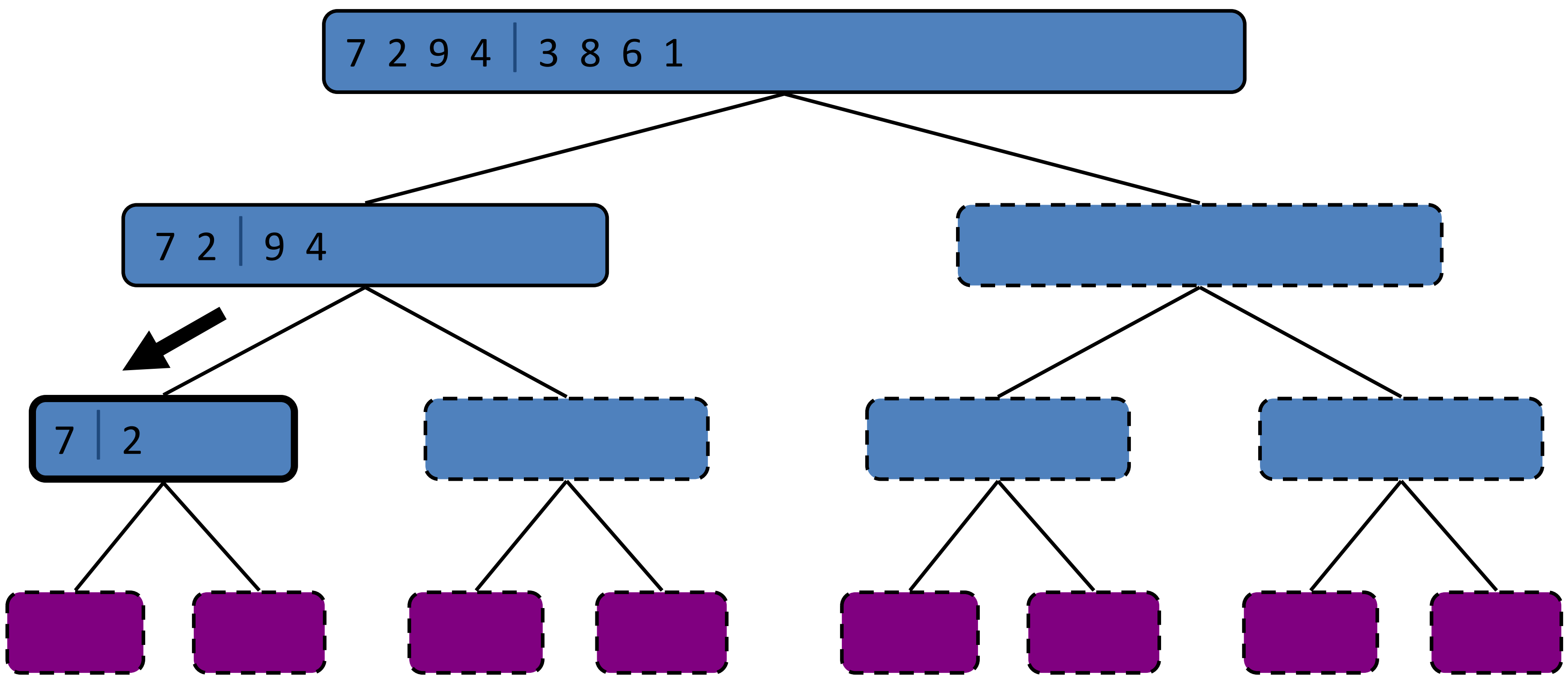
Mergesort (Example)

Recursive call, left part



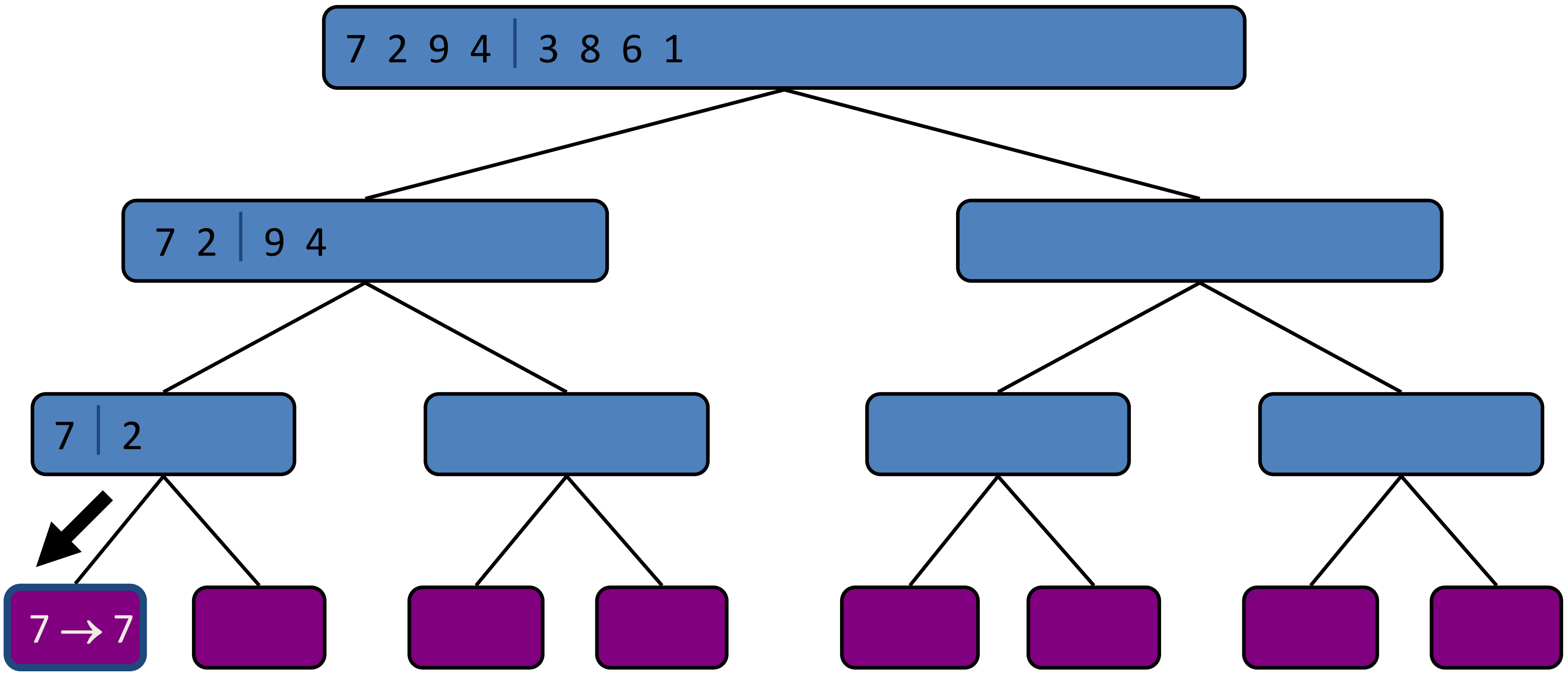
Mergesort (Example)

Recursive call, left part



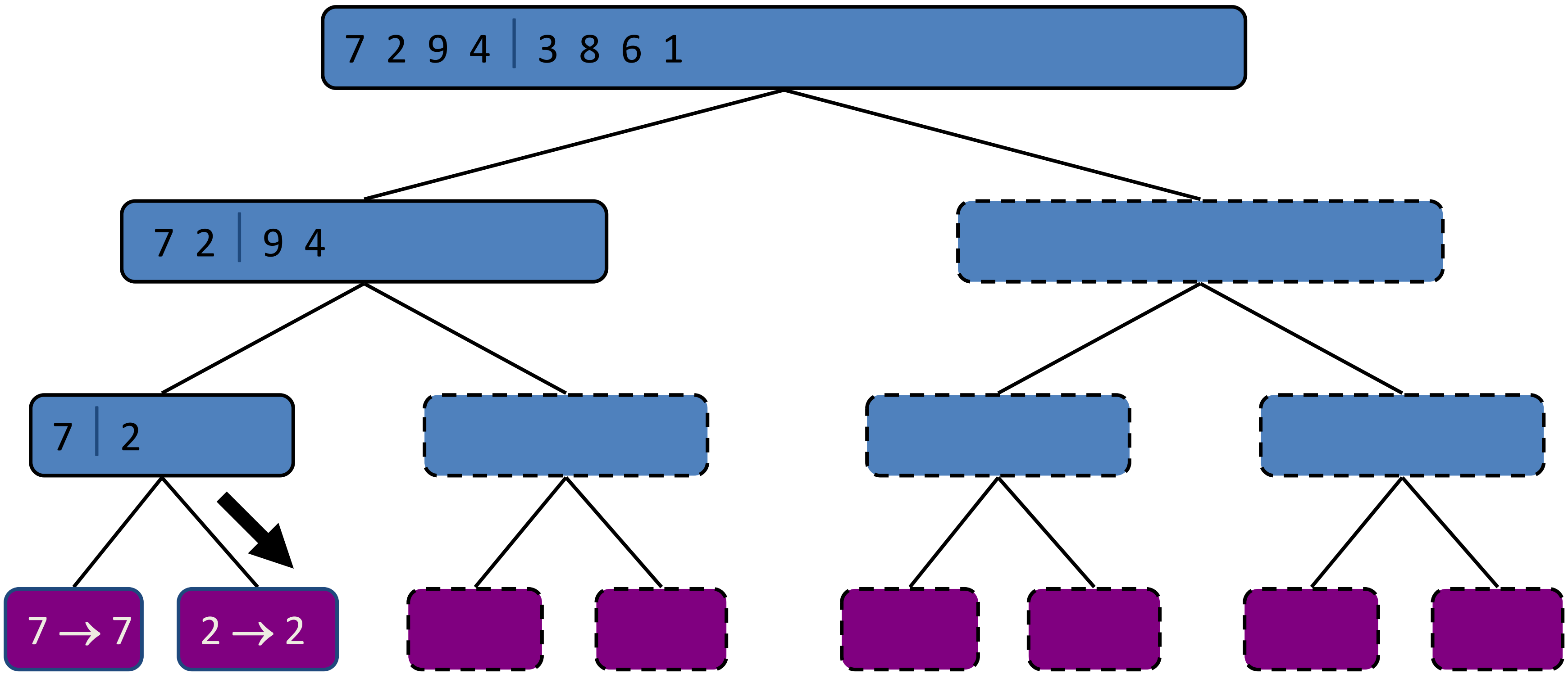
Mergesort (Example)

Recursive call, base case



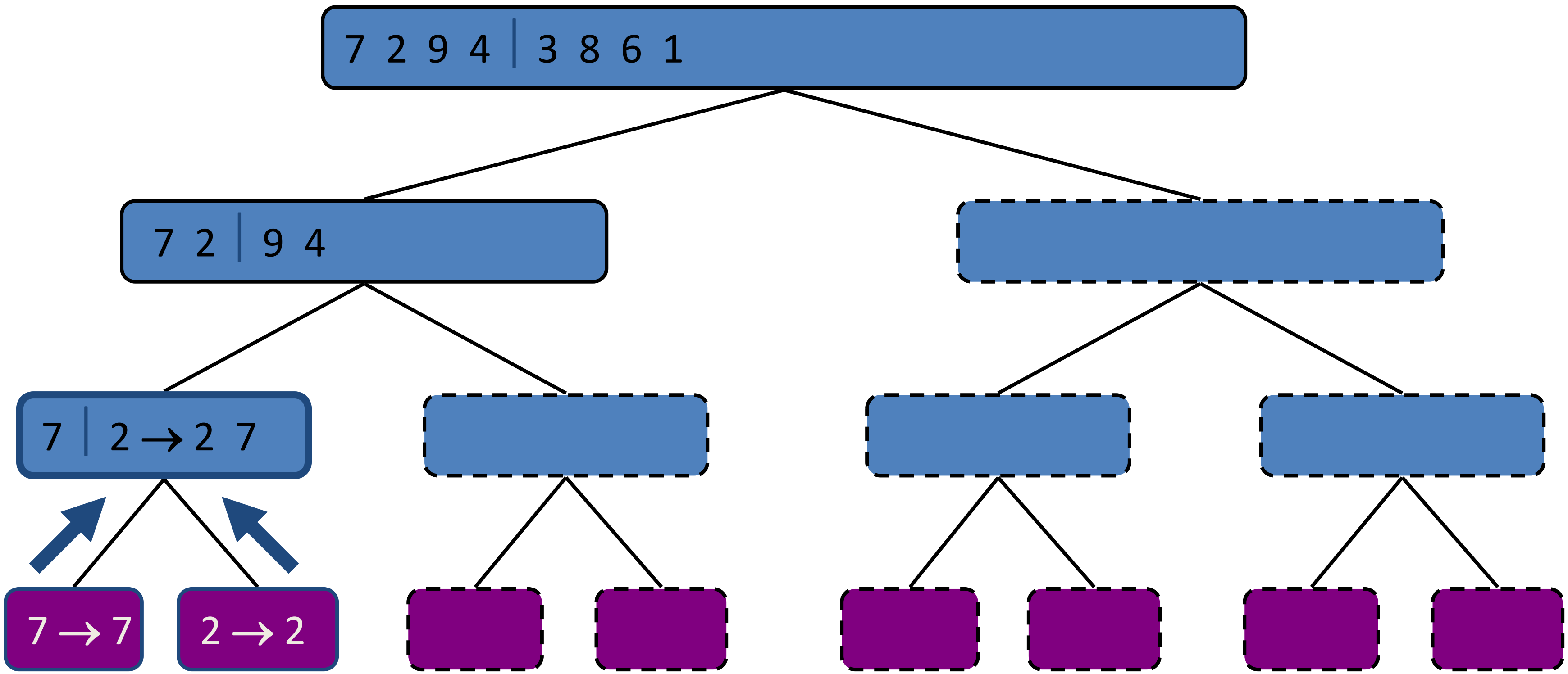
Mergesort (Example)

Recursive call, base case



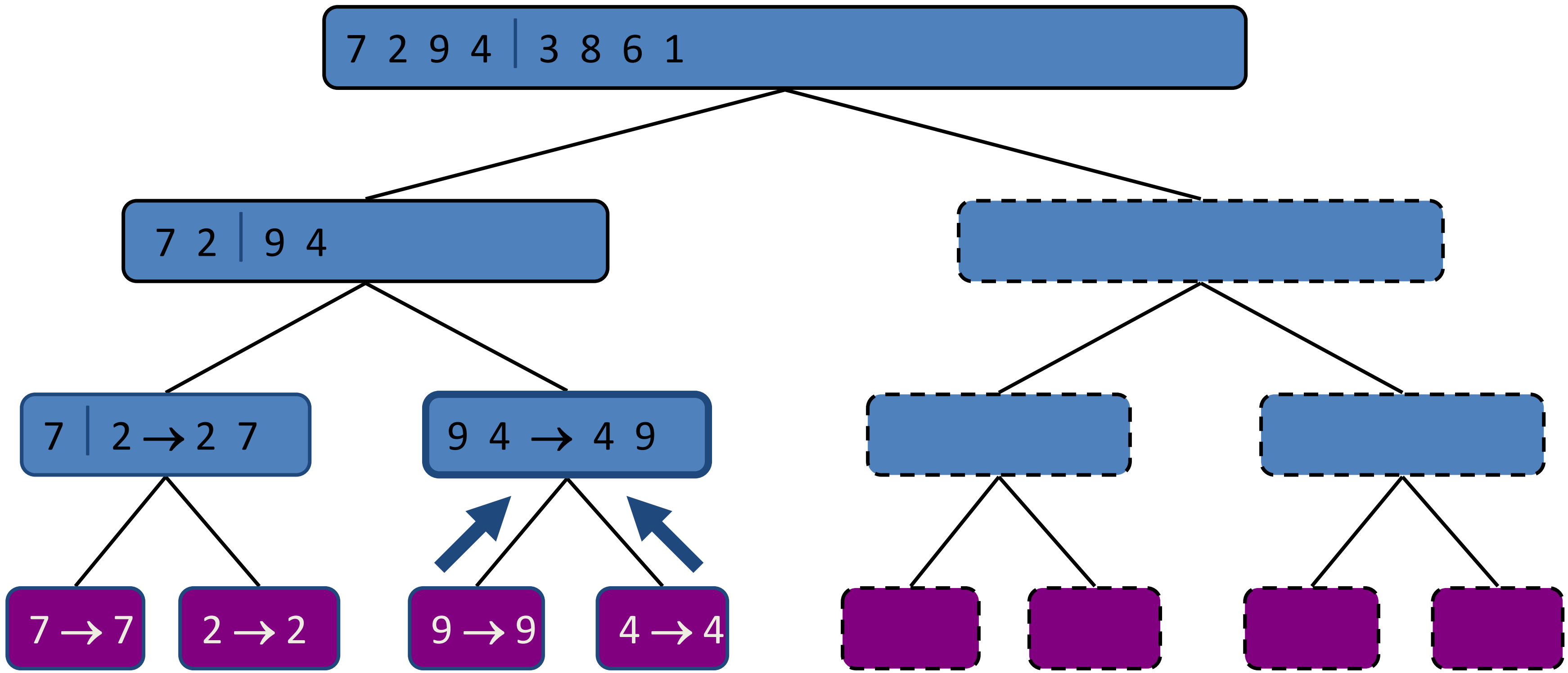
Mergesort (Example)

Merge



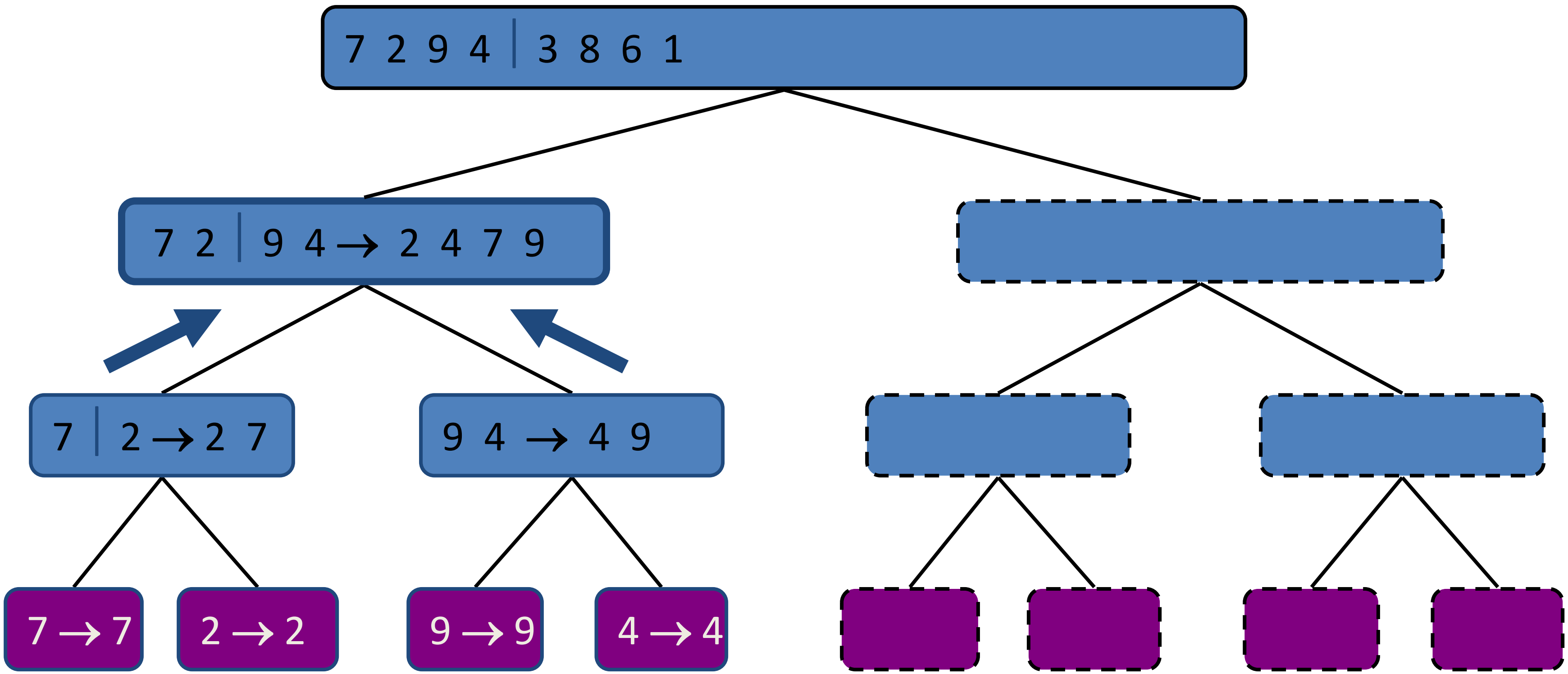
Mergesort (Example)

Similarly

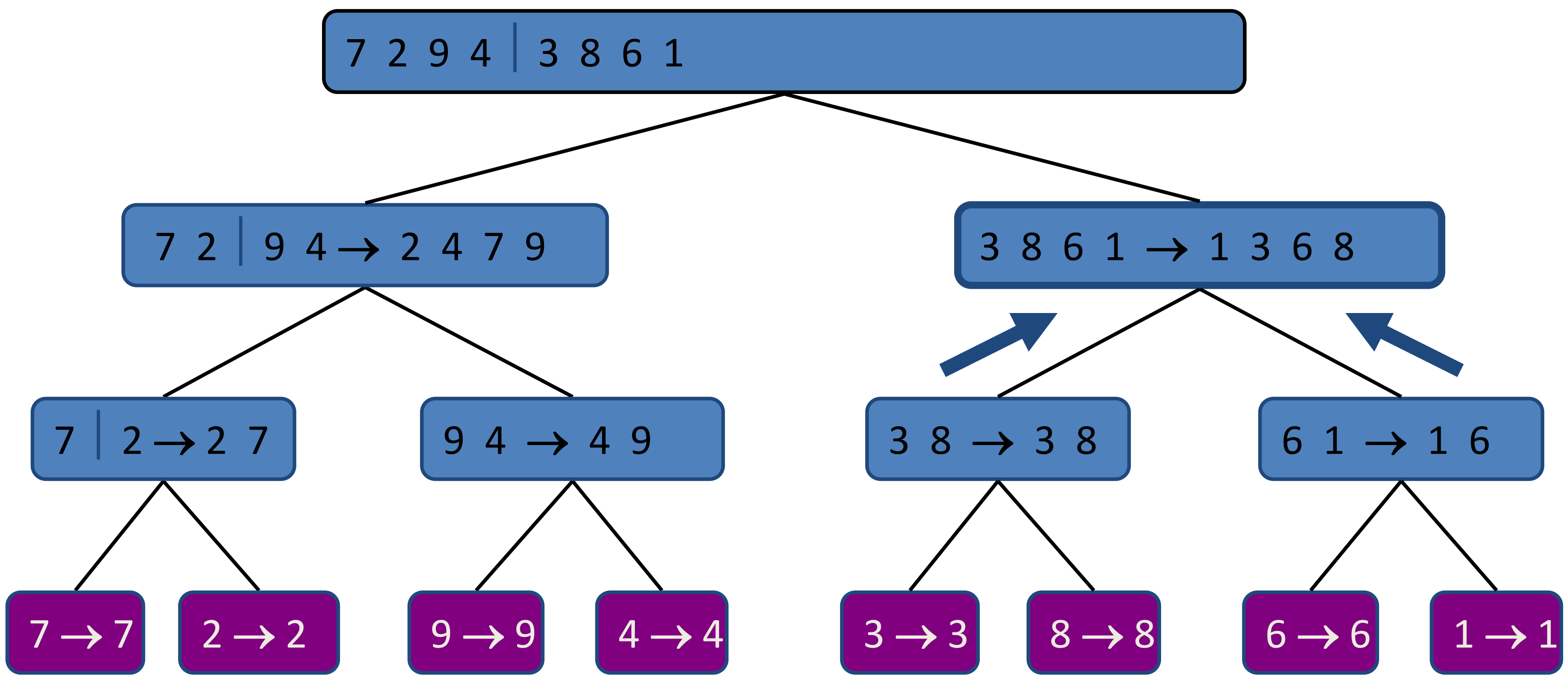


Mergesort (Example)

Merge

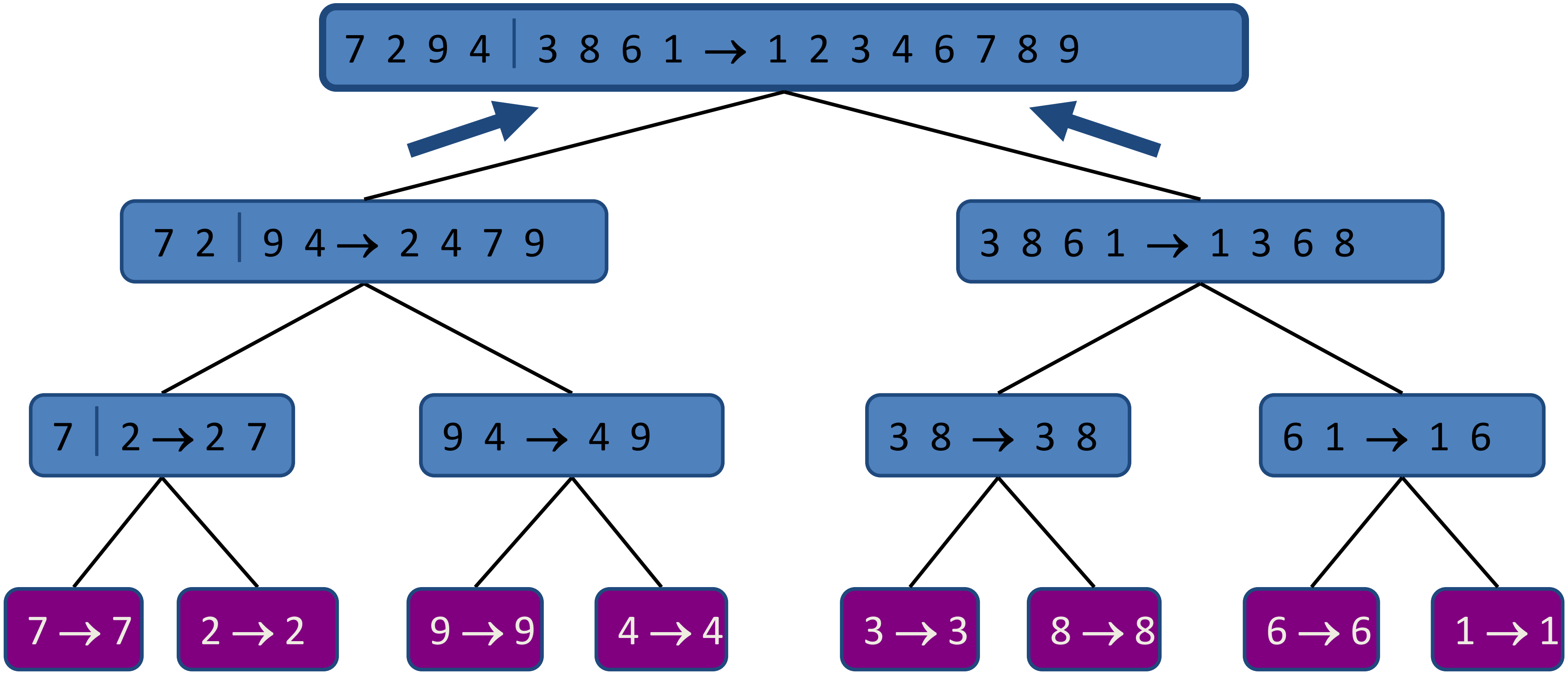


Mergesort (Example)



Mergesort (Example)

Merge



Mergesort

- Pseudocode:

$$
\begin{array}{l}
\text{Mergesort}(A[1:n]) \\
\quad \textbf{If } n == 1 \textbf{ then} \\
\quad\quad \textbf{return } A \\
\quad \text{Mergesort }(A[1:\frac{n}{2}]) \\
\quad \text{Mergesort }(A[\frac{n}{2}+1:n]) \\
\quad C \leftarrow \text{Merge}(A[1:\frac{n}{2}], A[\frac{n}{2}+1:n]) \\
\quad \textbf{return } C
\end{array}
$$

- Running time:

$$T(n) = T(n/2) + T(n/2) + \Theta(n) + \Theta(1)$$

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

Mergesort

- Pseudocode:

$$
\begin{array}{l}
\text{Mergesort}(A[1:n]) \\
\quad \textbf{If } n == 1 \textbf{ then} \\
\quad\quad \textbf{return } A \\
\quad \text{Mergesort}\,(A[1:\frac{n}{2}]) \\
\quad \text{Mergesort}\,(A[\frac{n}{2}+1:n]) \\
\quad C \leftarrow \text{Merge}(A[1:\frac{n}{2}], A[\frac{n}{2}+1:n]) \\
\quad \textbf{return } C
\end{array}
$$

- Running time:

$$
\begin{aligned}
T(n) &= T(n/2) + T(n/2) + \Theta(n) + \Theta(1) \\
&= 2T(n/2) + \Theta(n)
\end{aligned}
$$

How to analyze?

Master theorem

$$T(n) = \begin{cases} T(1) = \Theta(1) \\ aT(n/b) + f(n) \end{cases}$$

- The Master Theorem can find the order of $T(n)$ which is defined recursively.

Master theorem

$$T(n) = \begin{cases} T(1) = \Theta(1) \\ aT(n/b) + f(n) \end{cases}$$

- The **Master Theorem** can find the order of $T(n)$ which is defined recursively.

- Key idea: The answer depends on the comparison between $f(n)$ and $n^{\log_b a}$. So, there are **3** cases!

Master theorem

$$T(n) = \begin{cases} T(1) = \Theta(1) \\ aT(n/b) + f(n) \end{cases}$$

1. **If** $f(n)$ **is** $O(n^{\log_b a - \epsilon})$**, then** $T(n)$ **is** $\Theta(n^{\log_b a})$
2. If $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
3. If $f(n)$ is $\Omega(n^{\log_b a + \epsilon})$, then $T(n)$ is $\Theta(f(n))$,
   need to check $af(n/b) < f(n)$.

Case 1: $n^{\log_b a}$ **dominates** $f(n)$

Master theorem

$$T(n) = \begin{cases} T(1) = \Theta(1) \\ aT(n/b) + f(n) \end{cases}$$

1. If $f(n)$ is $O(n^{\log_b a - \epsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
2. **If $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$**
3. If $f(n)$ is $\Omega(n^{\log_b a + \epsilon})$, then $T(n)$ is $\Theta(f(n))$,
   need to check $af(n/b) < f(n)$.

Case 2: $n^{\log_b a}$ **have same order as** $f(n)$ (up to $\log^k n$)

Master theorem

$$T(n) = \begin{cases} T(1) = \Theta(1) \\ aT(n/b) + f(n) \end{cases}$$

$f(n) = \dfrac{n \log n}{n} \geq \dfrac{n^{1+\epsilon}}{n}$

$\log n \cancel{\geq} \quad n^{\epsilon}$ $\qquad \log n < n^{\epsilon}$

1. If $f(n)$ is $O(n^{\log_b a - \epsilon})$, then $T(n)$ is $\Theta(n^{\log_b a})$
2. If $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then $T(n)$ is $\Theta(n^{\log_b a} \log^{k+1} n)$
3. **If $f(n)$ is $\Omega(n^{\log_b a + \epsilon})$, then $T(n)$ is $\Theta(f(n))$, need to check $af(n/b) < f(n)$.**

Case 3: $n^{\log_b a}$ **is dominated by** $f(n)$ (+ another condition)

Master Theorem (Examples)

$$T(n) = 4T(n/2) + n$$

Master Theorem (Examples)

$$T(n) = 4T(n/2) + n$$

Solution:

We have $a = 4$, $b = 2$, hence $\log_b a = 2$.

Since $n^2 \gg n$, we are in case 1. Answer is $\Theta(n^2)$

Master Theorem (Examples)

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

Mergesort running time

Master Theorem (Examples)

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

Mergesort running time

Solution:

We have $a = 2$, $b = 2$, hence $\log_b a = 1$.

Since n is $\Theta(n)$, we are in case 2 with $k = 0$.
Answer is $\Theta(n \log n)$

Master Theorem (Examples)

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

Master Theorem (Examples)

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

Solution:

We have $a = 2$, $b = 2$, hence $\log_b a = 1$.

Since n is $\Theta(n)$, we are in case 2 with $k = 1$.
Answer is $\Theta(n\log^2 n)$

Master Theorem (Examples)

$$T(n) = 9T(n/3) + n^3$$

Master Theorem (Examples)

$$T(n) = 9T(n/3) + n^3$$

Solution:

We have $a = 9$, $b = 3$, hence $\log_b a = 2$.

Since $n^2 \ll n^3$, we are in case 3. Need to check that $\boldsymbol{9\left(\frac{n}{3}\right)^3 < n^3}$ which is equivalent to $\boldsymbol{\frac{n^3}{3} < n^3}$ (holds)

Answer is $\Theta(n^3)$

Master Theorem (Examples)

$$T(n) = T(n/2) + \Theta(1)$$

Binary search running time

Master Theorem (Examples)

$$T(n) = T(n/2) + \Theta(1)$$

Binary search running time

Solution:

We have $a = 1$, $b = 2$, hence $\log_b a = 0$.

Since n^0=1 is $\Theta(1)$, we are in case 2 with $k = 0$.
Answer is $\Theta(\log n)$

Divide and conquer method

- Steps of method:
  - **Divide** input into parts (smaller problems)
  - **Conquer** (solve) each part recursively
  - **Combine** results to obtain solution of original

$$
\begin{aligned}
T(n) = &\ \text{divide time} \\
&+ T(n_1) + T(n_2) + \ldots + T(n_k) \\
&+ \text{combine time}
\end{aligned}
$$

Case study I: Counting inversions

Given numbers $A_1, \ldots, A_n$ in an array A, compute the number of inversions.

(i, j) is an inversion: $A_i > A_j$ and $i < j$.

Case study I: Counting inversions

Given numbers $A_1, \dots, A_n$ in an array A, compute the number of inversions.

(i, j) is an inversion: $A_i > A_j$ and $i < j$.

Example [18, 29, 12, 15, 32, 10] has 9 inversions:

(18,12), (18,15), (18,10), (29,12), (29,15), (29,10), (12,10), (15,10), (32,10)

Case study I: Counting inversions

Given numbers $A_1, \ldots, A_n$ in an array A, compute the number of inversions.

(i, j) is an inversion: $A_i > A_j$ and $i < j$.

- Minimum number of inversions is zero (when sorted in increasing order)
- Maximum number of inversions is $\binom{n}{2}$ (when sorted in decreasing order)

Case study I: Counting inversions

- For all i, j with $i < j$, compare A_i with A_j and increase counter if $A_i > A_j$. Total number of comparisons $\frac{n(n-1)}{2}$. Running time $\Theta(n^2)$.

- Use Divide and conquer. Tricky part the combine step.

Case study I: Counting inversions

- For all i, j with $i < j$, compare A_i with A_j and increase counter if $A_i > A_j$. Total number of comparisons $\frac{n(n-1)}{2}$. Running time $\Theta(n^2)$.
- Use Divide and conquer. Tricky part the combine step.
- Question: Assume that $B_1, \ldots, B_k$ and $C_1, \ldots, C_l$ are both sorted. Can you compute the number of inversions of the sequence $B_1, \ldots, B_k, C_1, \ldots, C_l$?

Case study I: Counting inversions

- Question: Assume that $B_1, \dots, B_k$ and $C_1, \dots, C_l$ are both sorted. Can you compute the number of inversions of the sequence $B_1, \dots, B_k, C_1, \dots, C_l$?

If $B_i > C_j \geq B_{i-1}$ there are

including C_j

$B_1, \dots, B_i, \dots, B_k$	$C_1, \dots, C_j, \dots, C_l$
i	j

$C_j > C_1, C_2, \dots, C_{j-1}$

$C_j < \underbrace{B_i, B_{i+1}, B_{i+2}, \dots, B_k}$ $\quad k-i+1$

Case study I: Counting inversions

- Question: Assume that $B_1, \ldots, B_k$ and $C_1, \ldots, C_l$ are both sorted. Can you compute the number of inversions of the sequence $B_1, \ldots, B_k, C_1, \ldots, C_l$?

If $B_i > C_j \geq B_{i-1}$ there are

$k - i + 1$ including C_j

$B_1, \ldots, B_i, \ldots, B_k$ $\qquad$ $C_1, \ldots, C_j, \ldots, C_l$

i $\qquad$ j

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$\textbf{While } k < \mathrm{len(B)} + \mathrm{len(C)} \textbf{ do}$$

$$\quad \textbf{If } B[i] <= C[j] \textbf{ then}$$

$$\quad\quad A[k] \leftarrow B[i]$$

$$\quad\quad i = i + 1, k = k + 1$$

$$\quad \textbf{else}$$

$$\quad\quad A[k] \leftarrow C[j]$$

$$\quad\quad \mathrm{counter} = \mathrm{counter} + \mathrm{len}(B) - i + 1$$

$$\quad\quad j = j + 1, k = k + 1$$

$B :$ 1 3 4 9 220 $\quad$ $C :$ 2 3 5 7 8 10

i $\quad$ j

$A :$

$\mathrm{counter} = 0$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

While $k < \text{len(B)} + \text{len(C)}$ **do**

  If $B[i] <= C[j]$ **then**

    $A[k] \leftarrow B[i]$

    $i = i + 1, k = k + 1$

  else

    $A[k] \leftarrow C[j]$

    $\text{counter} = \text{counter} + \text{len}(B) - i + 1$

    $j = j + 1, k = k + 1$

B : 1 3 4 9 220 C : 2 3 5 7 8 10

i j

$A : 1$

$\text{counter} = 0$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$\textbf{While } k < \text{len(B)} + \text{len(C)} \textbf{ do}$

$\quad \textbf{If } B[i] <= C[j] \textbf{ then}$

$\quad\quad A[k] \leftarrow B[i]$

$\quad\quad i = i + 1, k = k + 1$

$\quad \textbf{else}$

$\quad\quad A[k] \leftarrow C[j]$

$\quad\quad \text{counter} = \text{counter} + \text{len}(B) - i + 1$

$\quad\quad j = j + 1, k = k + 1$

$B: 1\ 3\ 4\ 9\ 220$ $\qquad$ $C: 2\ 3\ 5\ 7\ 8\ 10$

i $\qquad$ j

$A: 1\ 2$

$\text{counter} = 4$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$
\begin{array}{l}
\textbf{While } k < \mathrm{len(B)} + \mathrm{len(C)} \textbf{ do} \\
\quad \textbf{If } B[i] <= C[j] \textbf{ then} \\
\quad\quad A[k] \leftarrow B[i] \\
\quad\quad i = i + 1, k = k + 1 \\
\quad \textbf{else} \\
\quad\quad A[k] \leftarrow C[j] \\
\quad\quad \mathrm{counter} = \mathrm{counter} + \mathrm{len}(B) - i + 1 \\
\quad\quad j = j + 1, k = k + 1
\end{array}
$$

$B: 1\ 3\ 4\ 9\ 220$ $\qquad C: 2\ 3\ 5\ 7\ 8\ 10$

$\qquad\quad i \qquad\qquad\qquad\qquad\qquad j$

$A: 1\ 2\ 3$

$\mathrm{counter} = 4$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$
\begin{array}{l}
\textbf{While } k < \mathrm{len(B)} + \mathrm{len(C)} \textbf{ do} \\
\quad \textbf{If } B[i] <= C[j] \textbf{ then} \\
\quad\quad A[k] \leftarrow B[i] \\
\quad\quad i = i + 1, k = k + 1 \\
\quad \textbf{else} \\
\quad\quad A[k] \leftarrow C[j] \\
\quad\quad \mathrm{counter} = \mathrm{counter} + \mathrm{len}(B) - i + 1 \\
\quad\quad j = j + 1, k = k + 1
\end{array}
$$

$B: 1\ 3\ 4\ 9\ 220$ $\quad$ $C: 2\ 3\ 5\ 7\ 8\ 10$

i $\quad$ j

$A: 1\ 2\ 3\ 3$

$\mathrm{counter} = 7$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

While $k < \text{len(B)} + \text{len(C)}$ **do**

  If $B[i] <= C[j]$ **then**

    $A[k] \leftarrow B[i]$

    $i = i + 1, k = k + 1$

  else

    $A[k] \leftarrow C[j]$

    $\text{counter} = \text{counter} + \text{len}(B) - i + 1$

    $j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 220$ $C : 2\ 3\ 5\ 7\ 8\ 10$

i j

$A : 1\ 2\ 3\ 3\ 4$

$\text{counter} = 7$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$
\begin{aligned}
&\textbf{While } k < \text{len(B)} + \text{len(C)} \textbf{ do}\\
&\quad \textbf{If } B[i] <= C[j] \textbf{ then}\\
&\quad\quad A[k] \leftarrow B[i]\\
&\quad\quad i = i + 1, k = k + 1\\
&\quad \textbf{else}\\
&\quad\quad A[k] \leftarrow C[j]\\
&\quad\quad \text{counter} = \text{counter} + \text{len}(B) - i + 1\\
&\quad\quad j = j + 1, k = k + 1
\end{aligned}
$$

B : 1 3 4 9 220 $\qquad$ C : 2 3 5 7 8 10

i $\qquad$ j

$A : 1\ 2\ 3\ 3\ 4\ 5$

$\text{counter} = 9$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$
\begin{aligned}
&\textbf{While } k < \text{len(B)} + \text{len(C)} \textbf{ do}\\
&\quad \textbf{If } B[i] <= C[j] \textbf{ then}\\
&\qquad A[k] \leftarrow B[i]\\
&\qquad i = i + 1, k = k + 1\\
&\quad \textbf{else}\\
&\qquad A[k] \leftarrow C[j]\\
&\qquad \text{counter} = \text{counter} + \text{len}(B) - i + 1\\
&\qquad j = j + 1, k = k + 1
\end{aligned}
$$

$B: 1\ 3\ 4\ 9\ 220$ $\quad C: 2\ 3\ 5\ 7\ 8\ 10$

i $\quad j$

$A: 1\ 2\ 3\ 3\ 4\ 5\ 7$

$\text{counter} = 11$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$\textbf{While } k < \text{len(B)} + \text{len(C)} \textbf{ do}$$

$\quad \textbf{If } B[i] <= C[j] \textbf{ then}$

$\quad\quad A[k] \leftarrow B[i]$

$\quad\quad i = i + 1, k = k + 1$

$\quad \textbf{else}$

$\quad\quad A[k] \leftarrow C[j]$

$\quad\quad \text{counter} = \text{counter} + \text{len}(B) - i + 1$

$\quad\quad j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 220$ $\qquad C : 2\ 3\ 5\ 7\ 8\ 10$

i $\qquad j$

$A : 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8$

$\text{counter} = 13$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$
\begin{array}{l}
\textbf{While } k < \text{len(B)} + \text{len(C)} \textbf{ do} \\
\quad \textbf{If } B[i] <= C[j] \textbf{ then} \\
\quad\quad A[k] \leftarrow B[i] \\
\quad\quad i = i + 1, k = k + 1 \\
\quad \textbf{else} \\
\quad\quad A[k] \leftarrow C[j] \\
\quad\quad \text{counter} = \text{counter} + \text{len}(B) - i + 1 \\
\quad\quad j = j + 1, k = k + 1
\end{array}
$$

$B: 1\ 3\ 4\ 9\ 220$ $\qquad$ $C: 2\ 3\ 5\ 7\ 8\ 10$

i $\qquad$ j

$A: 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9$

$\text{counter} = 13$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

While $k <$ len(B) + len(C) **do**

    If $B[i] <= C[j]$ **then**

        $A[k] \leftarrow B[i]$

        $i = i + 1, k = k + 1$

    else

        $A[k] \leftarrow C[j]$

        $\text{counter} = \text{counter} + \text{len}(B) - i + 1$

        $j = j + 1, k = k + 1$

B : 1 3 4 9 220 C : 2 3 5 7 8 10

i j

A : 1 2 3 3 4 5 7 8 9 10

$\text{counter} = 14$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C, merge them to a sorted array and count number of inversions simultaneously.

Solution: Index i for B, index j for C, index k for A, counter.

$$
\begin{array}{l}
\textbf{While } k < \mathrm{len(B)} + \mathrm{len(C)} \textbf{ do} \\
\quad \textbf{If } B[i] <= C[j] \textbf{ then} \\
\qquad A[k] \leftarrow B[i] \\
\qquad i = i + 1, k = k + 1 \\
\quad \textbf{else} \\
\qquad A[k] \leftarrow C[j] \\
\qquad \text{counter} = \text{counter} + \mathrm{len}(B) - i + 1 \\
\qquad j = j + 1, k = k + 1
\end{array}
$$

$B: 1\ 3\ 4\ 9\ 220$ $\quad$ $C: 2\ 3\ 5\ 7\ 8\ 10$

i $\quad$ j

$A: 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9\ 10\ 220$

$\text{counter} = 14$

Case study I: Counting inversions

- For all i, j with $i < j$, compare A_i with A_j and increase counter if $A_i > A_j$. Total number of comparisons $\frac{n(n-1)}{2}$. Running time $\Theta(n^2)$.

- Use Divide and conquer. Tricky part the combine step.

- Solution: Run a modification of Mergesort that has a counter that counts inversions during merge steps.

Case study I: Counting inversions

- Pseudocode:

$$\text{Mergesort}(A[1:n])$$
$$\quad \textbf{If } n == 1 \textbf{ then}$$
$$\qquad \textbf{return } A, 0$$
$$\quad B, \textbf{countL} \;=\; \text{Mergesort }(A[1:\tfrac{n}{2}])$$
$$\quad C, \textbf{countR} \;=\; \text{Mergesort }(A[\tfrac{n}{2}+1:n])$$
$$\quad A \leftarrow \text{Merge}(B, C)$$
$$\quad \text{Get } \textbf{counter} \text{ from merging}$$
$$\quad \textbf{return } A, \text{countL} + \text{countR} + \text{counter}$$