There is an attendance quiz …

Where have we been? Where are we going?

There is an attendance quiz going …

Mathy stuff: Σ and big-Oh

Brute force

Decrease and conquer

Divide and conquer

Transform and conquer

This week:

There is an attendance quiz going …

- *Divide and Conquer* technique
- Example: Count a specific key in an array
- How to analyze Divide and Conquer (“Master Theorem”)
- Example: Mergesort
- Binary tree examples
  - *Computing the height*
  - *Compute the number of leaves*

But first …

Kahoot!

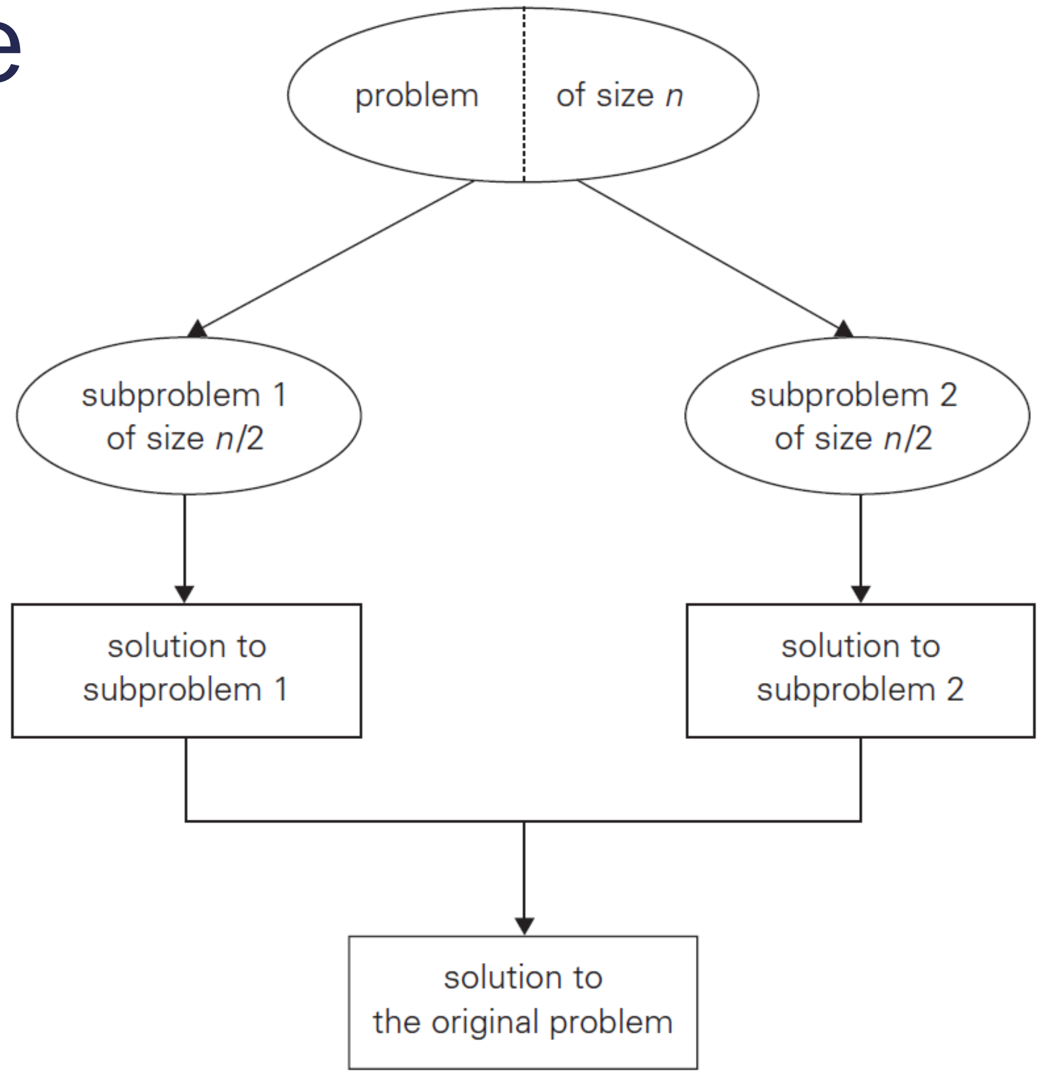
DIVIDE AND CONQUER

(Chapter 5)

Divide and Conquer technique

- Divide instance of problem into two or more smaller instances
- Solve smaller instances (usually recursively)
- Obtain solution to original (larger) instance by combining these solutions

Divide and Conquer technique



A natural question

- How is this different from Decrease and Conquer?

- Think of the fake coin problem:
  - *We discarded half the coins at each step*
  - *So we didn't do any work on those "sub problems"*

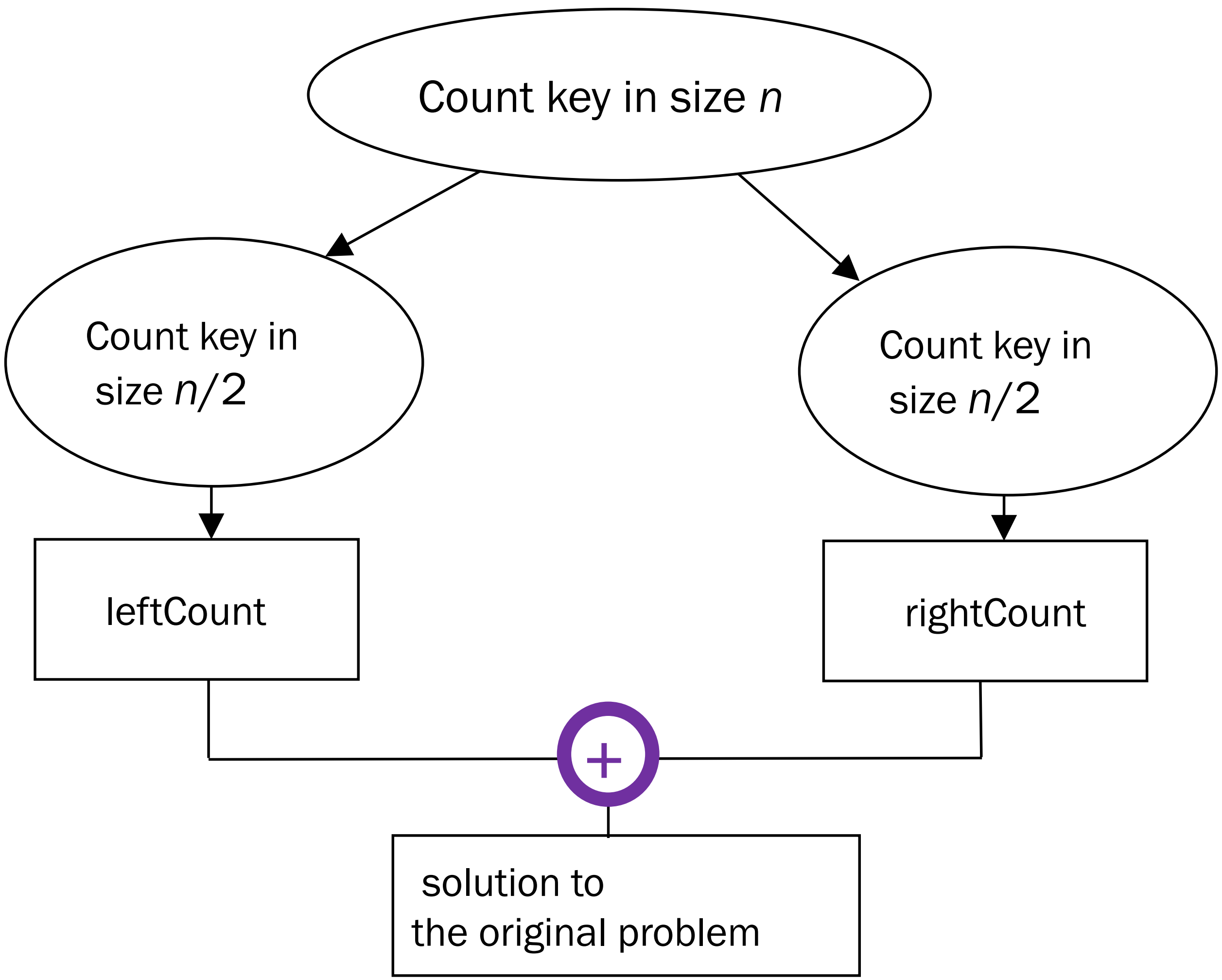
- For divide and conquer…
  - *You need to solve all of the sub problems*

EXAMPLE: COUNT A SPECIFIC KEY IN AN ARRAY

Count a specific key in an array

- Problem:
  - *Count the number of times a specific key occurs in an array.*
- For example:
  - *If input array is A=[2,7,6,6,2,4,6,9,2] and key=6…*
  - *… should return the value 3.*
- Design an algorithm using divide and conquer technique

Count a specific key in an array



Count a specific key in an array

```
Algorithm CountKey(A[], L, R, Key)
//Input: A[] is an array A[0..n−1] from indices L to R (L ≤ R)
//Output: A count of the number of time Key exists in A[L..R]

if L = R
   if (A[L] = Key) return 1
   else return 0
else
   lCount = CountKey(A[],        L,     ⌊(L+R)/2⌋, Key)
   rCount = CountKey(A[], ⌊(L+R)/2⌋+1,    R,      Key)
   return lCount + rCount
```

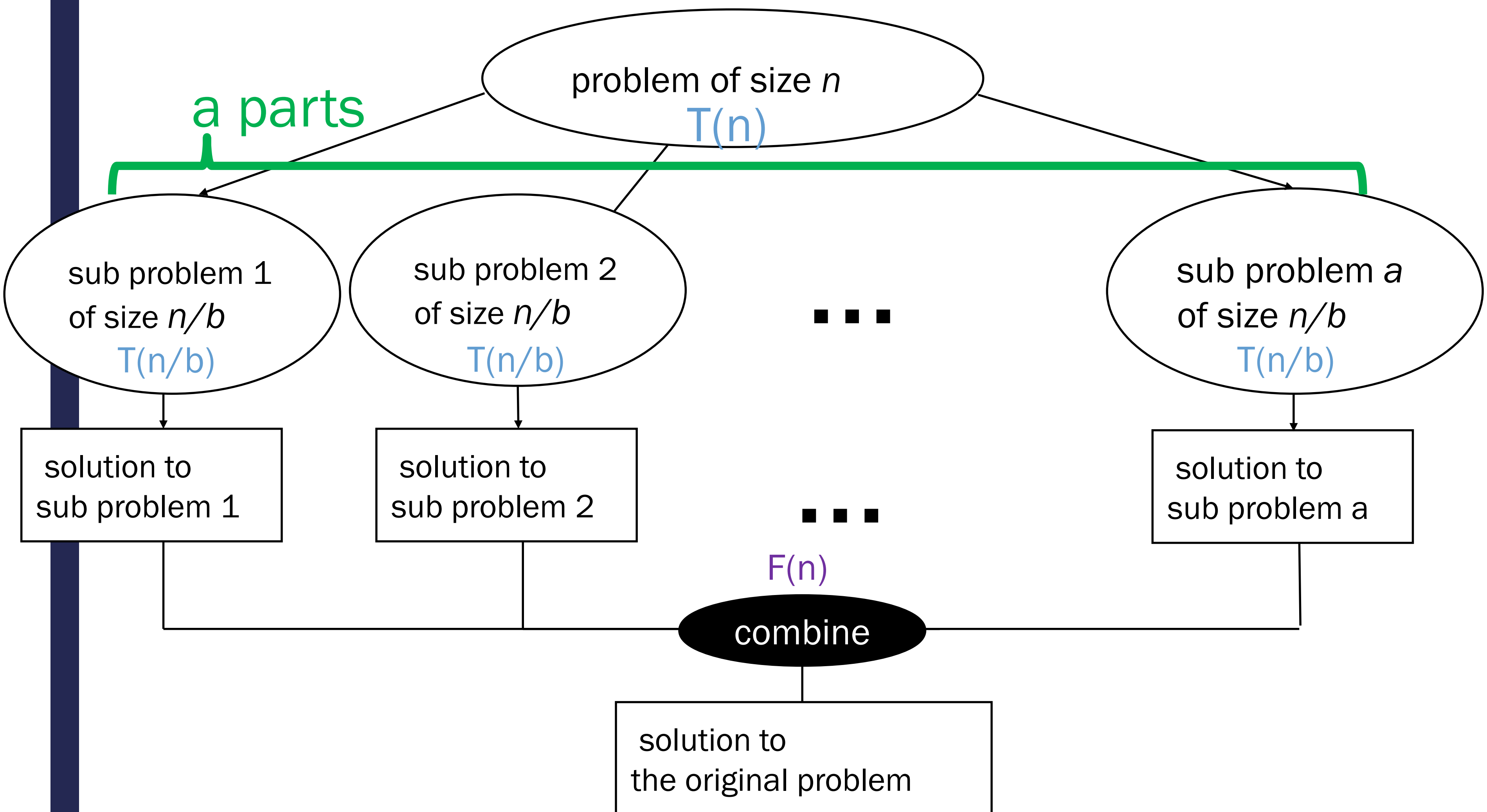
Count a specific key in an array

- CountKey looks familiar…
  - *What’s the difference between Binary Search and CountKey?*

- We have to search both sides
  - *In the counter, both sides must be searched*
  - *In Binary Search, one half gets ignored*

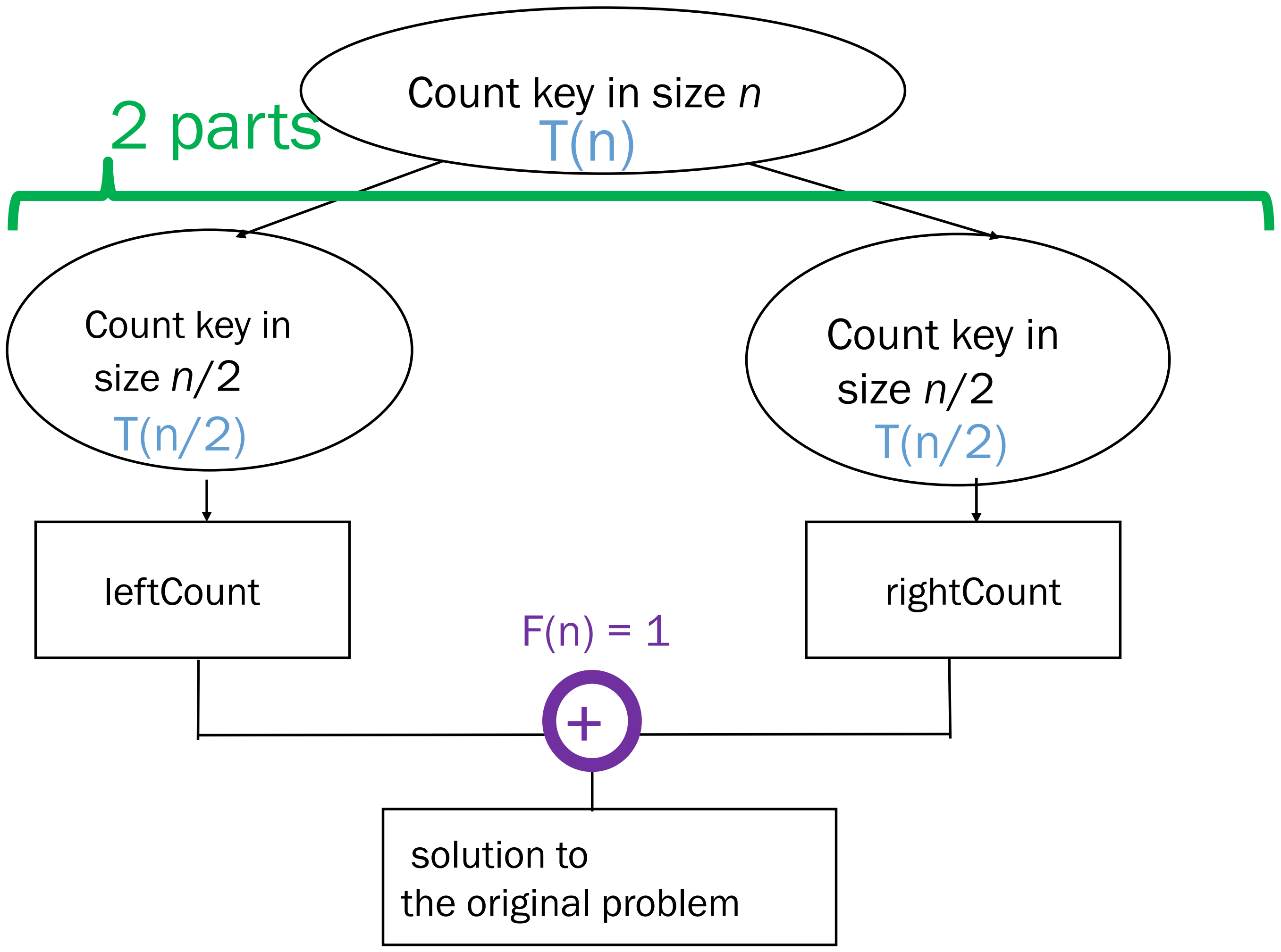
ANALYSIS OF DIVIDE AND CONQUER

Analysis of a divide and conquer algorithm



$$T(n) = a\, T(n/b) + F(n)$$

Example: analysis of Count a specific key in an array



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

What is the complexity (efficiency class)?

If $T(n) = a\, T(n/b) + F(n)$

The Master Theorem:

1) If $n^{\log_b a} < F(n)$, $\quad T(n) \in O(F(n))$
2) If $n^{\log_b a} > F(n)$, $\quad T(n) \in O(n^{\log_b a})$
3) If $n^{\log_b a} = F(n)$, $\quad T(n) \in O(n^{\log_b a} \log n)$

Master theorem

If $T(n) = a\, T(n/b) + F(n)$

Compare $n^{\log_b a}$ and $F(n)$

$\xrightarrow{<}$ $T(n) \in O(F(n))$

$\xrightarrow{=}$ $T(n) \in O(n^{\log_b a} \log n)$

$\xrightarrow{>}$ $T(n) \in O(n^{\log_b a})$

Example 1: $T(n) = 4T(n/2) + n^3$

What is the efficiency class of T(n)?

$$\left.\begin{array}{r} n^{\log_b a} \rightarrow n^{\log_2 4} \rightarrow n^2 \\ F(n) = n^3 \end{array}\right\} \rightarrow T(n) \in O(n^3)$$

Master theorem

Example 2: $T(n) = 4T(n/2) + n \Rightarrow T(n) \in ?$

$$\left.\begin{array}{l} a = 4 \\ b = 2 \end{array} \Rightarrow n^{\log_b a} \Rightarrow n^{\log_2 4} \Rightarrow n^2 \atop F(n) = n \right\} \Rightarrow T(n) \in O(n^2)$$

Example 3: $T(n) = 4T(n/2) + n^2 \Rightarrow T(n) \in ?$

$$\left.\begin{array}{l} a = 4 \\ b = 2 \end{array} \Rightarrow n^{\log_b a} \Rightarrow n^{\log_2 4} \Rightarrow n^2 \atop F(n) = n^2 \right\} \Rightarrow T(n) \in O(n^2 \log n)$$

Comparing those three examples

$T(n) = 4T(n/2) + n^3$

$n^{\log_b a} = n^2$
$F(n) = n^3$
$\Rightarrow T(n) \in O(n^3)$

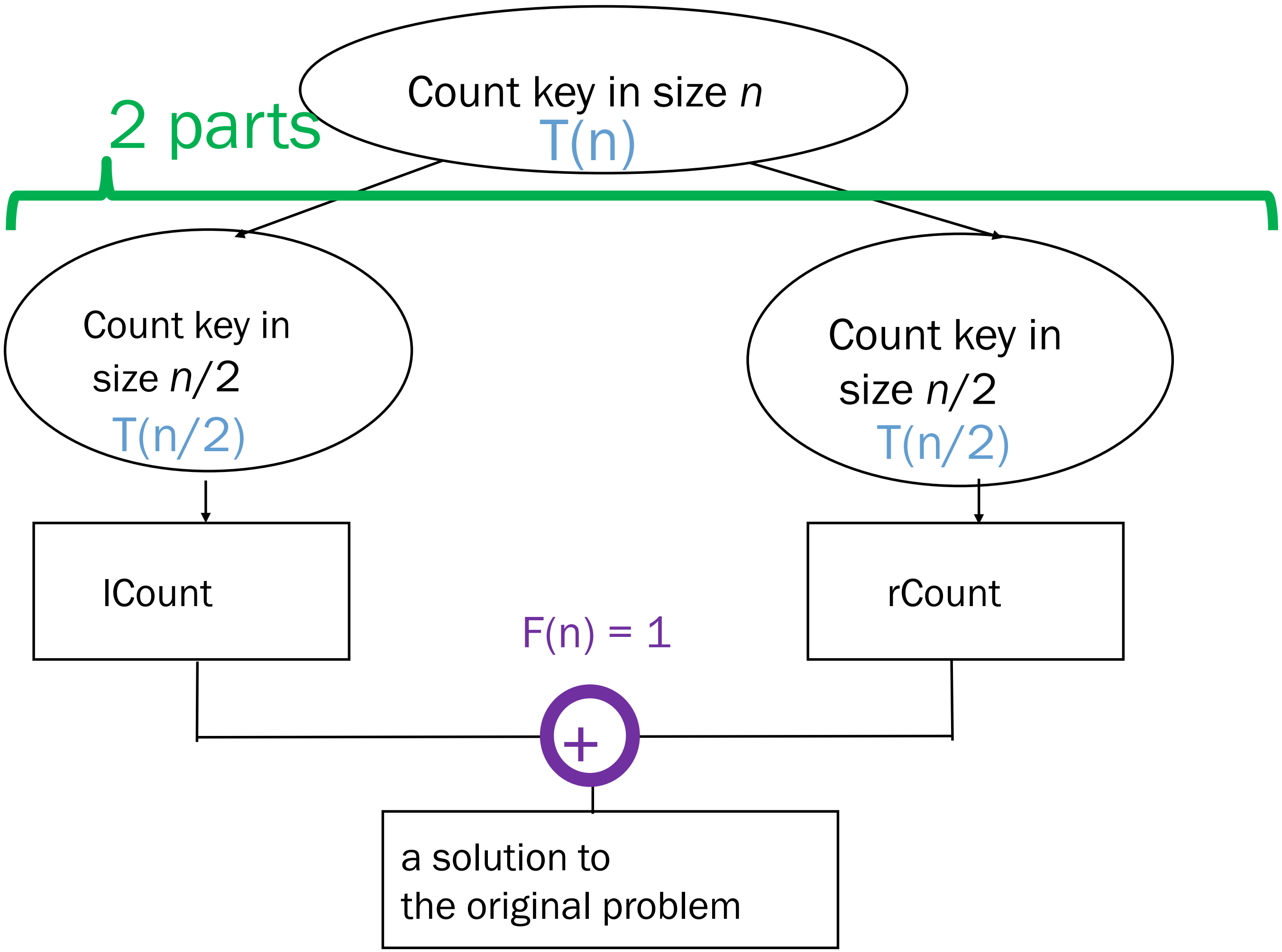
$T(n) = 4T(n/2) + n^2$

$n^{\log_b a} = n^2$
$F(n) = n^2$
$\Rightarrow T(n) \in O(n^2 \log n)$

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

$n^{\log_b a} = n^2$
$F(n) = n$
$\Rightarrow T(n) \in O(n^2)$

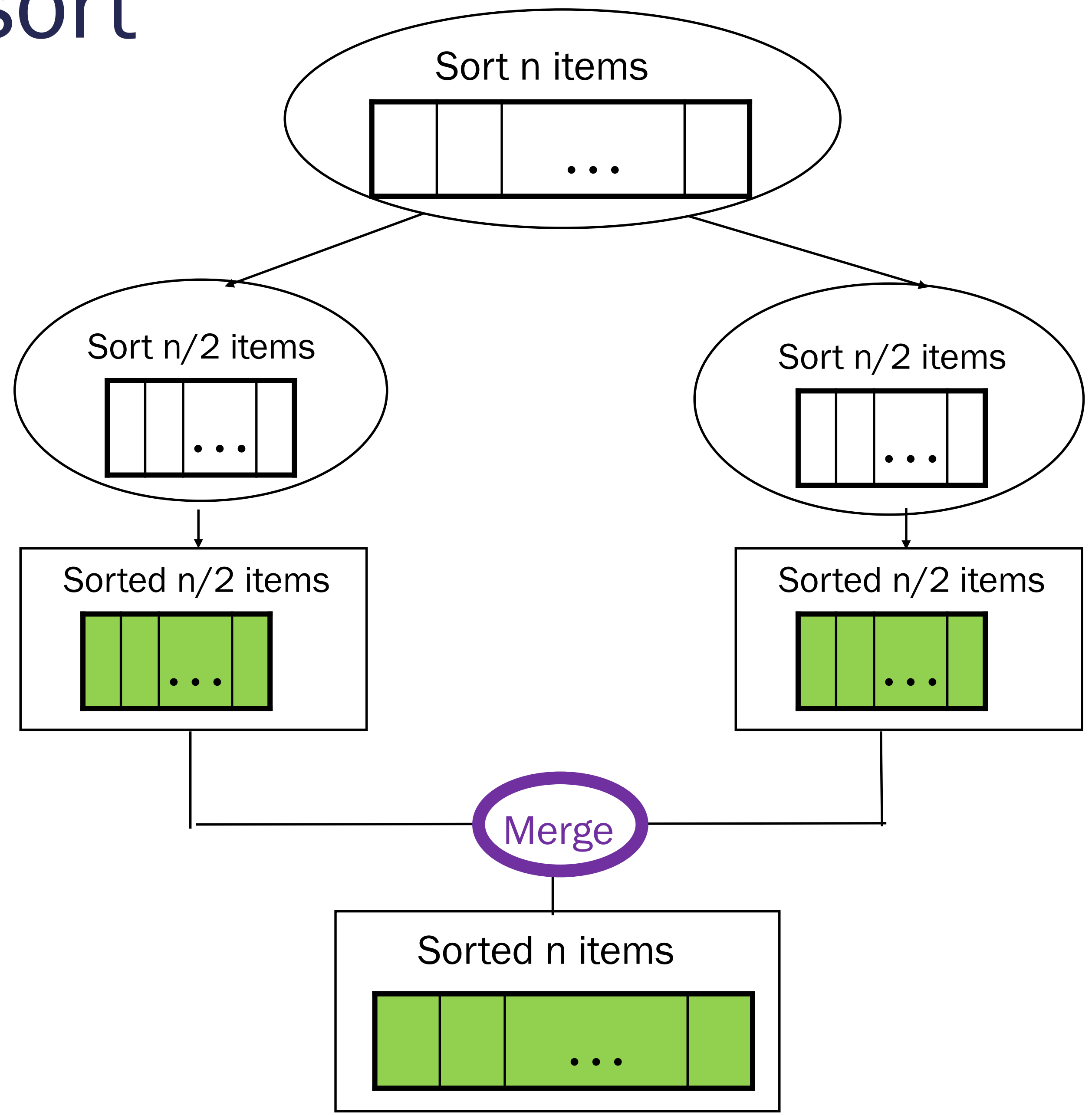
Analysis of
Count a specific key in an array



$$T(n) = 2\,T(n/2) + 1 \quad \Rightarrow \quad T(n) \in O(n)$$

MERGESORT

Mergesort



Sort n items
Sort n/2 items
Sort n/2 items
Sorted n/2 items
Sorted n/2 items
Merge
Sorted n items

Pseudocode of Mergesort

ALGORITHM $Mergesort(A[0..n-1])$

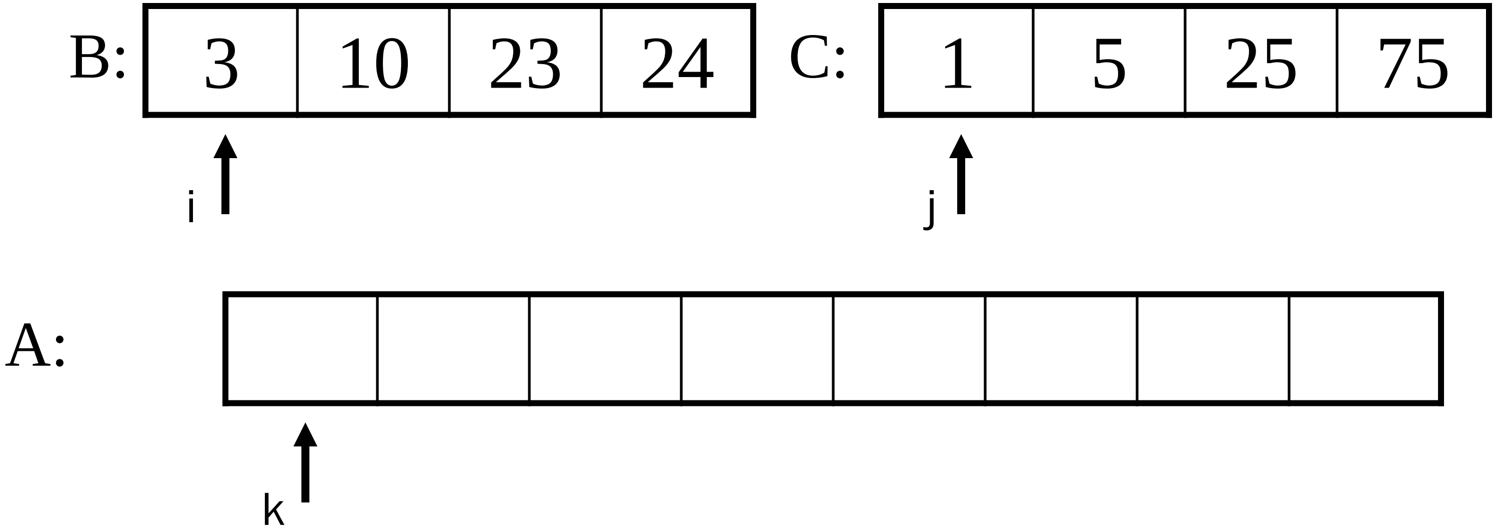
//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$
$Mergesort(B[0..\lfloor n/2 \rfloor - 1])$
$Mergesort(C[0..\lceil n/2 \rceil - 1])$
$Merge(B, C, A)$

Mergesort

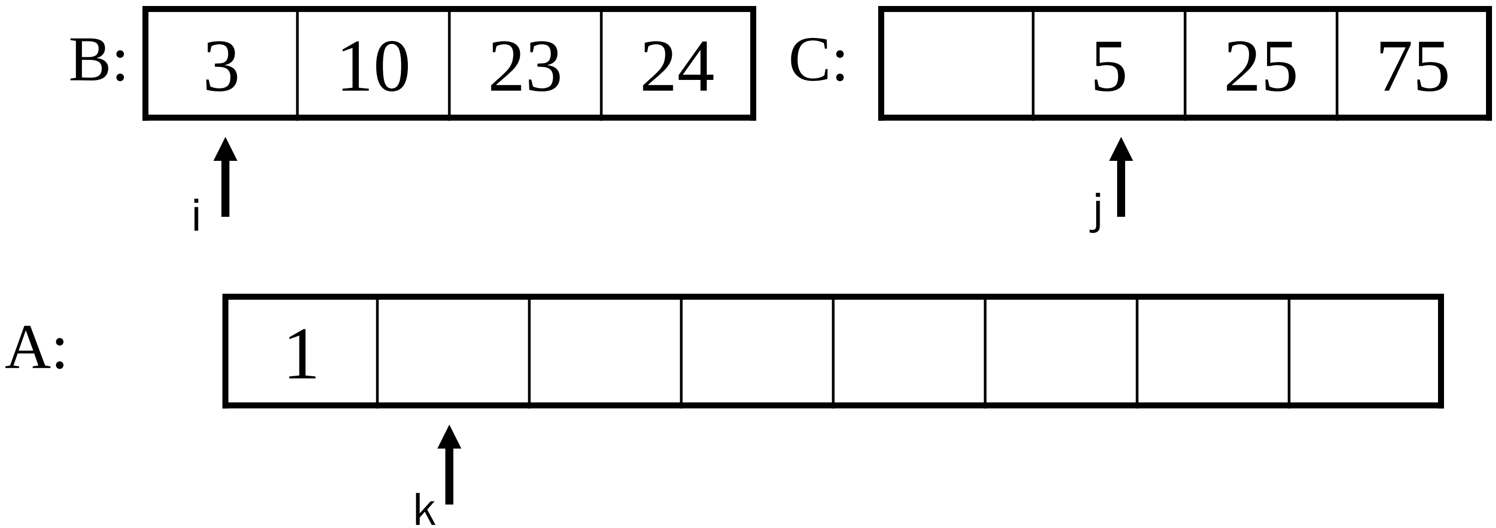
- The “combine partial solutions” part of mergesort is to merge two sorted arrays into one

- Example:
  - $B = \{ 3\ 8\ 9 \}$ $C = \{ 1\ 5\ 7 \}$
  - $merge(B, C) = \{ 1\ 3\ 5\ 7\ 8\ 9 \}$

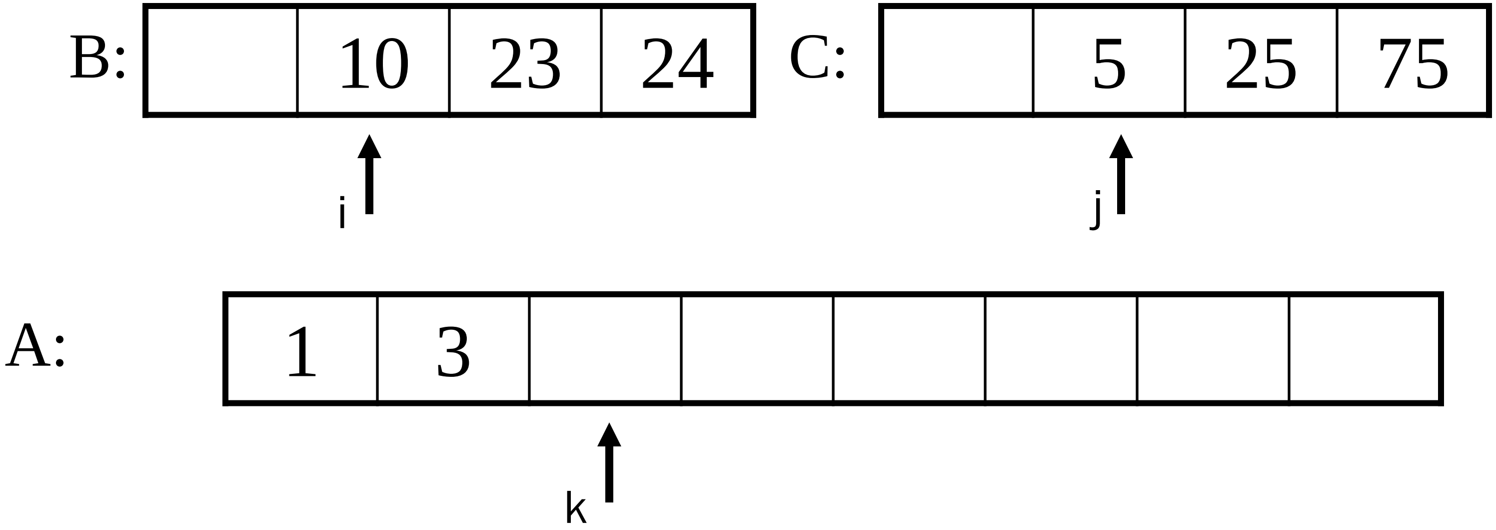
Merging



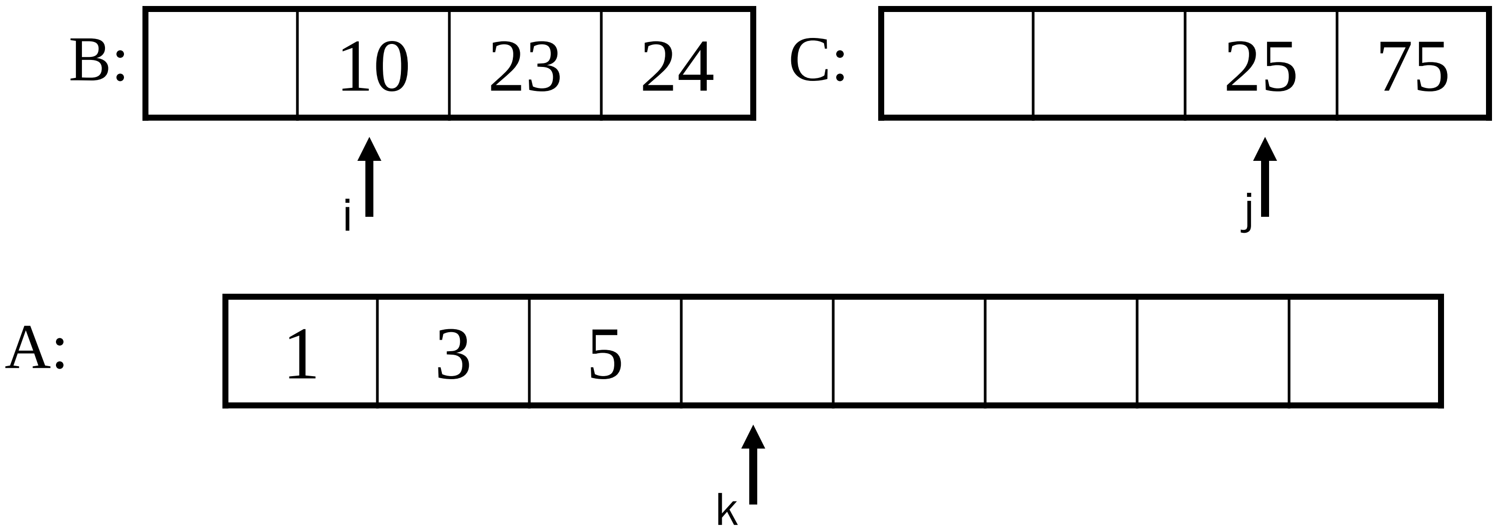
Merging (cont.)



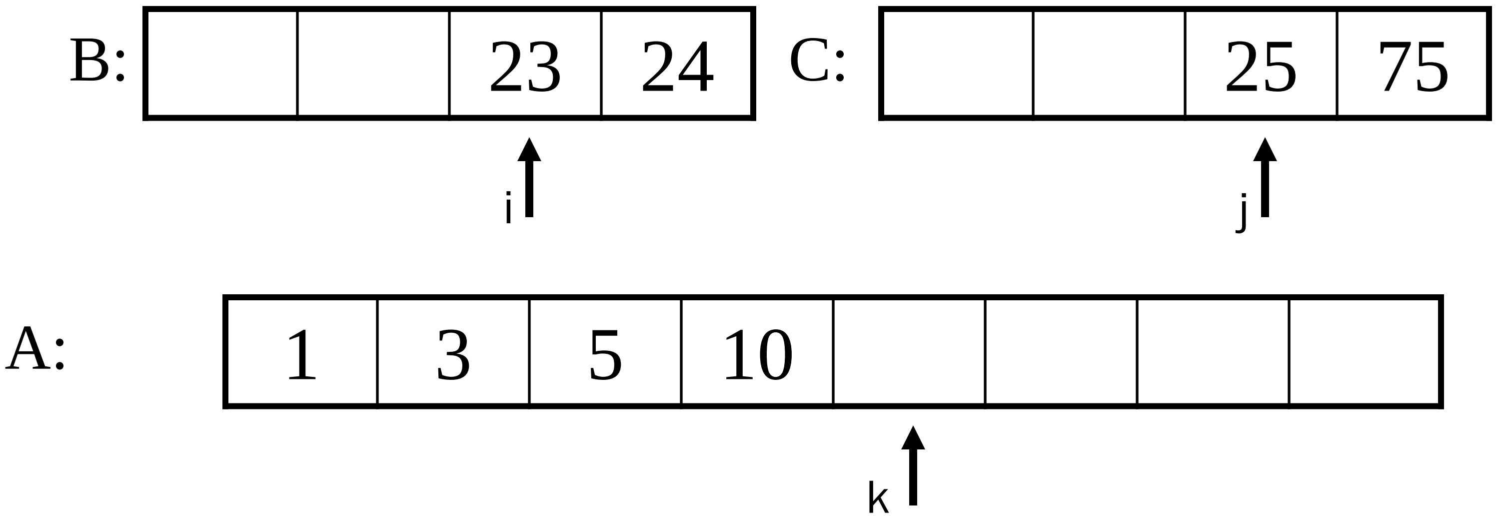
Merging (cont.)



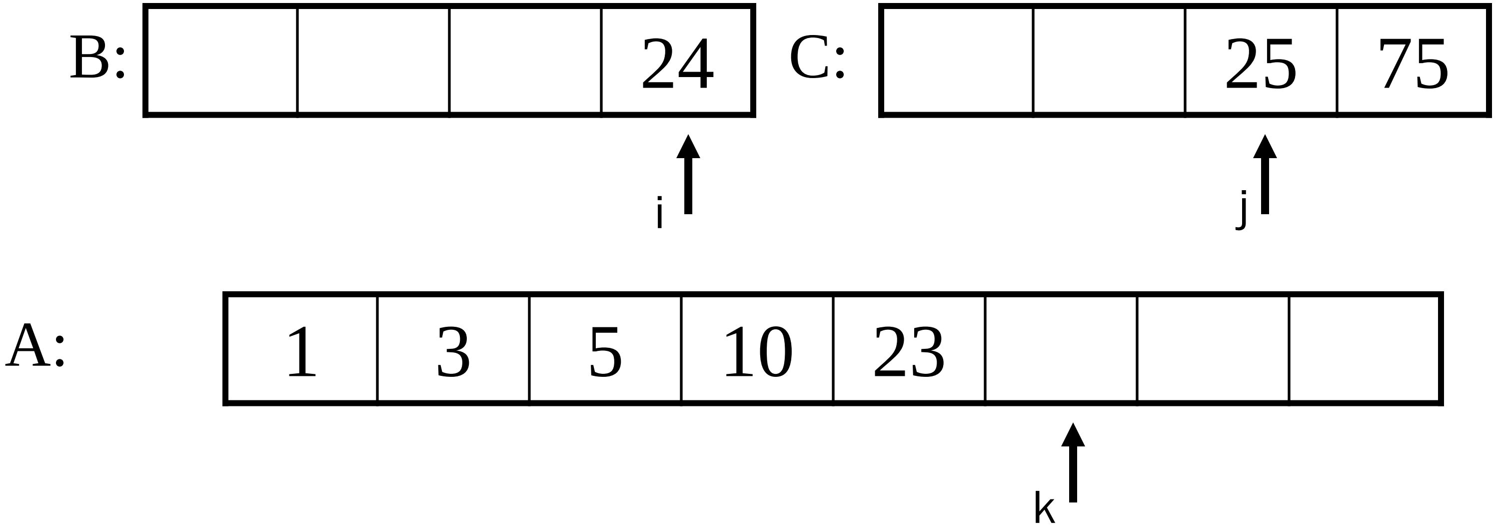
Merging (cont.)



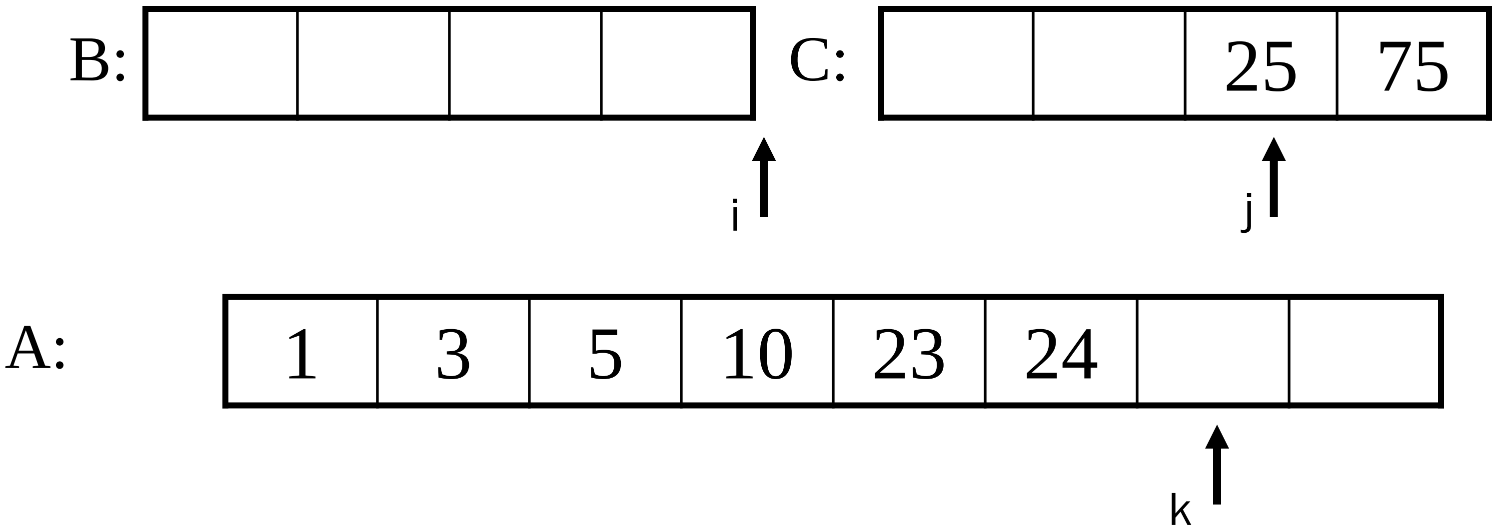
Merging (cont.)



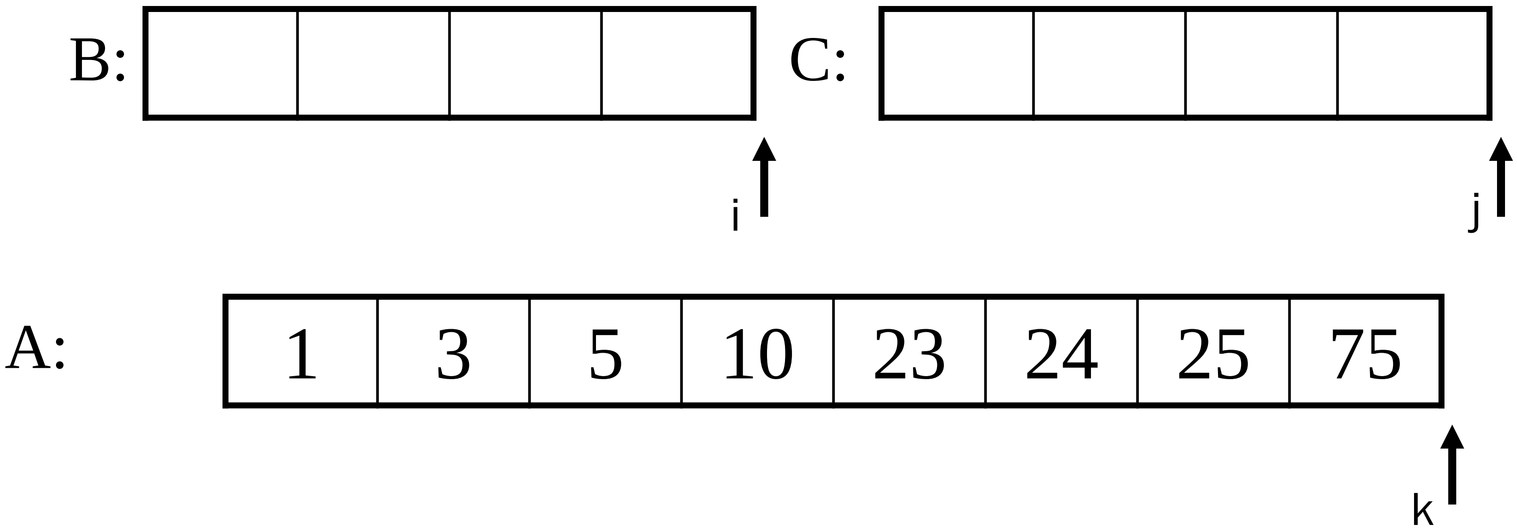
Merging (cont.)



Merging (cont.)



Merging (cont.)

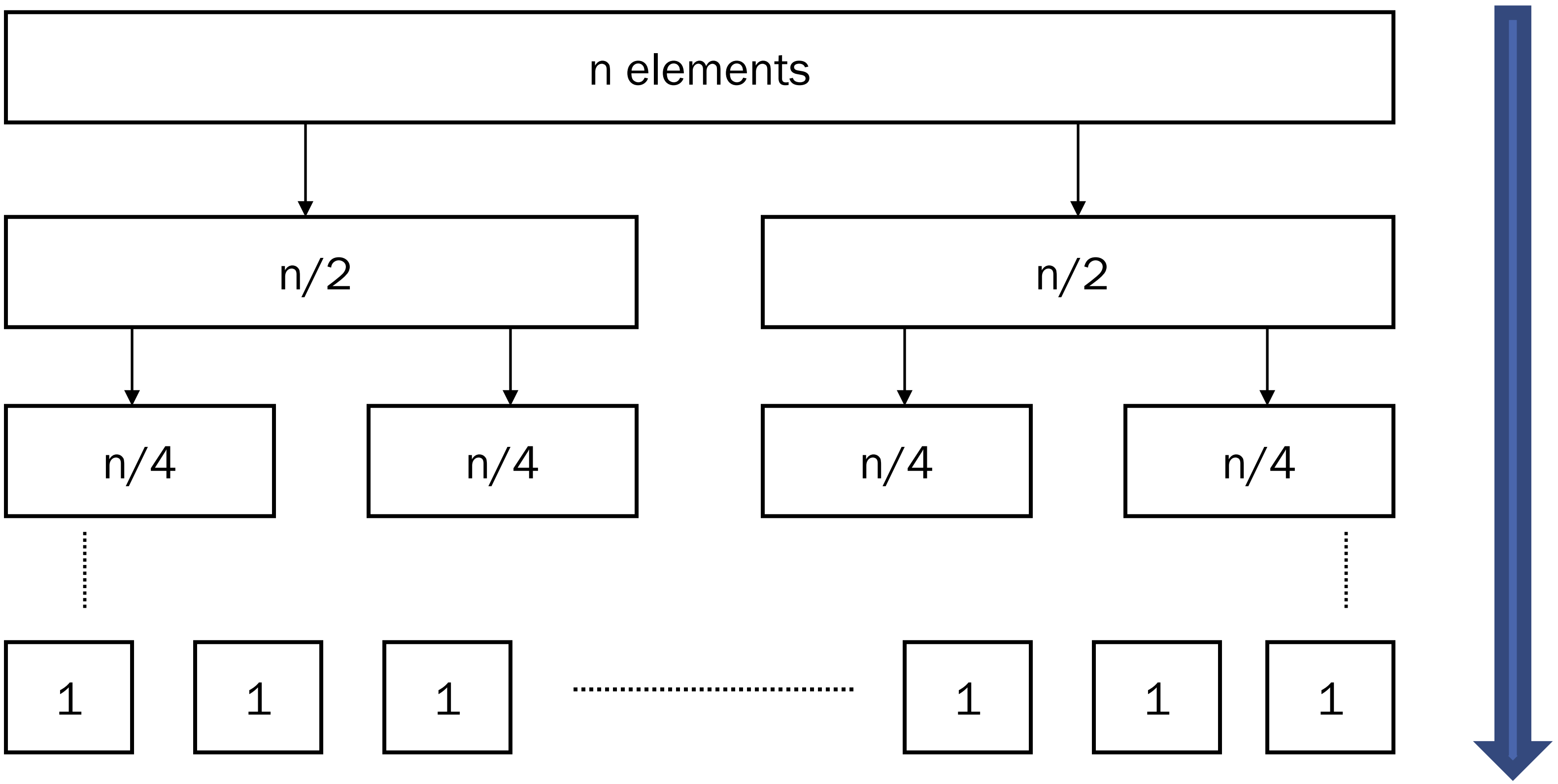


Pseudocode of Merge

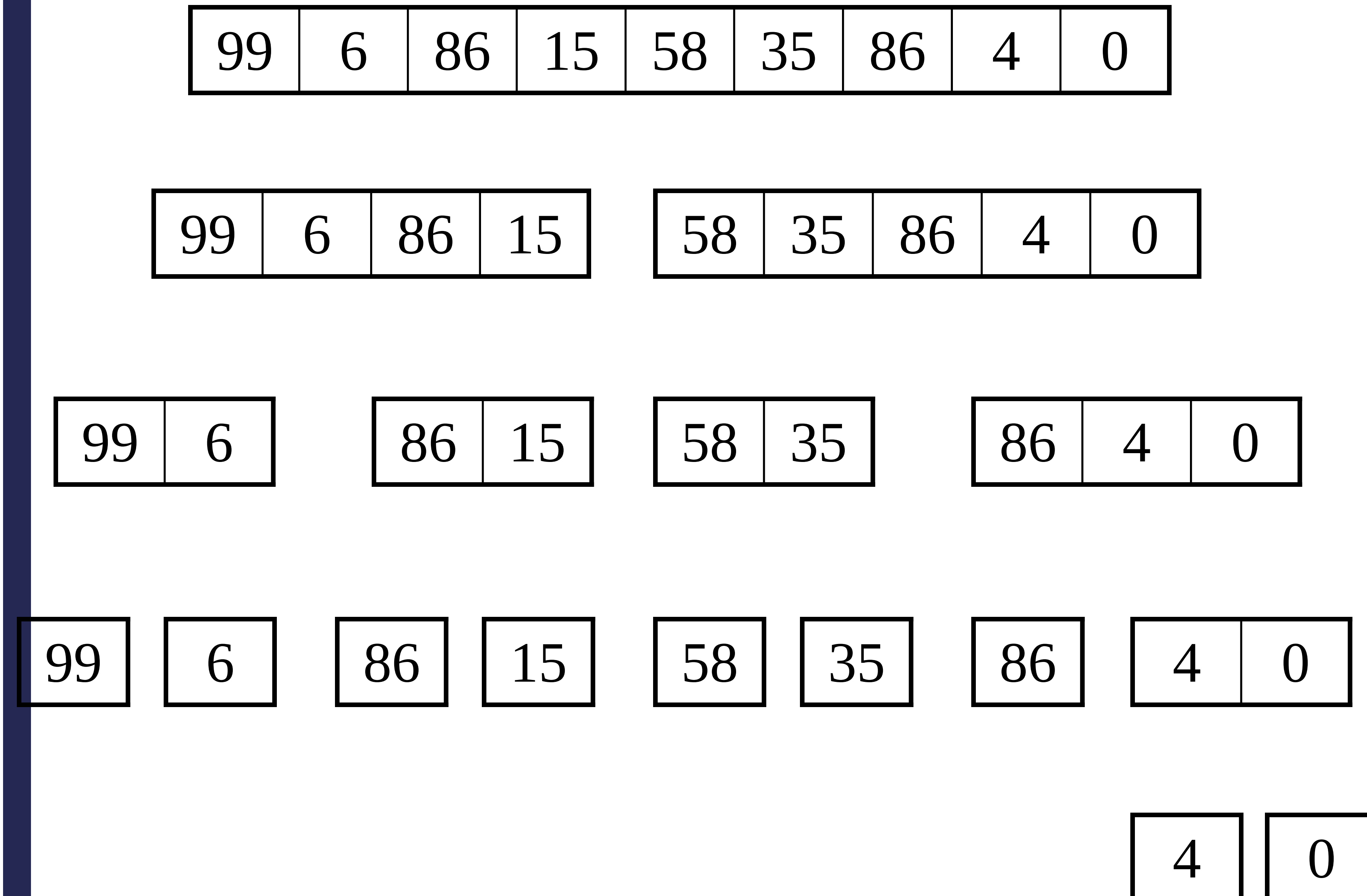
ALGORITHM *Merge*$(B[0..p-1], C[0..q-1], A[0..p+q-1])$

//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
$i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
  if $B[i] \leq C[j]$
    $A[k] \leftarrow B[i]$; $i \leftarrow i + 1$
  else $A[k] \leftarrow C[j]$; $j \leftarrow j + 1$
  $k \leftarrow k + 1$
if $i = p$
  copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

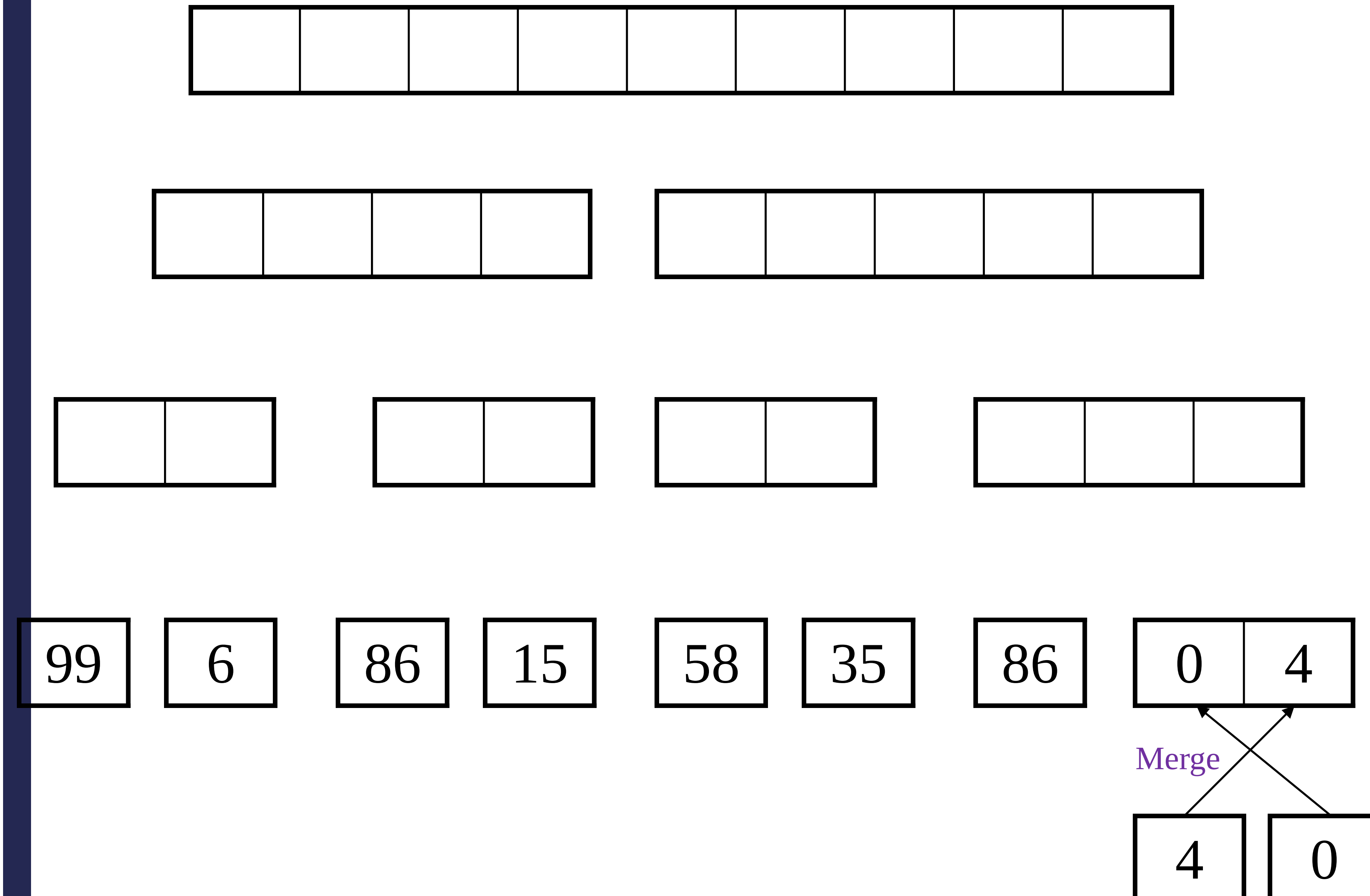
Mergesort



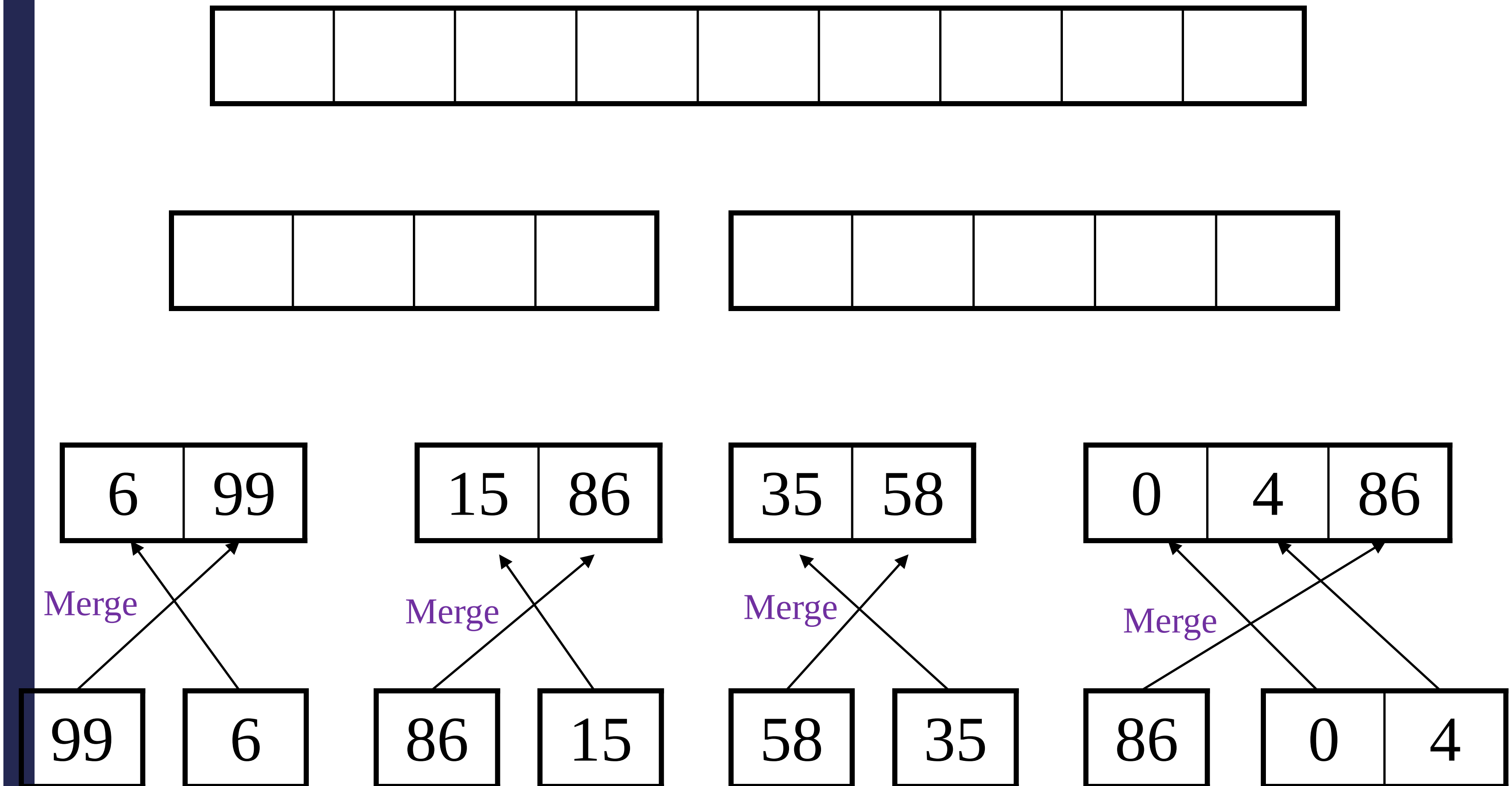
Mergesort Example



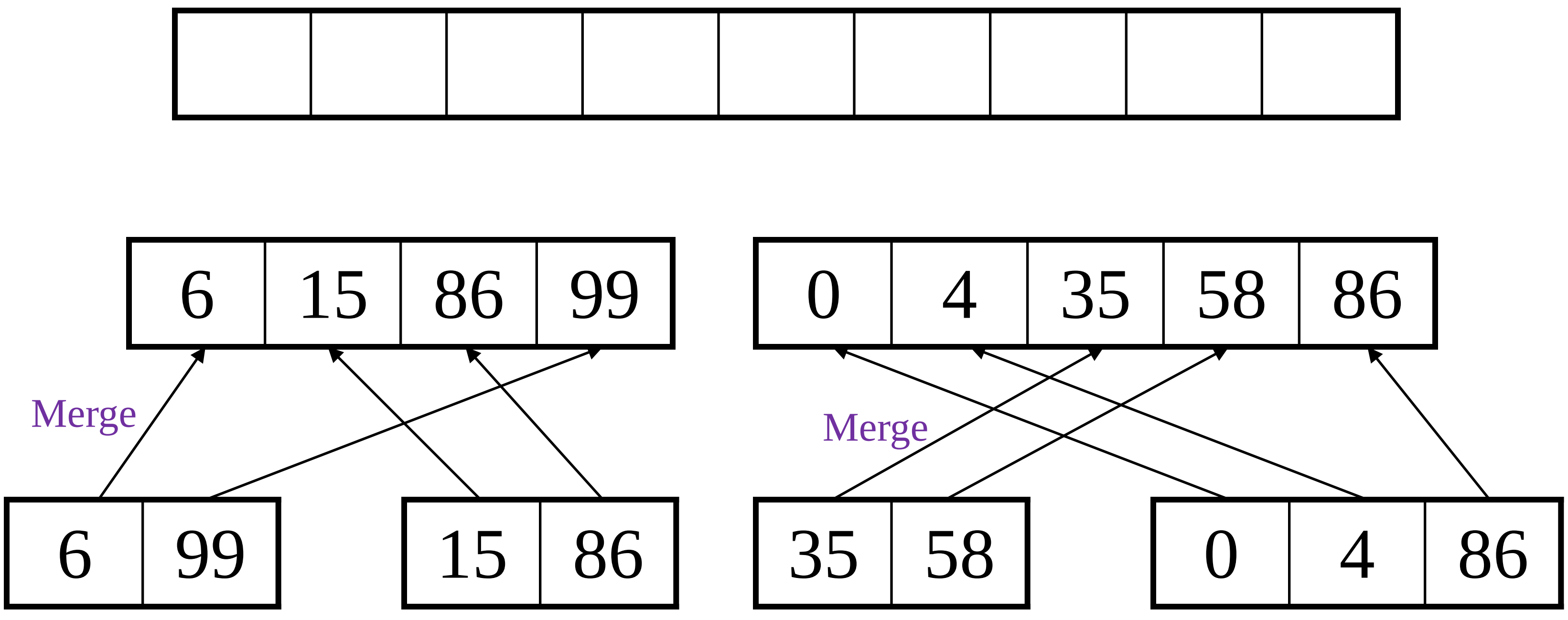
Mergesort Example



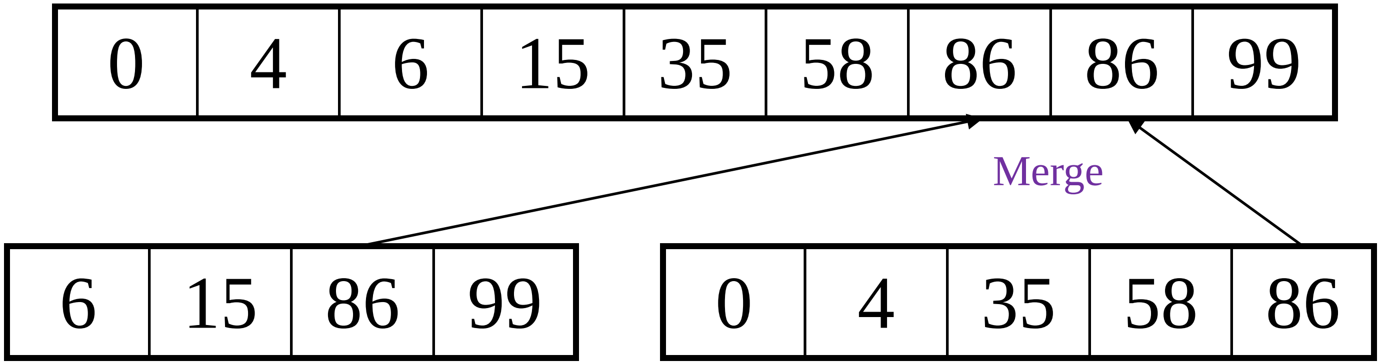
Mergesort Example



Mergesort Example



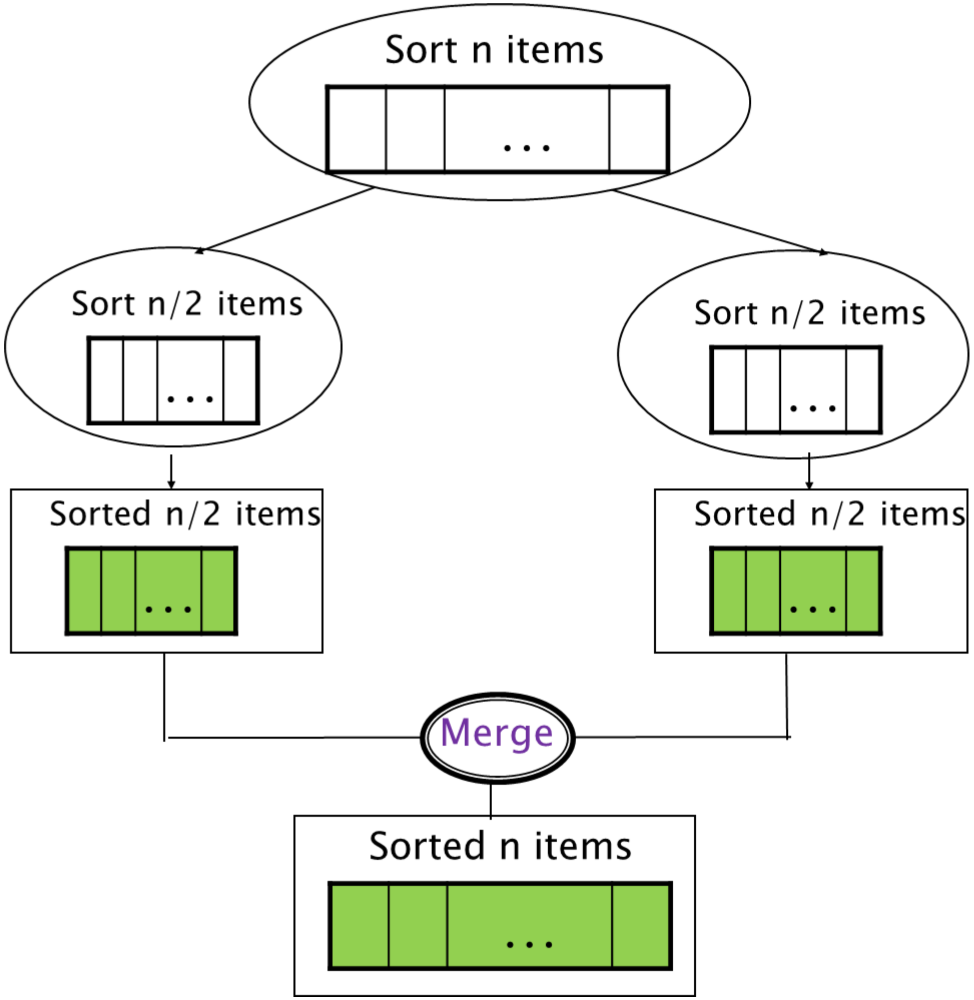
Mergesort Example



Mergesort Example

0	4	6	15	35	58	86	86	99

Running Time



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

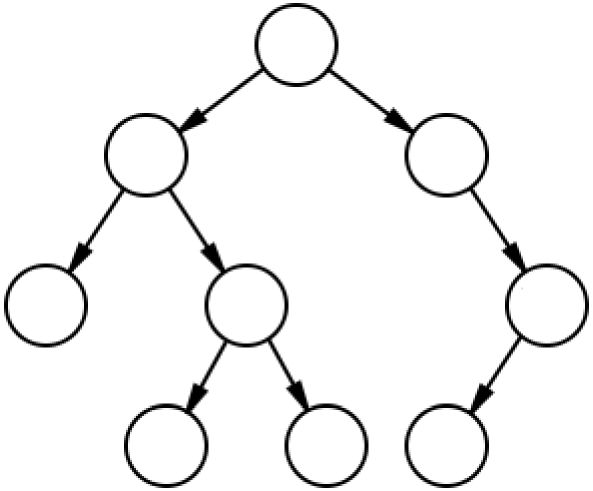
$T(n) \in O(n \log n)$

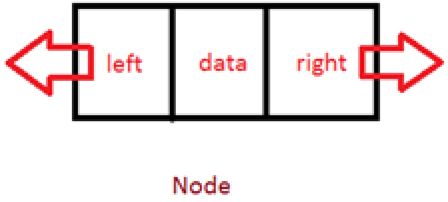
BINARY TREES

Binary tree structure

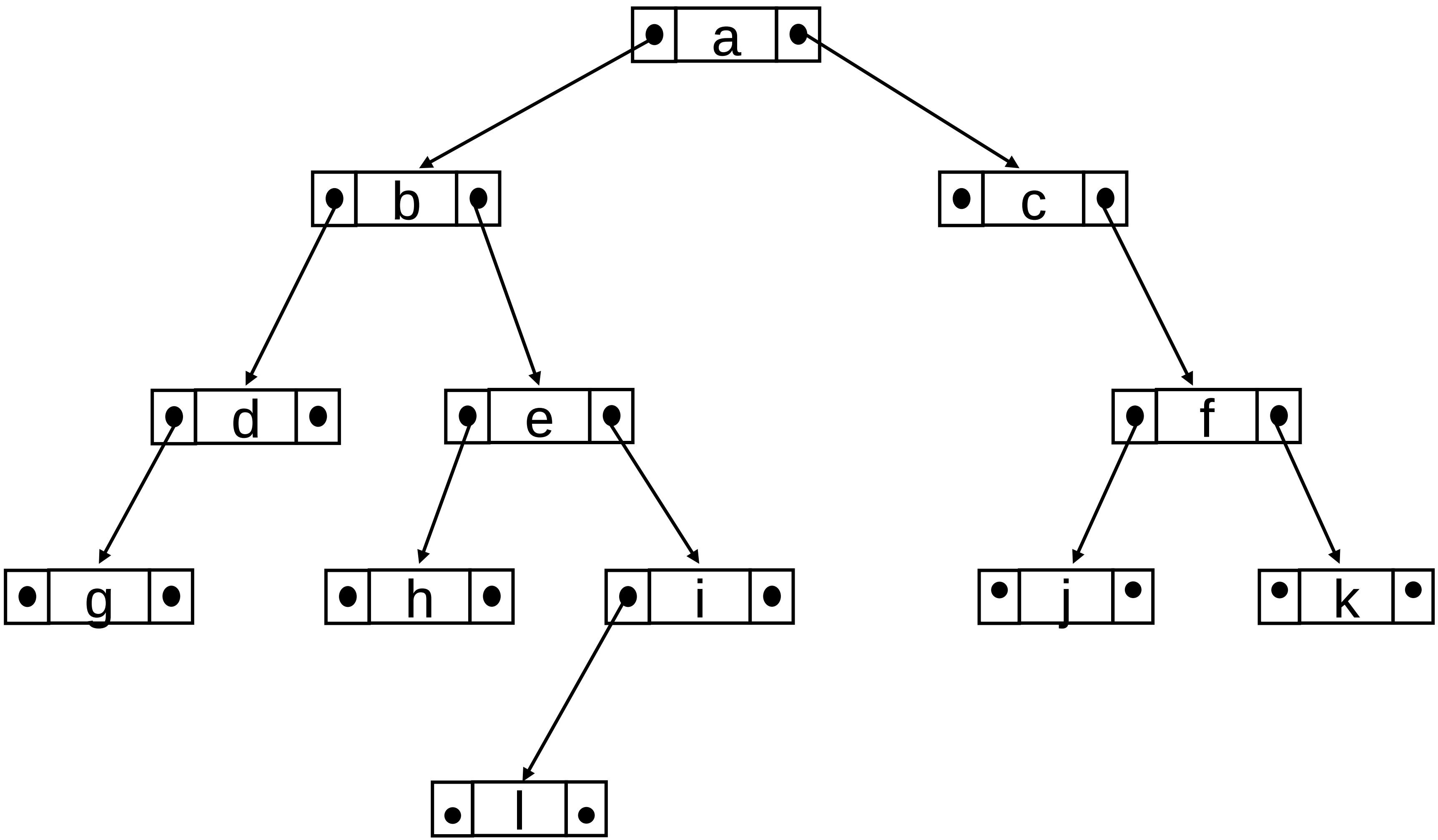
```
public class Node {
        public char data;
        public Node left;
        public Node right;

        public Node(char d) {
                data = d;
        }
}
```

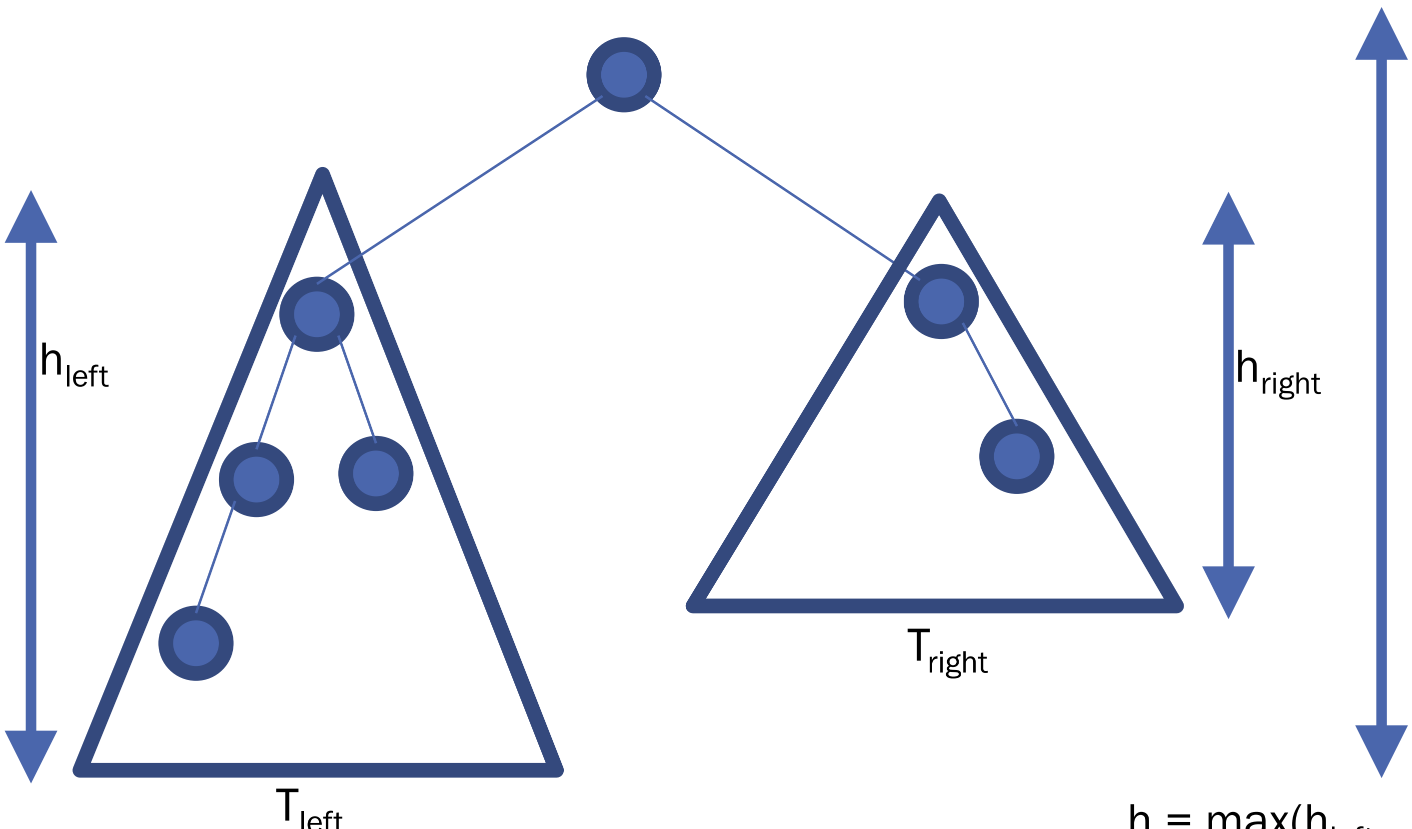




Binary tree implementation



Computing the height of a binary tree



$$h = \max(h_{left}, h_{right}) + 1$$

Computing the height of a binary tree

ALGORITHM $Height(T)$

//Computes recursively the height of a binary tree
//Input: A binary tree T
//Output: The height of T
if $T = \varnothing$ **return** -1
else return $\max\{Height(T_{left}), Height(T_{right})\} + 1$

Compute the number of leaves

Algorithm $LeafCounter(T)$
//Computes recursively the number of leaves in a binary tree
//Input: A binary tree T
//Output: The number of leaves in T
if $T = \varnothing$ **return** 0 //empty tree
else if $T_L = \varnothing$ **and** $T_R = \varnothing$ **return** 1 //one-node tree
else return $LeafCounter(T_L) + LeafCounter(T_R)$ //general case

Try it/ homework

1. Chapter 5.1, page 174, questions 1, 6
2. Chapter 5.3, page 185, question 2
3. Implement a function to check if a tree is balanced. A balanced tree is defined to be a tree such that no two leaf nodes differ in distance from the root by more than one.