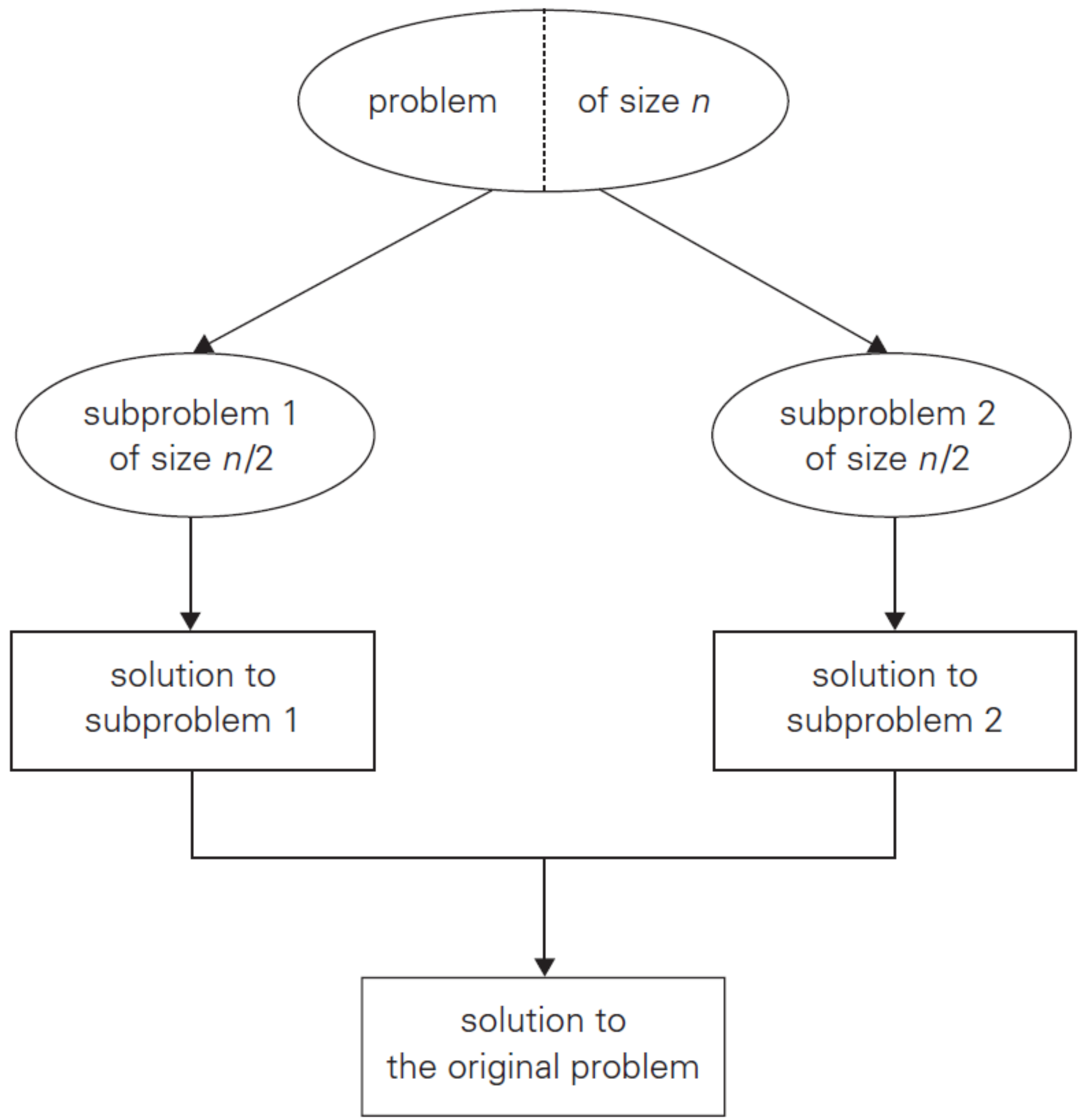
Divide and Conquer

(Chapter 5)

This week:

- Divide and Conquer technique
- Count a specific key in an array
- Master theorem
- Merge sort

Divide and Conquer technique

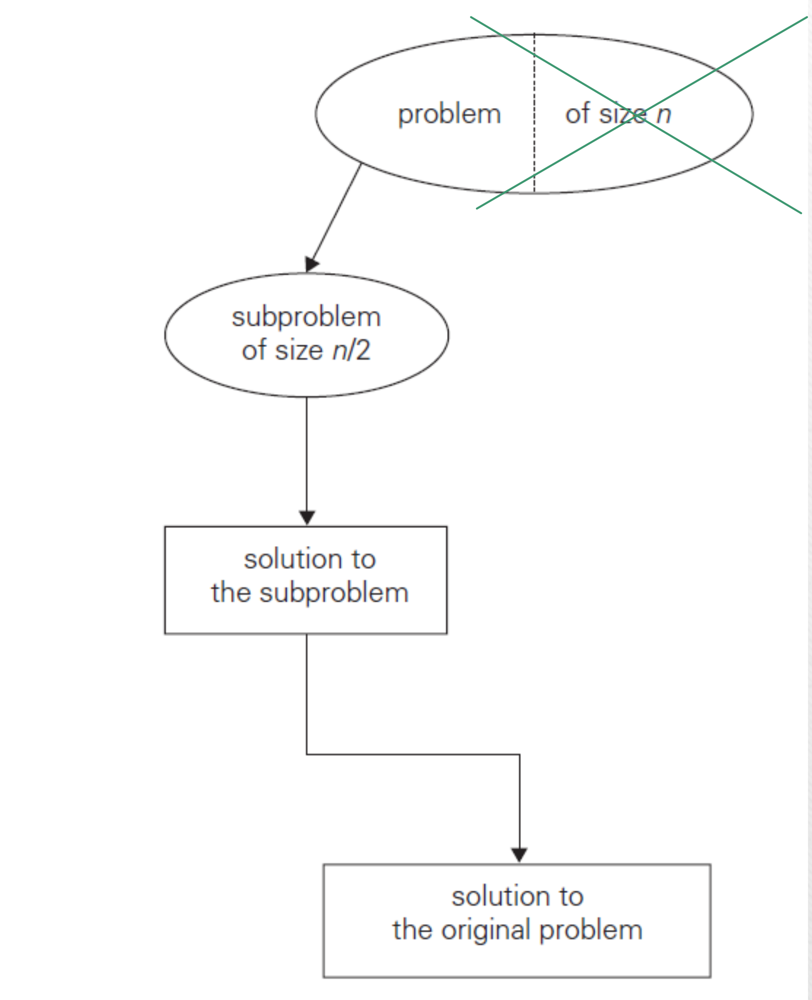


Divide and Conquer technique

A well known algorithm design technique:

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

Decrease and Conquer (last week)



A Natural Question

- How is this different from decrease and conquer technique
- 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

This week:

- Divide and Conquer technique
- Count a specific key in an array
- Master theorem
- Merge sort

Count a specific key in an array

Problem:

- Count the number of times a specific key occurs in an array.

For example:

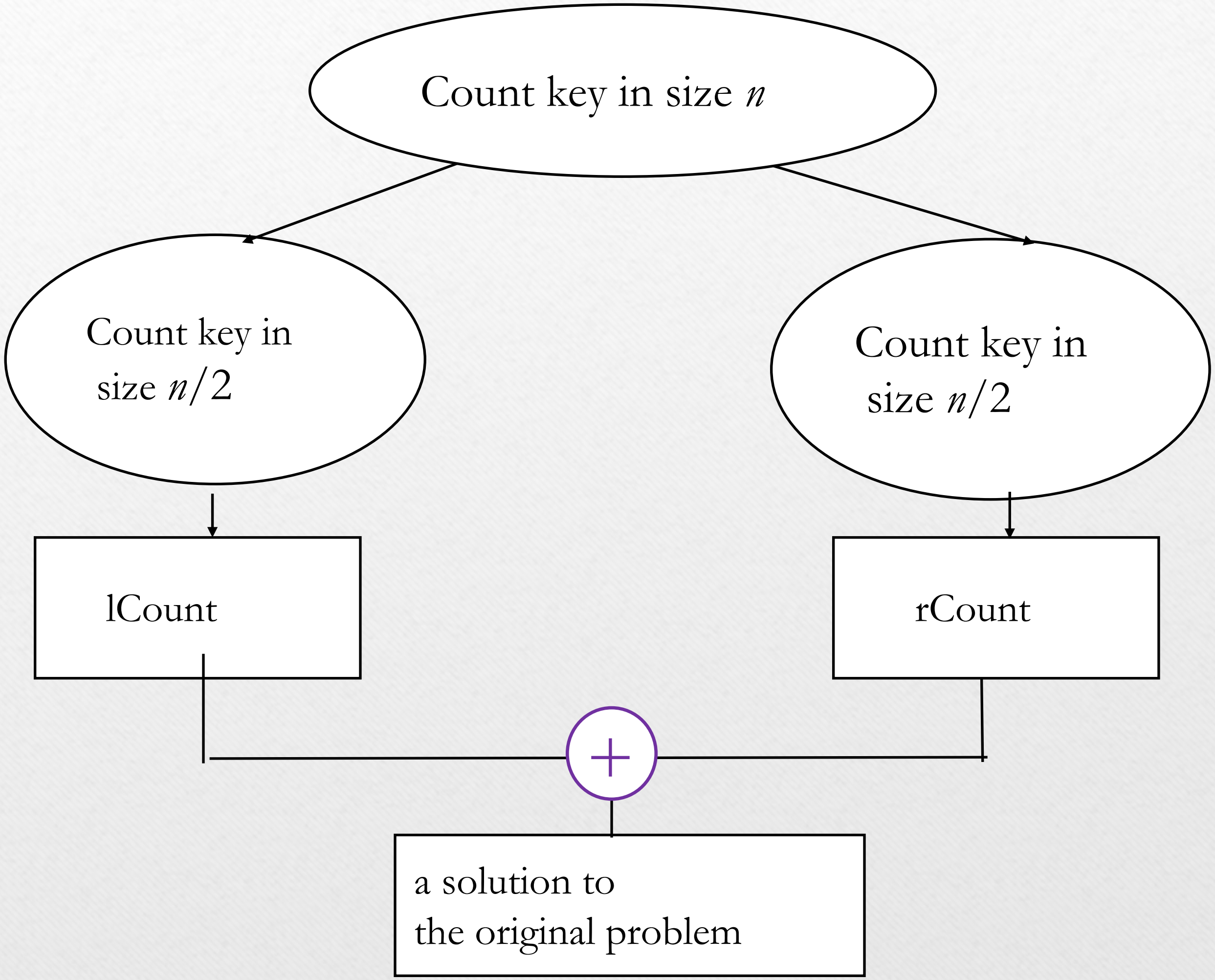
- If input array is A=[2,7,6,6,2,4,6,9,2] and key=6…

... should return the value 3.

Design an algorithm using divide and conquer technique

Whiteboard

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 and R (L ≤ R)
//Output: A count of the number of time Key exists in A[L..R]
1.  if L = R
2.      if (A[L] = Key) return 1
3.      else return 0
4.  else
5.      lCount = CountKey(A[],      L,      ⌊(L+R)/2⌋, Key)
6.      rCount = CountKey(A[], ⌊(L+R)/2⌋+1,     R,     Key)
7.      return lCount + rCount
```


Big O? All in good time…

Count a specific key in an array

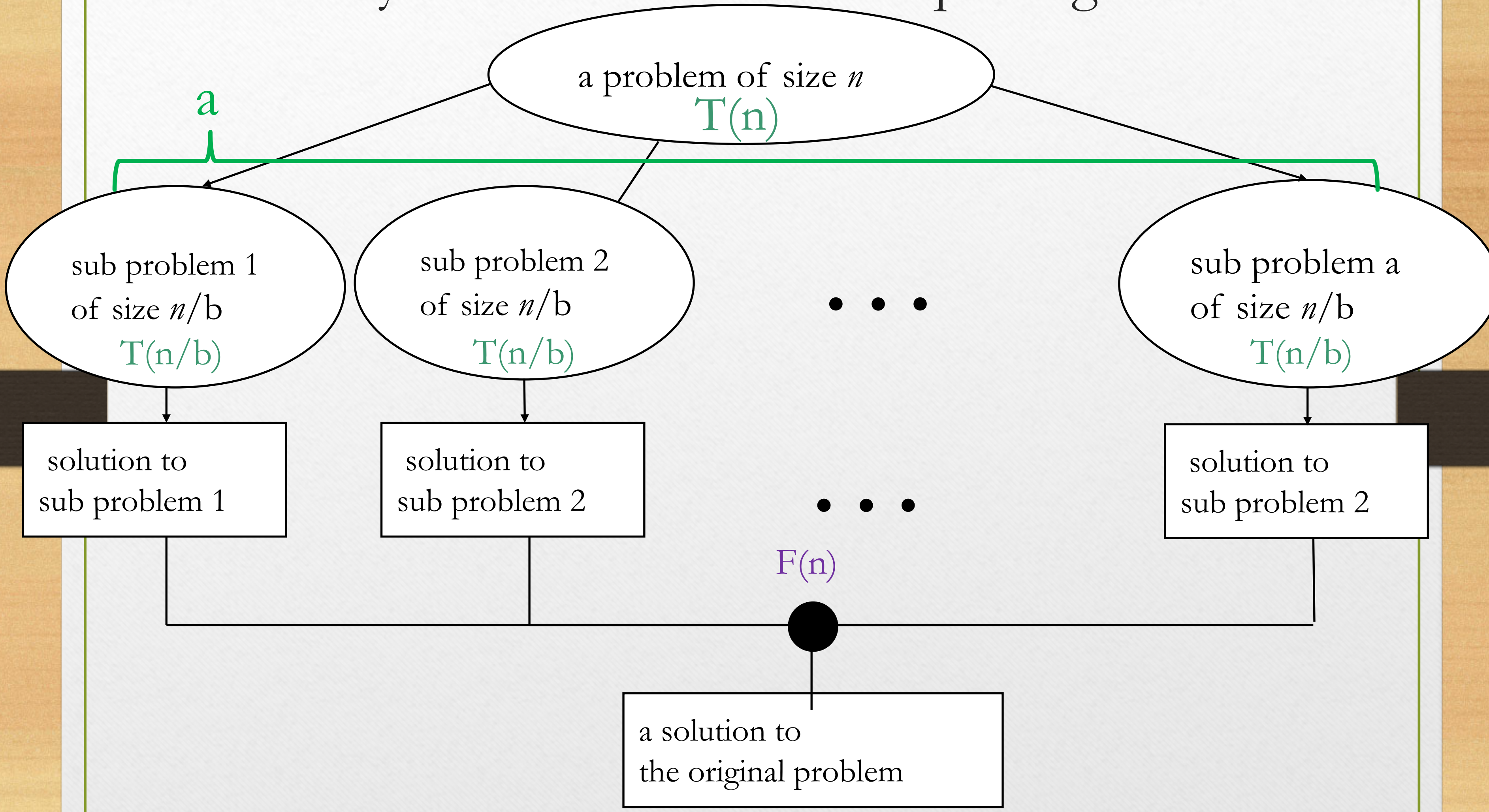
- *CountKey* looks familiar…
  - What's the difference between *Binary Search and CountKey?*

- We have to searched both sides
  - In Binary Search, one half gets ignored if out of bounds
  - In CountKey, both sides are searched to get sum

This week:

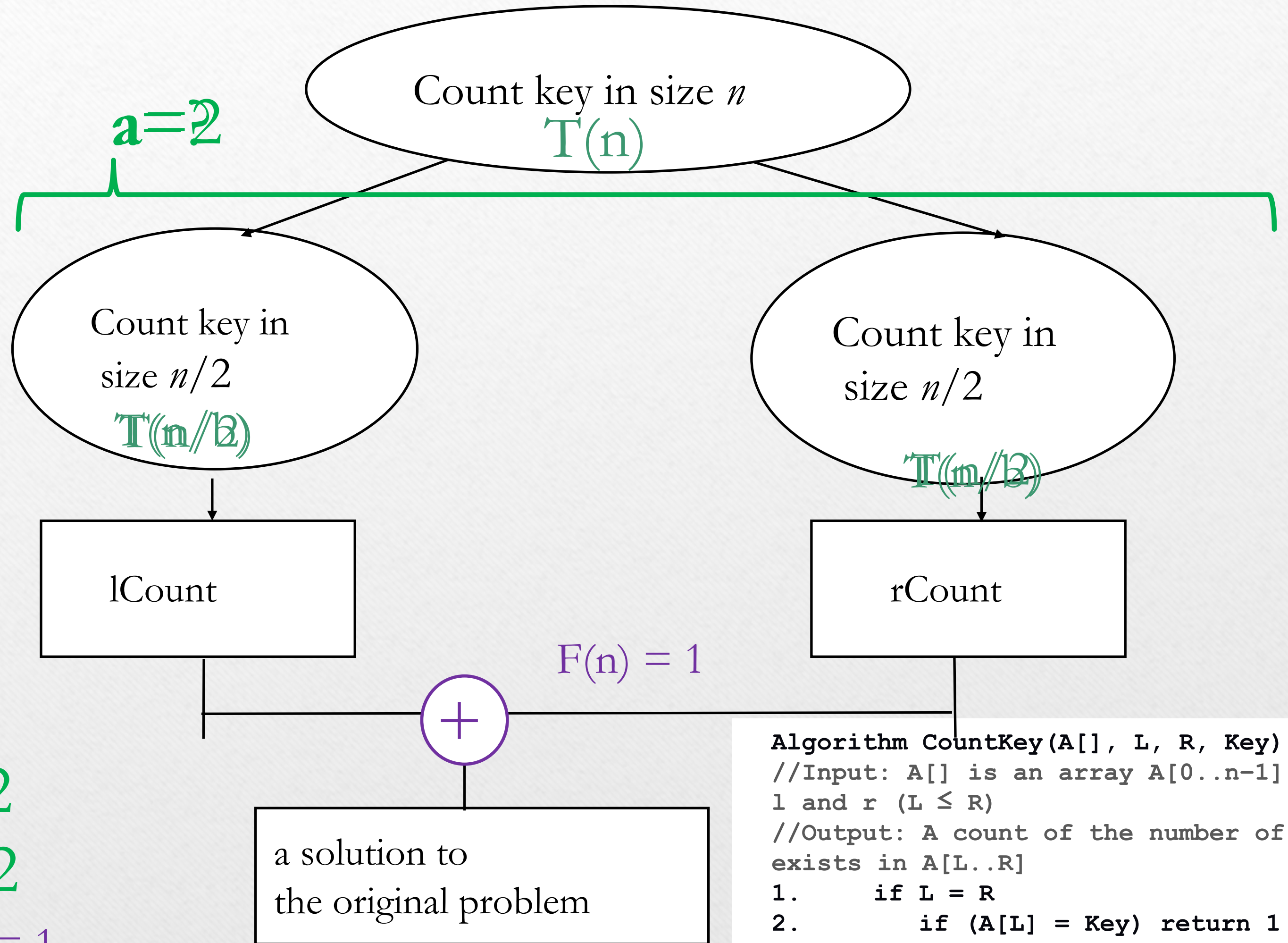
- Divide and Conquer technique
- Count a specific key in an array
- Master theorem
- Merge sort

Analysis of a divide and conquer algorithm



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

Analysis of Count a specific key in an array



a=2

b=2

F(n) = 1

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

T(n/2)

```
Algorithm CountKey(A[], L, R, Key)
//Input: A[] is an array A[0..n−1] from indices
l and 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
```

F(n) = 1

Master Theorem

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

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

Add to Cheat Sheet

Count Keys: $T(n) = 2T(n/2) + 1 \Rightarrow T(n) \in ?$

$a = 2$
$b = 2$

$\longrightarrow n^{\log_b a} \longrightarrow n^{\log_2 2} \longrightarrow n$

$F(n) = 1$

$\longrightarrow$ $T(n) \in O(n^{\log_b a})$
$T(n) \in O(n^{1})$

Master Theorem

```
binarySearch(a[], k, start, end)
    if end < start
        return not found
    middle ← floor((start+end)/2)
    if k = a[middle]
        return found
    if k > a[middle]
        return binarySearch(a[], k, middle+1, end)
    else if k < a[middle]
        return binarySearch(a[], k, start, middle-1)
```

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

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

Binary Search: $T(n) = T(n/2) + 1 \Rightarrow T(n) \in ?$

a = 1
b = 2

$n^{\log_b a}$ $n^{\log_2 1}$ 1^{2}

F(n) = 1

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

Master Theorem

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

```
Alg(n)
    Alg(n/2)
    Alg(n/2)
    Alg(n/2)
    Alg(n/2)

    for each i in n
      for each j in n
        for each k in n
          do something
```

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

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

$a = 4$
$b = 2$

$\Rightarrow n^{\log_b a} \Rightarrow n^{\log_2 4} \Rightarrow n^2$

$F(n) = n^3$

$\Rightarrow T(n) \in O(n^3)$

Master theorem

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

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

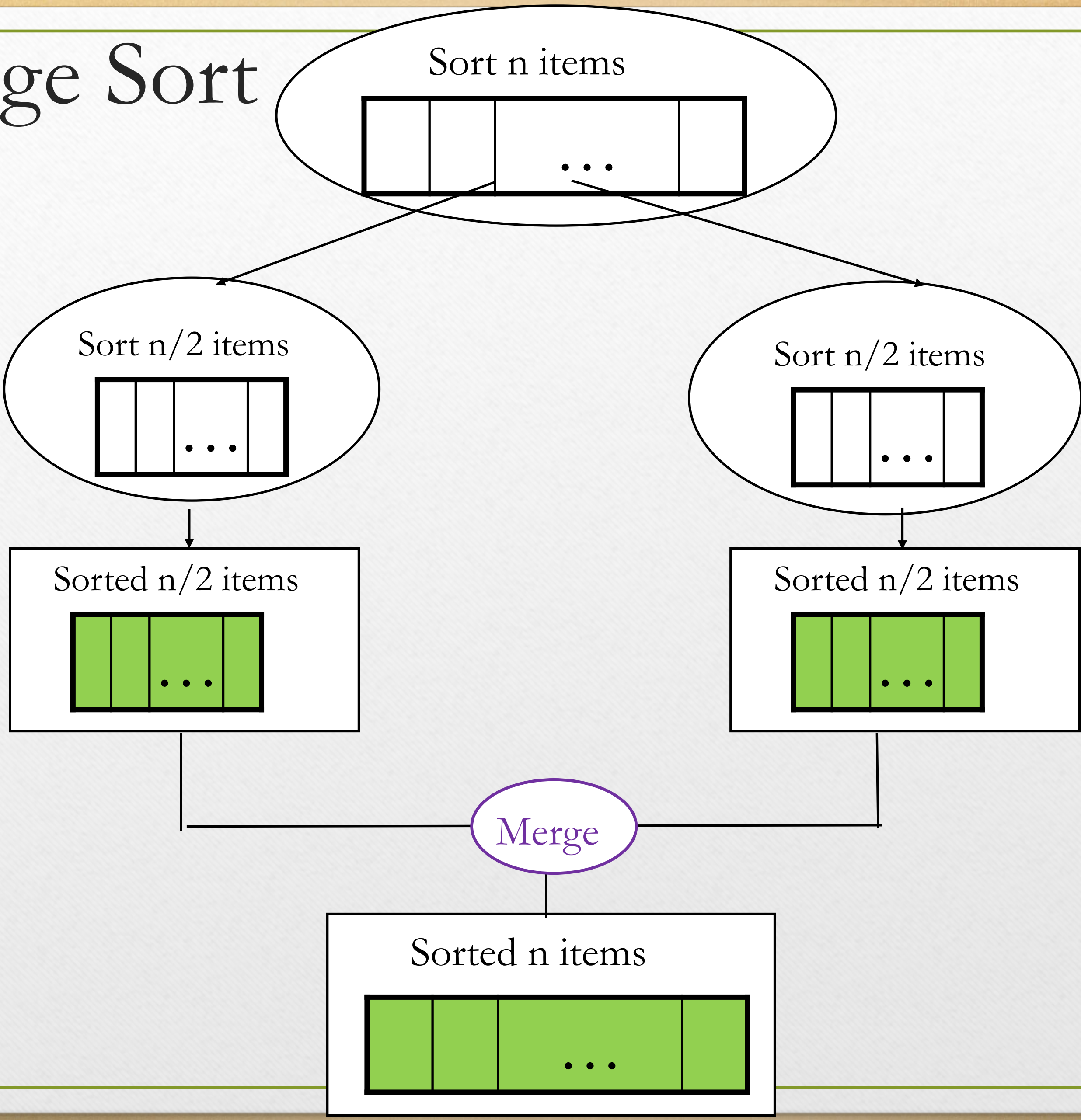
Example 3: $T(n) = 4T(n/2) + n^2 \Rightarrow T(n) \in ?$

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

This week:

- Divide and Conquer technique
- Count a specific key in an array
- Master theorem
- Merge sort

Merge Sort



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)

Merging

Implementation of Merge(B,C,A)

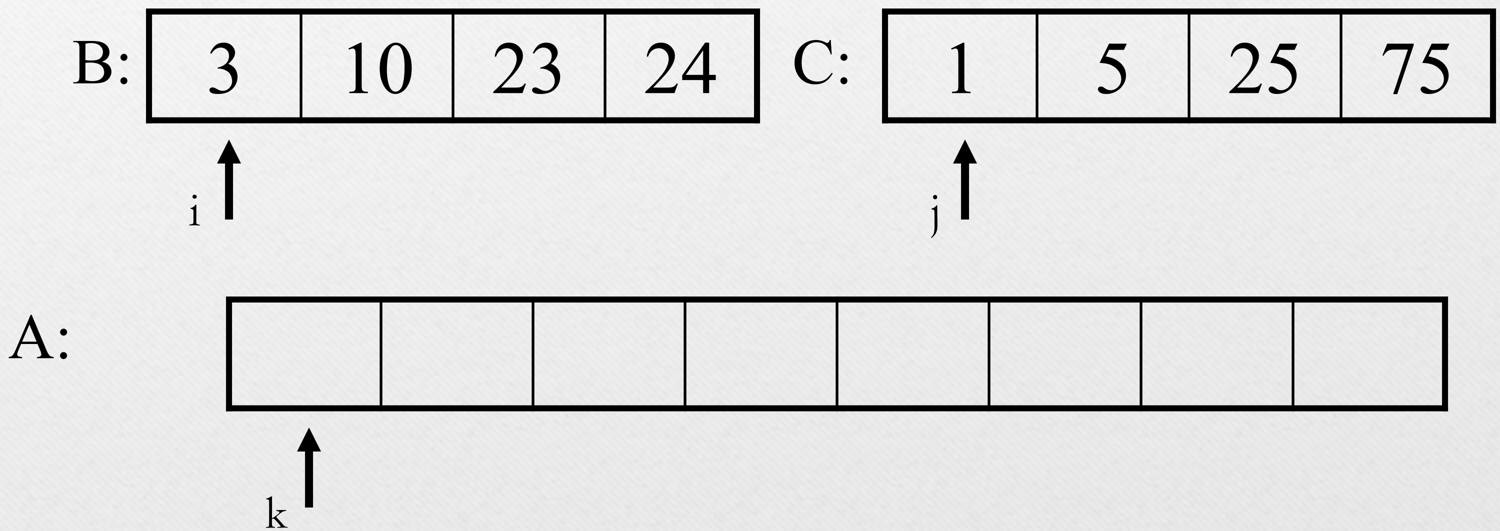
- Important two sorted arrays (B & C) and merge it into an empty array (A)

- Example:

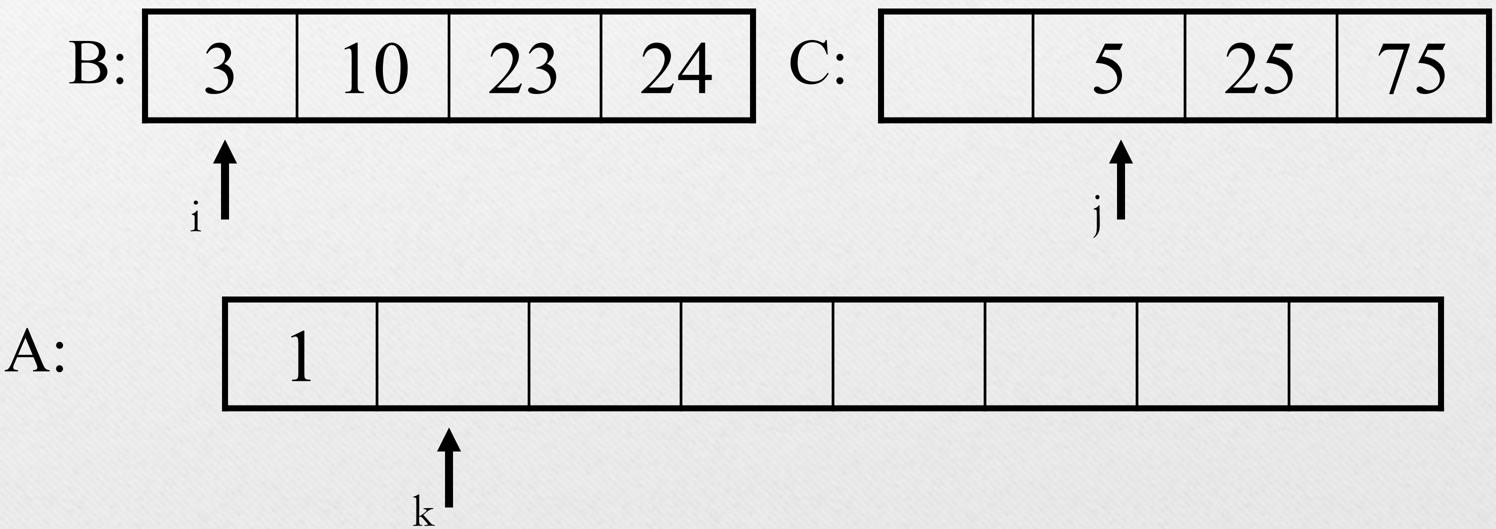
B = { 3 8 9 }   C = { 1 5 7 }

merge(B, C, A) = A ={ 1 3 5 7 8 9 }

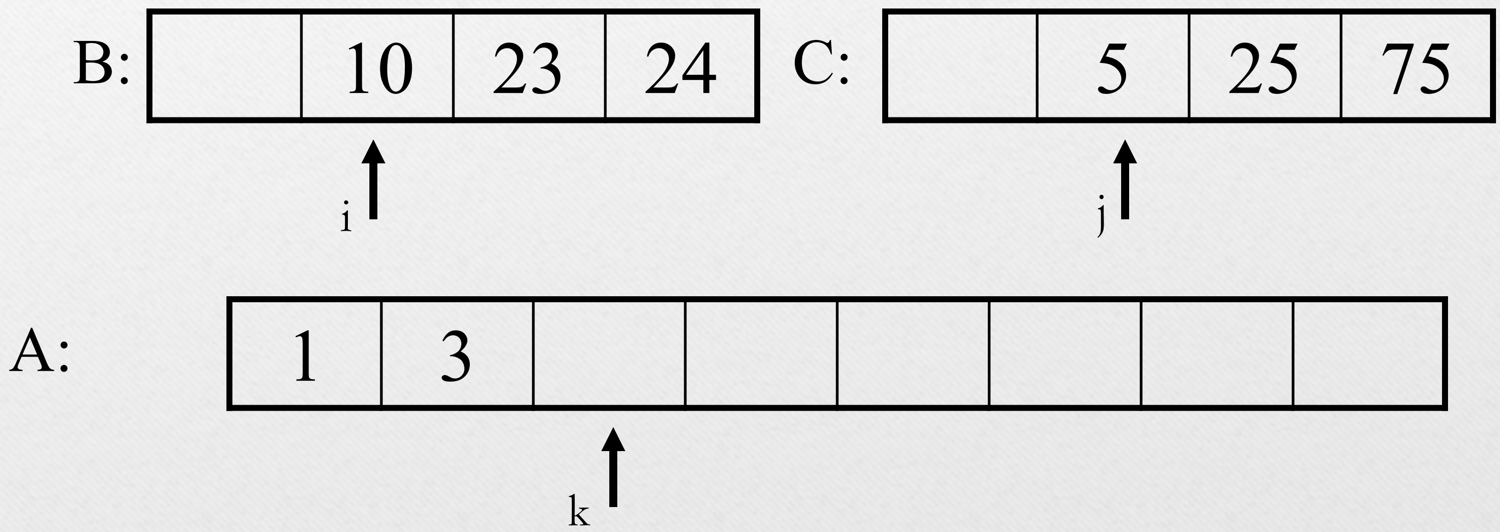
Merging (cont.)



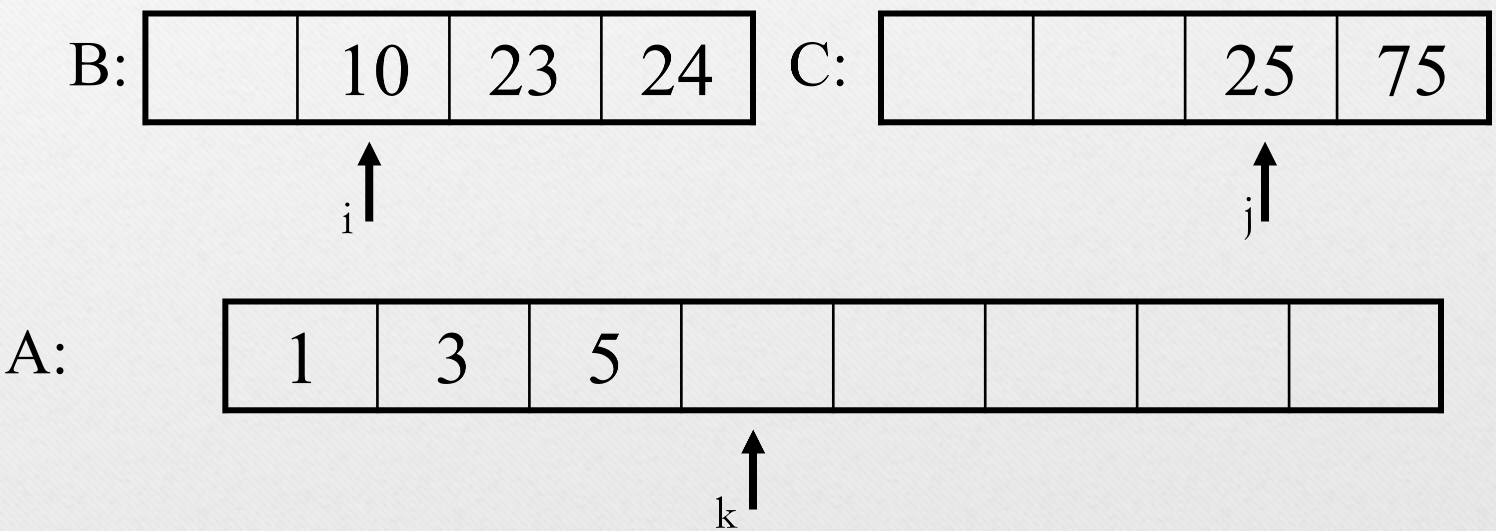
Merging (cont.)



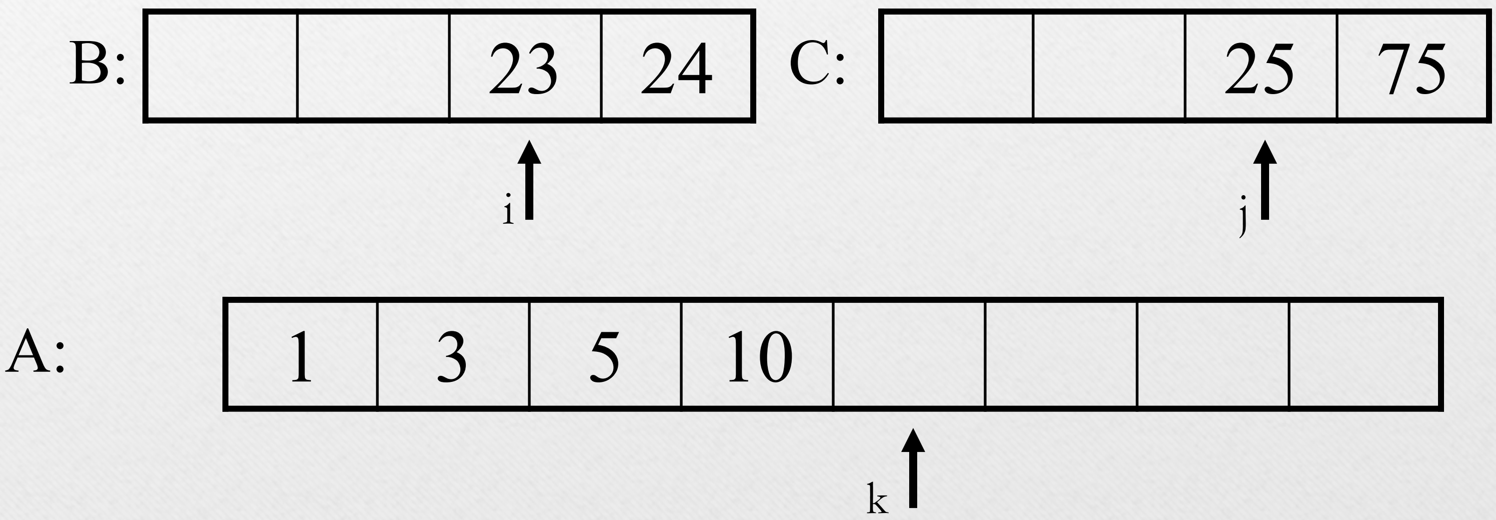
Merging (cont.)



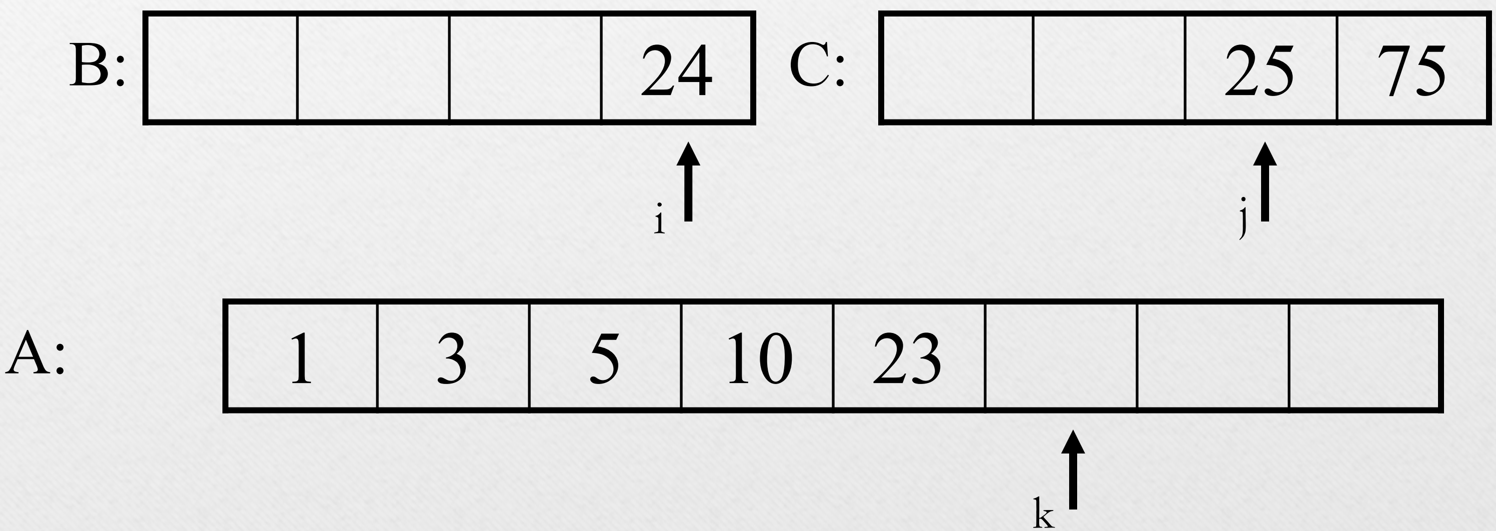
Merging (cont.)



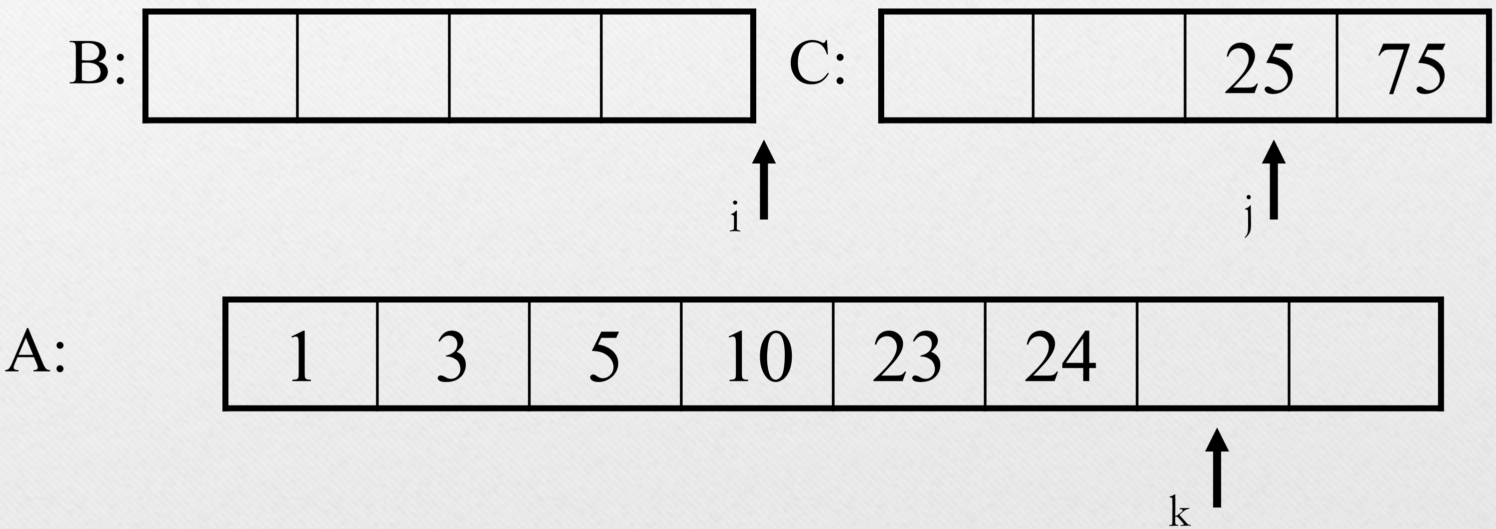
Merging (cont.)



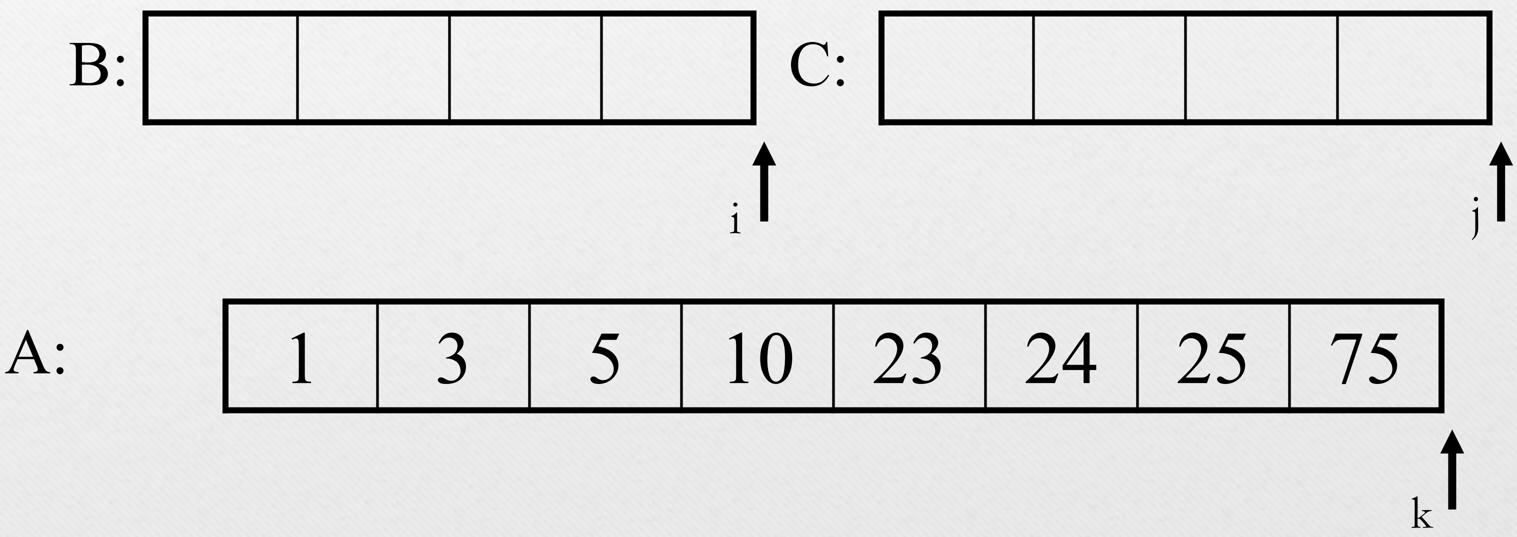
Merging (cont.)



Merging (cont.)



Merging (cont.)



Merging (cont.)

- Must put the next-smallest element into the merged list at each point
- Each next-smallest could come from either list

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 $i < p$ or $j < q$ **do**
  if $B[i] \leq C[j]$
    $A[k] \leftarrow B[i]$; $i \leftarrow i + 1$
  else $A[k] \leftarrow C[j]$; $j \leftarrow j + 1$
  $k \leftarrow k + 1$

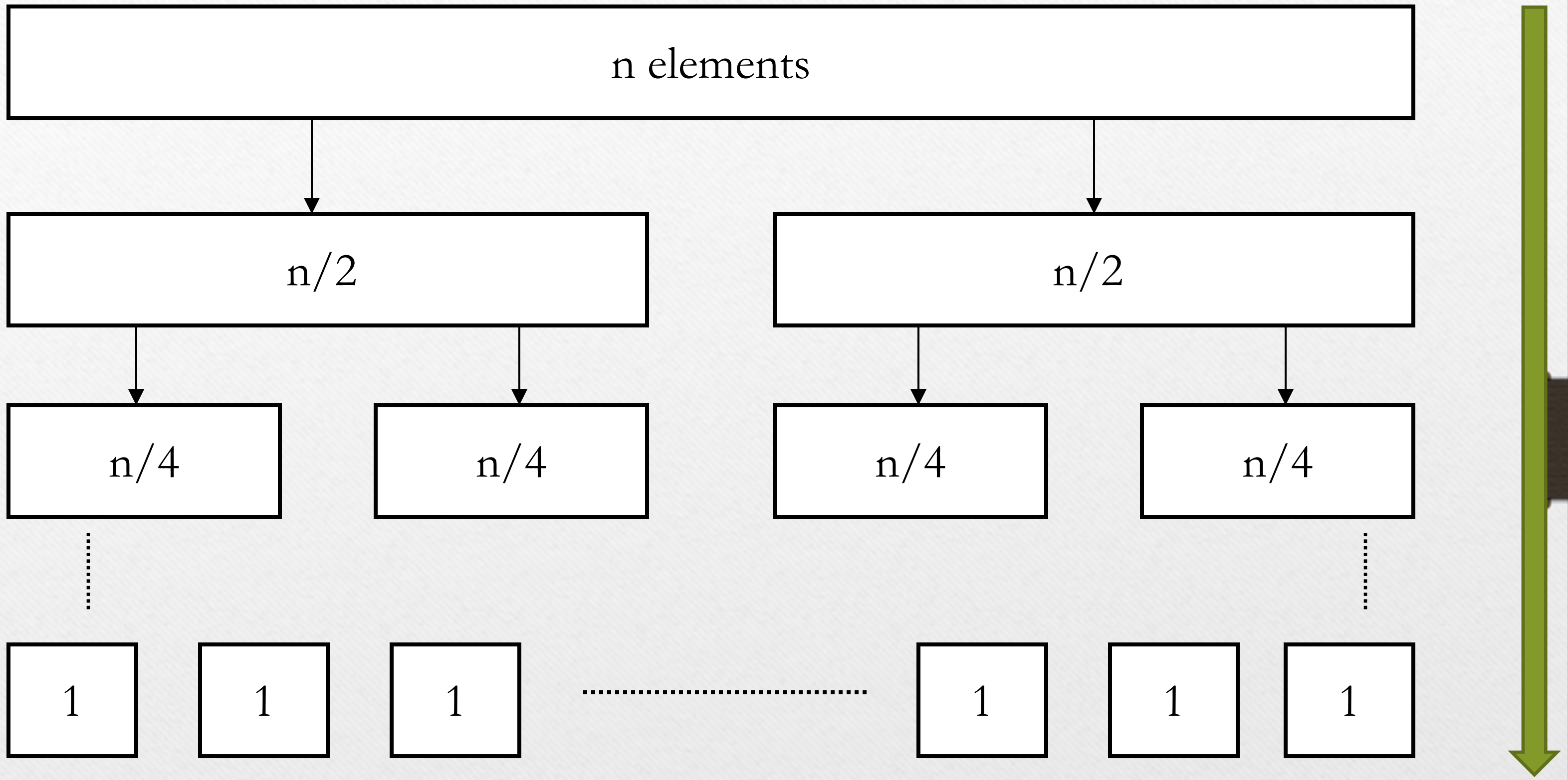
Can this be optimized? (Hint: i or j maxes out)

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 $A[k..p+q-1]$

Is this needed?


n elements
n/2
n/2
n/4
n/4
n/4
n/4
1
1
1
1
1
1

Merge Sort Example

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

99	6

86	15

58	35

86	4	0

99

6

86

15

58

35

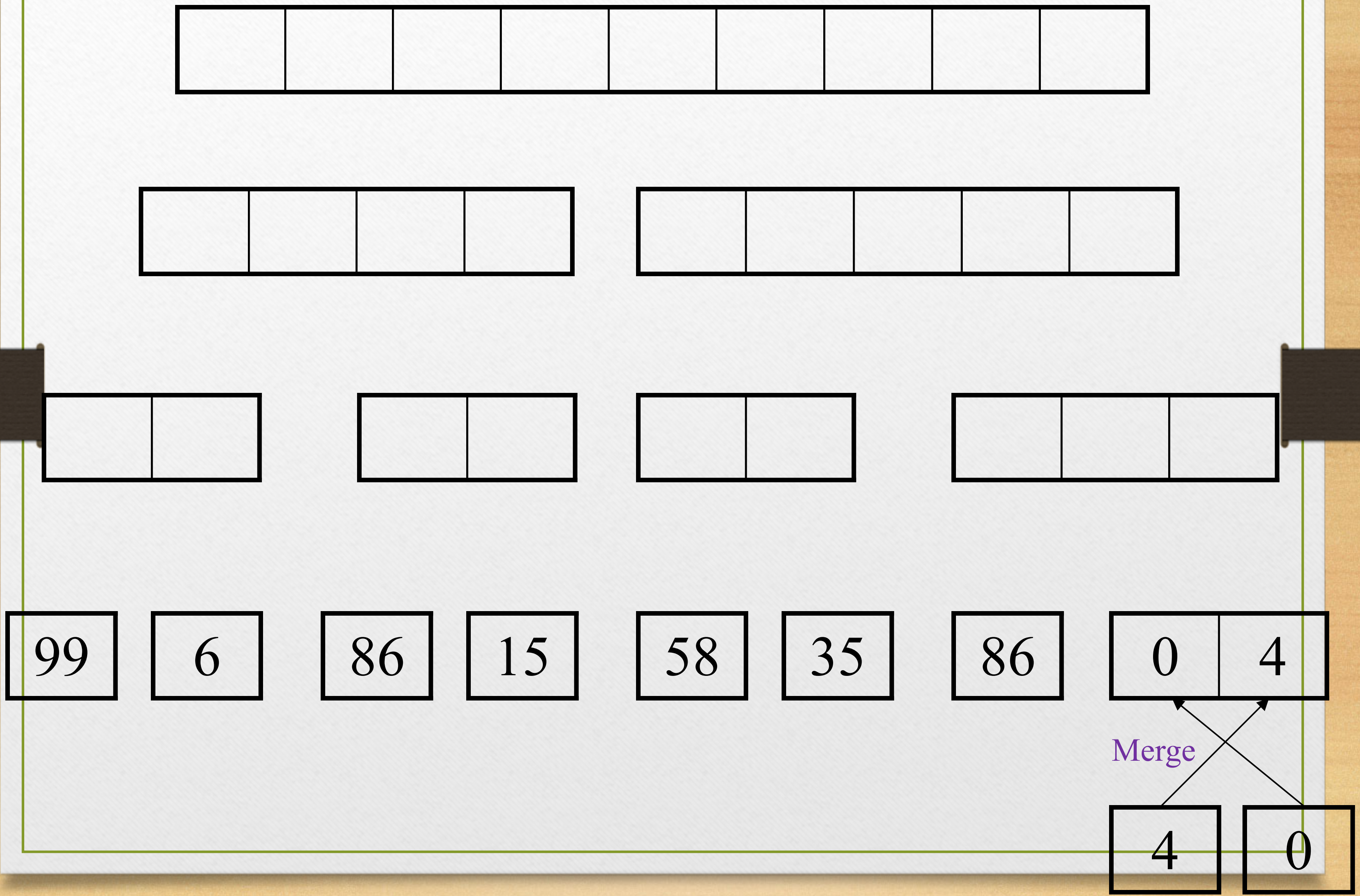
86

4	0

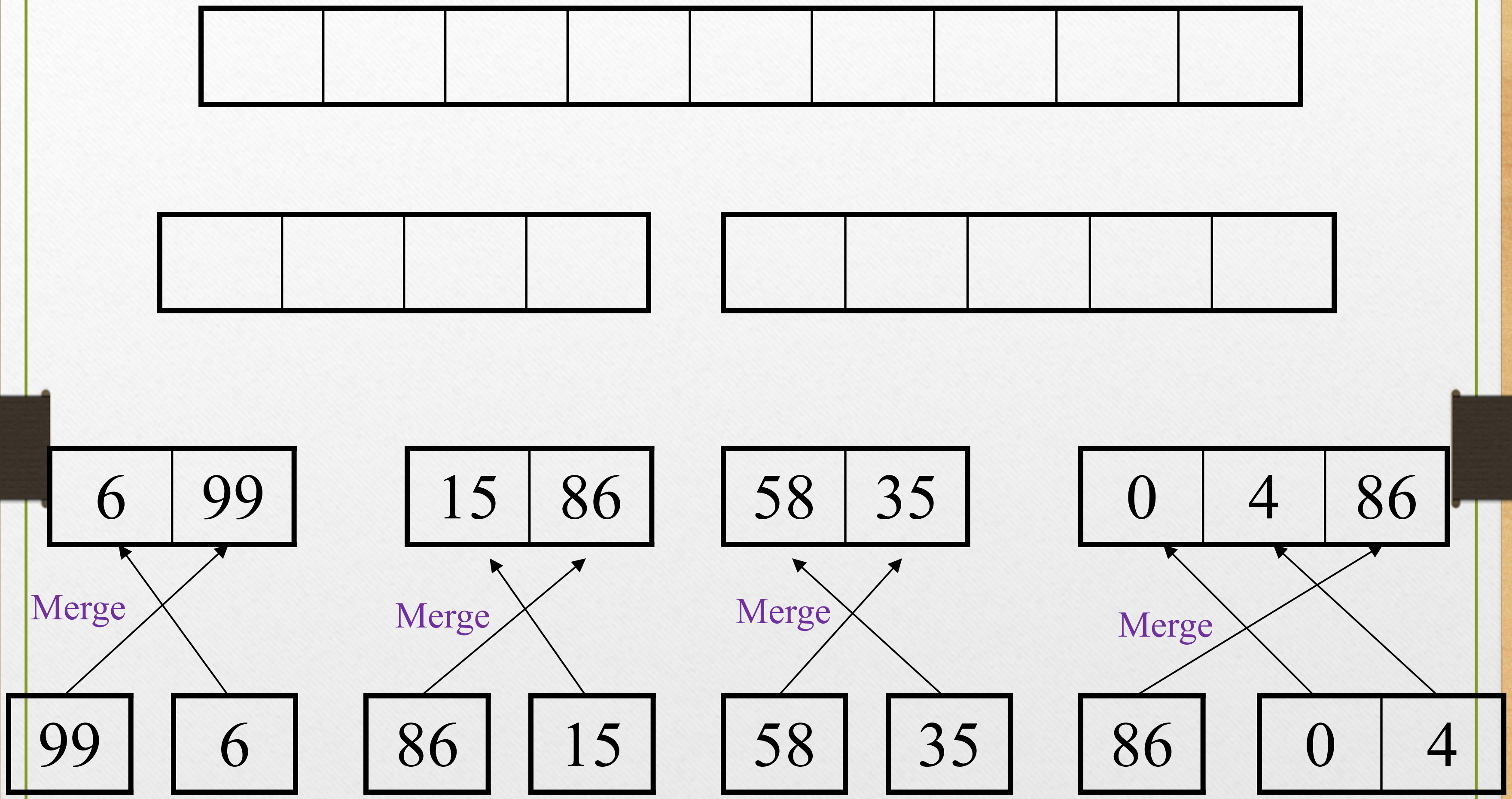
4

0

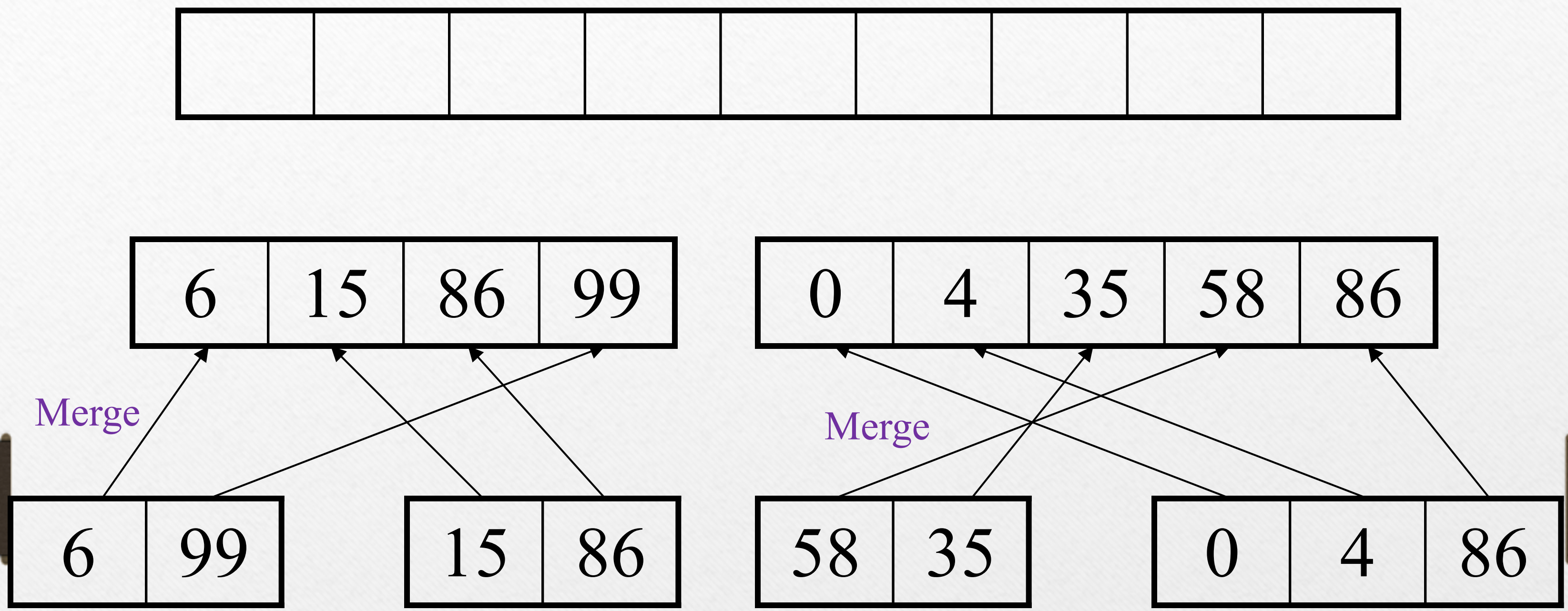
Merge Sort Example



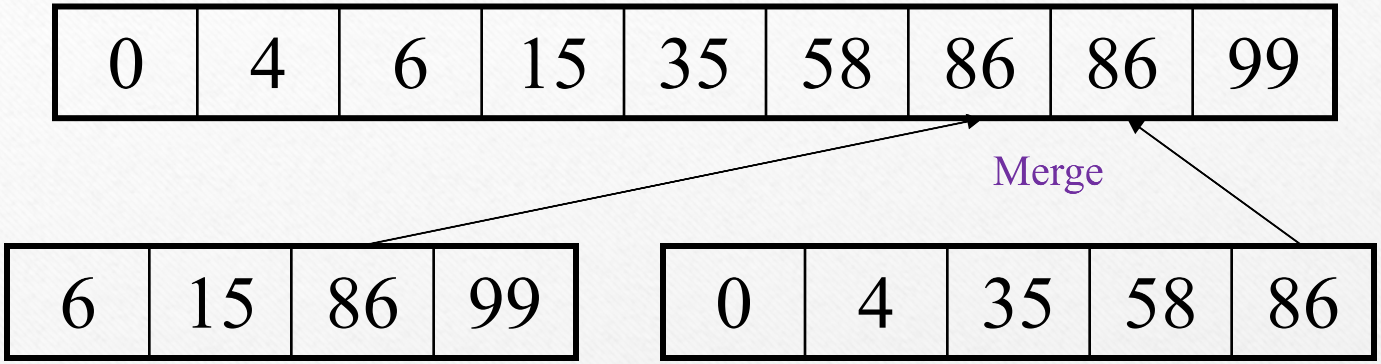
Merge Sort Example



Merge Sort Example



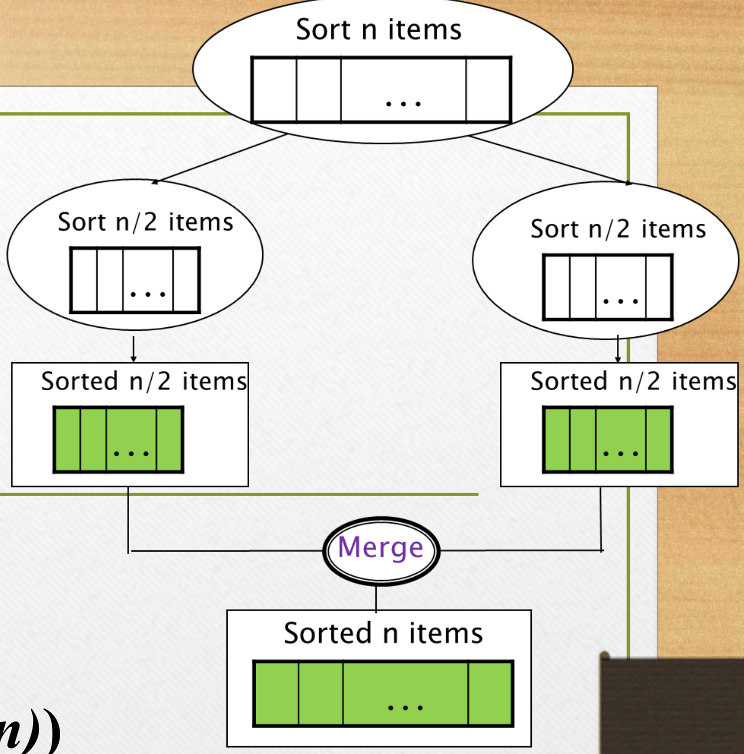
Merge Sort Example



Merge Sort Example

0	4	6	15	35	58	86	86	99

Masters Theorem



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

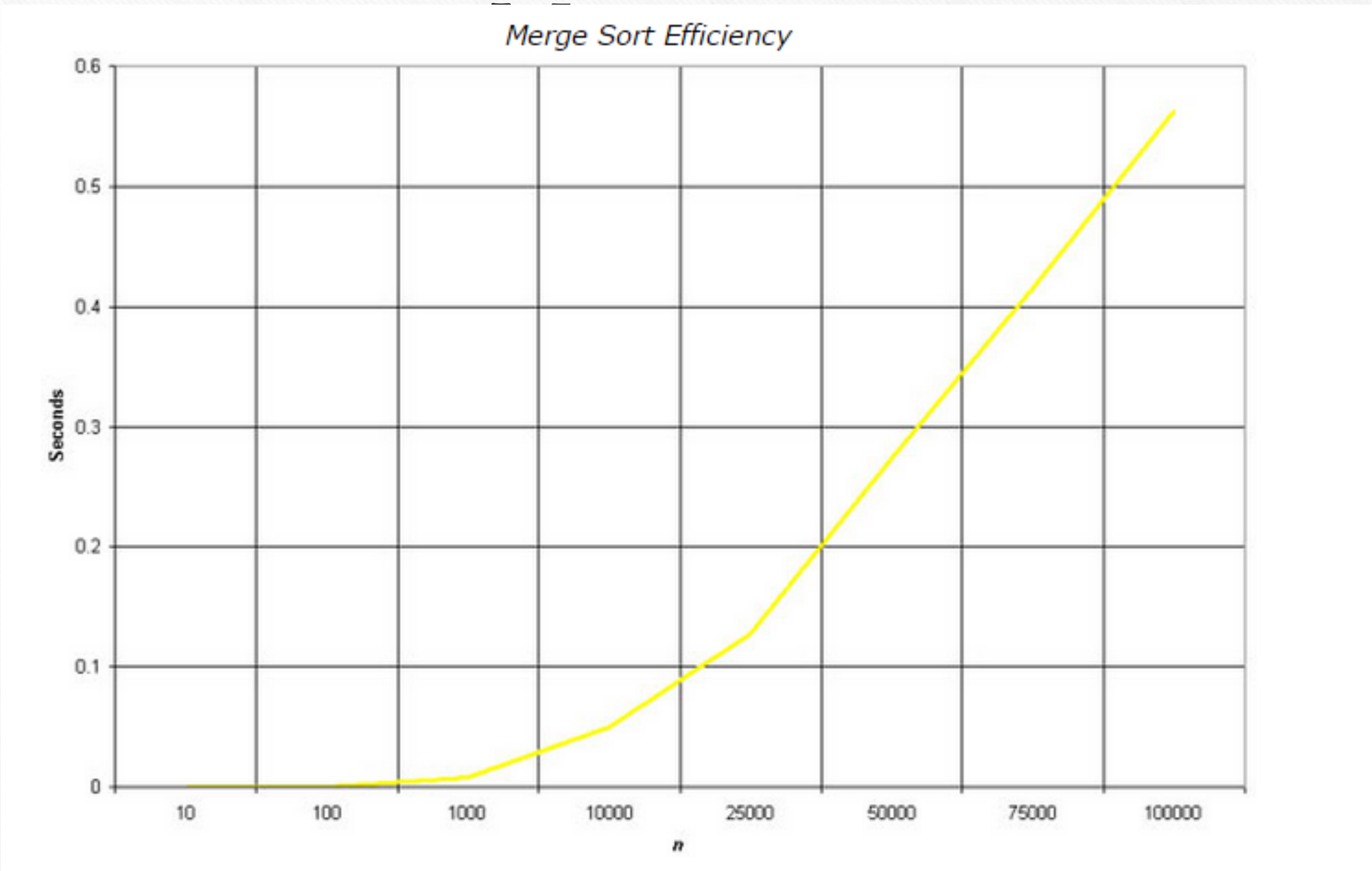
1) If $n^{\log_b a} < F(n)$, $T(n) \in O(F(n))$
2) If $n^{\log_b a} > F(n)$, $T(n) \in O(n^{\log_b a})$
3) If $n^{\log_b a} = F(n)$, $T(n) \in O(n^{\log_b a} \log n)$

Merge Sort: $T(n) = 2T(n/2) + n \Rightarrow T(n) \in ?$

$a = 2$
$b = 2$ $\Rightarrow n^{\log_b a} \Rightarrow n^{\log_2 2} \Rightarrow n$

$F(n) = n$

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



Merge Sort Efficiency
0.6
0.5
0.4
0.3
0.2
0.1
0
Seconds
10
100
1000
10000
25000
50000
75000
100000
n

This week:

- Divide and Conquer technique
- Count a specific key in an array
- Master theorem
- Merge sort

Try it/ homework

1. Chapter 5.1, page 174, questions 1, 6
2. Chapter 5.3, page 185, question 2