

# **Divide-and-Conquer Algorithms**

Antonio Carzaniga

Faculty of Informatics  
Università della Svizzera italiana

March 9, 2023

- Merging (or set union)
- Searching
- Sorting
- Multiplying
- Computing the *median*

# Merging (Set Union)

■ **Input:** sequences  $A = \langle a_1, a_2, \dots, a_n \rangle$  and  $B = \langle b_1, b_2, \dots, b_m \rangle$

**Output:** a sequence  $X = \langle x_1, x_2, \dots, x_\ell \rangle$  such that

■ **Input:** sequences  $A = \langle a_1, a_2, \dots, a_n \rangle$  and  $B = \langle b_1, b_2, \dots, b_m \rangle$

**Output:** a sequence  $X = \langle x_1, x_2, \dots, x_\ell \rangle$  such that

- ▶ every element of  $A$  appears once in  $X$
- ▶ every element of  $B$  appears once in  $X$
- ▶ every element of  $X$  appears in  $A$  or in  $B$  or in both

■ **Input:** sequences  $A = \langle a_1, a_2, \dots, a_n \rangle$  and  $B = \langle b_1, b_2, \dots, b_m \rangle$

**Output:** a sequence  $X = \langle x_1, x_2, \dots, x_\ell \rangle$  such that

- ▶ every element of  $A$  appears once in  $X$
- ▶ every element of  $B$  appears once in  $X$
- ▶ every element of  $X$  appears in  $A$  or in  $B$  or in both

■ **Example:**

$A = \langle 34, 7, 11, 31, 14, 51, 8, 21, 10 \rangle$

$B = \langle 51, 21, 14, 15, 27, 31, 2 \rangle$

$X =$

■ **Input:** sequences  $A = \langle a_1, a_2, \dots, a_n \rangle$  and  $B = \langle b_1, b_2, \dots, b_m \rangle$

**Output:** a sequence  $X = \langle x_1, x_2, \dots, x_\ell \rangle$  such that

- ▶ every element of  $A$  appears once in  $X$
- ▶ every element of  $B$  appears once in  $X$
- ▶ every element of  $X$  appears in  $A$  or in  $B$  or in both

■ **Example:**

$A = \langle 34, 7, 11, 31, 14, 51, 8, 21, 10 \rangle$

$B = \langle 51, 21, 14, 15, 27, 31, 2 \rangle$

$X = \langle 34, 7, 11, 31, 14, 51, 8, 21, 10, 15, 27, 2 \rangle$

# A Simple Merge Algorithm

- Algorithm strategy



## ■ Algorithm strategy

- ▶ iterate through every position  $i$ , first through  $A$ , and then  $B$
- ▶ output  $a_i$  if  $a_i$  is not in  $\langle a_1, a_2, \dots, a_{i-1} \rangle$
- ▶ output  $b_i$  if  $b_i$  is not in  $\langle a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_{i-1} \rangle$

## ■ Algorithm strategy

- ▶ iterate through every position  $i$ , first through  $A$ , and then  $B$
- ▶ output  $a_i$  if  $a_i$  is not in  $\langle a_1, a_2, \dots, a_{i-1} \rangle$
- ▶ output  $b_i$  if  $b_i$  is not in  $\langle a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_{i-1} \rangle$

### **MERGESIMPLE**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not FIND( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not FIND( $A, B[i]$ ) and not FIND( $B[1..i-1], B[i]$ )
6          output  $B[i]$ 
```

**MERGESIMPLE**( $A, B$ )

1 **for**  $i = 1$  **to**  $\text{length}(A)$

2     **if not** **FIND**( $A[1..i-1], A[i]$ )

3         output  $A[i]$

4 **for**  $i = 1$  **to**  $\text{length}(B)$

5     **if not** **FIND**( $A, B[i]$ ) **and not** **FIND**( $B[1..i-1], B[i]$ )

6         output  $B[i]$

**MERGESIMPLE**( $A, B$ )

```

1  for  $i = 1$  to  $length(A)$ 
2      if not FIND( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $length(B)$ 
5      if not FIND( $A, B[i]$ ) and not FIND( $B[1..i-1], B[i]$ )
6          output  $B[i]$ 

```

let  $n = length(A) + length(B)$

$$T(n) = \sum_{i=1}^{length(A)} T_{\mathbf{FIND}}(i) + \sum_{i=1}^{length(B)} (T_{\mathbf{FIND}}(i) + T_{\mathbf{FIND}}(length(A)))$$

**MERGESIMPLE**( $A, B$ )

```

1  for  $i = 1$  to  $length(A)$ 
2      if not FIND( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $length(B)$ 
5      if not FIND( $A, B[i]$ ) and not FIND( $B[1..i-1], B[i]$ )
6          output  $B[i]$ 

```

let  $n = length(A) + length(B)$

$$T(n) = \sum_{i=1}^{length(A)} T_{\mathbf{FIND}}(i) + \sum_{i=1}^{length(B)} (T_{\mathbf{FIND}}(i) + T_{\mathbf{FIND}}(length(A)))$$

$$T(n) = \sum_{i=1}^n T_{\mathbf{FIND}}(i)$$

■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

**FIND**( $A, key$ )

```
1  for  $i = 1$  to  $length(A)$ 
2      if  $A[i] == key$ 
3          return TRUE
4  return FALSE
```

**FIND**( $A, begin, end, key$ )

```
1  for  $i = begin$  to  $end$ 
2      if  $A[i] == key$ 
3          return TRUE
4  return FALSE
```

■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

```
FIND( $A, key$ )
```

```
1  for  $i = 1$  to  $length(A)$ 
```

```
2      if  $A[i] == key$ 
```

```
3          return TRUE
```

```
4  return FALSE
```

```
FIND( $A, begin, end, key$ )
```

```
1  for  $i = begin$  to  $end$ 
```

```
2      if  $A[i] == key$ 
```

```
3          return TRUE
```

```
4  return FALSE
```

■ The complexity of **FIND** is



■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

**FIND**( $A, key$ )

```
1  for  $i = 1$  to  $length(A)$ 
2      if  $A[i] == key$ 
3          return TRUE
4  return FALSE
```

**FIND**( $A, begin, end, key$ )

```
1  for  $i = begin$  to  $end$ 
2      if  $A[i] == key$ 
3          return TRUE
4  return FALSE
```

■ The complexity of **FIND** is

$$T(n) = O(n)$$

■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

```
FINDINLIST( $A, key$ )
```

```
1  $item = first(A)$ 
```

```
2 while  $item \neq last(A)$ 
```

```
3     if  $value(item) == key$ 
```

```
4         return TRUE
```

```
5      $item = next(item)$ 
```

```
6 return FALSE
```

■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

```
FINDINLIST( $A, key$ )  
1   $item = first(A)$   
2  while  $item \neq last(A)$   
3      if  $value(item) == key$   
4          return TRUE  
5       $item = next(item)$   
6  return FALSE
```

■ The complexity of **FINDINLIST** is

■ **Input:** a sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

```
FINDINLIST( $A, key$ )  
1   $item = first(A)$   
2  while  $item \neq last(A)$   
3      if  $value(item) == key$   
4          return TRUE  
5       $item = next(item)$   
6  return FALSE
```

■ The complexity of **FINDINLIST** is

$$T(n) = O(n)$$

## Complexity of MERGESIMPLE

**MERGESIMPLE**( $A, B$ )

1 **for**  $i = 1$  **to**  $\text{length}(A)$

2     **if not** **FIND**( $A[1..i-1], A[i]$ )

3         output  $A[i]$

4 **for**  $i = 1$  **to**  $\text{length}(B)$

5     **if not** **FIND**( $A, B[i]$ ) **and not** **FIND**( $B[1..i-1], B[i]$ )

6         output  $B[i]$

# Complexity of MERGESIMPLE

**MERGESIMPLE**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not FIND( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not FIND( $A, B[i]$ ) and not FIND( $B[1..i-1], B[i]$ )
6          output  $B[i]$ 
```

$$T(n) = \sum_{i=1}^n T_{\mathbf{FIND}}(i)$$

# Complexity of MERGESIMPLE

**MERGESIMPLE**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not FIND( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not FIND( $A, B[i]$ ) and not FIND( $B[1..i-1], B[i]$ )
6          output  $B[i]$ 
```

$$T(n) = \sum_{i=1}^n T_{\mathbf{FIND}}(i)$$

$$T(n) = \sum_{i=1}^n O(i) =$$



# Complexity of MERGESIMPLE

**MERGESIMPLE**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not FIND( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not FIND( $A, B[i]$ ) and not FIND( $B[1..i-1], B[i]$ )
6          output  $B[i]$ 
```

$$T(n) = \sum_{i=1}^n T_{\mathbf{FIND}}(i)$$

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

# Complexity of MERGESIMPLE

**MERGESIMPLE**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not FIND( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not FIND( $A, B[i]$ ) and not FIND( $B[1..i-1], B[i]$ )
6          output  $B[i]$ 
```

$$T(n) = \sum_{i=1}^n T_{\mathbf{FIND}}(i)$$

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

- **Input:** a *sorted* sequence  $A$  and a value  $key$   
**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

■ **Input:** a *sorted* sequence  $A$  and a value  $key$

**Output:** TRUE if  $A$  contains  $key$ , or FALSE otherwise

**BINARYSEARCH**( $A, key$ )

```
1  first = 1
2  last = length( $A$ )
3  while first ≤ last
4      middle =  $\lceil (first + last) / 2 \rceil$ 
5      if  $A[middle] == key$ 
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif  $A[middle] > key$ 
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```

**BINARYSEARCH**(*A*, *key*)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle =  $\lceil (\textit{first} + \textit{last}) / 2 \rceil$ 
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```

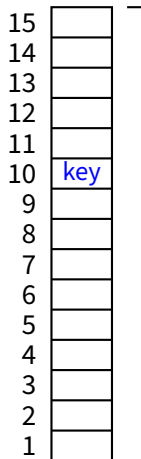
**BINARYSEARCH**(A, key)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```

15	
14	
13	
12	
11	
10	key
9	
8	
7	
6	
5	
4	
3	
2	
1	

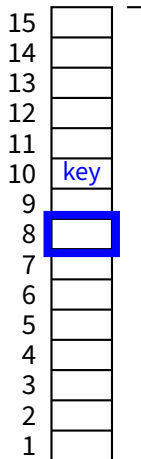
## **BINARYSEARCH**(A, key)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```



**BINARYSEARCH**(A, key)

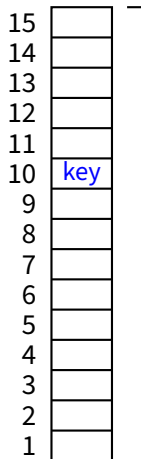
```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```





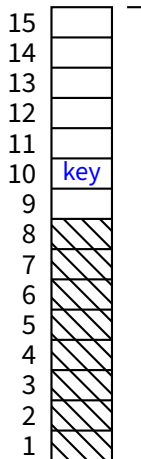
## **BINARYSEARCH**(A, key)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```



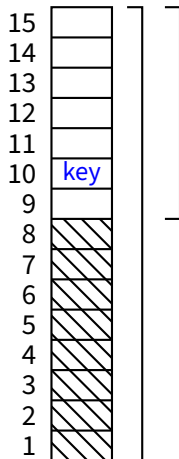
**BINARYSEARCH**(A, key)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```



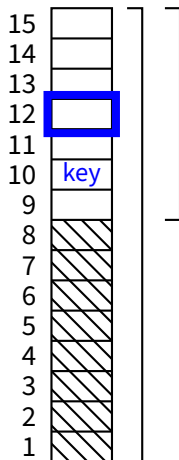
**BINARYSEARCH**(A, key)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```



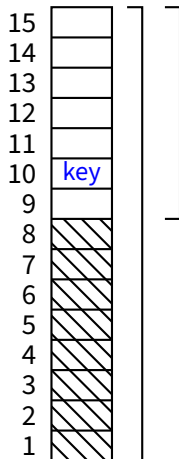
**BINARYSEARCH**(A, key)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```



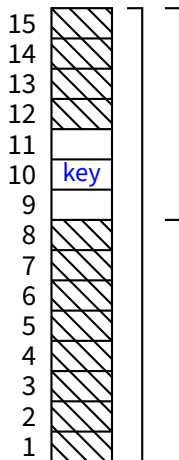
**BINARYSEARCH**(A, key)

```
1 first = 1
2 last = length(A)
3 while first ≤ last
4     middle = ⌈(first + last)/2⌉
5     if A[middle] == key
6         return TRUE
7     elseif first = last
8         return FALSE
9     elseif A[middle] > key
10        last = middle - 1
11    else first = middle + 1
12 return FALSE
```



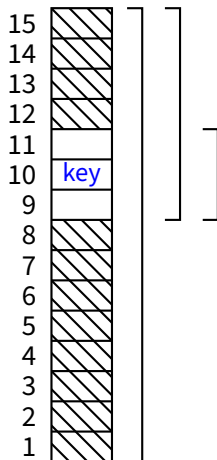
**BINARYSEARCH**(A, key)

```
1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE
```



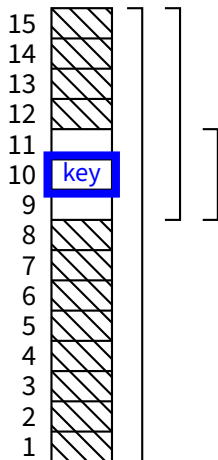
**BINARYSEARCH**(A, key)

```
1 first = 1
2 last = length(A)
3 while first ≤ last
4     middle = ⌈(first + last)/2⌉
5     if A[middle] == key
6         return TRUE
7     elseif first = last
8         return FALSE
9     elseif A[middle] > key
10        last = middle - 1
11    else first = middle + 1
12 return FALSE
```



**BINARYSEARCH**(A, key)

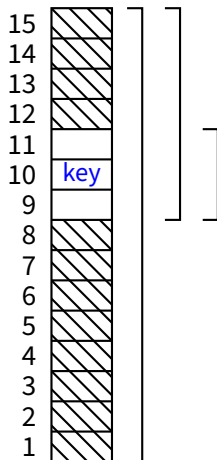
```
1 first = 1
2 last = length(A)
3 while first ≤ last
4     middle = ⌈(first + last)/2⌉
5     if A[middle] == key
6         return TRUE
7     elseif first = last
8         return FALSE
9     elseif A[middle] > key
10        last = middle - 1
11    else first = middle + 1
12 return FALSE
```





**BINARYSEARCH**(A, key)

```
1 first = 1
2 last = length(A)
3 while first ≤ last
4     middle = ⌈(first + last)/2⌉
5     if A[middle] == key
6         return TRUE
7     elseif first = last
8         return FALSE
9     elseif A[middle] > key
10        last = middle - 1
11    else first = middle + 1
12 return FALSE
```

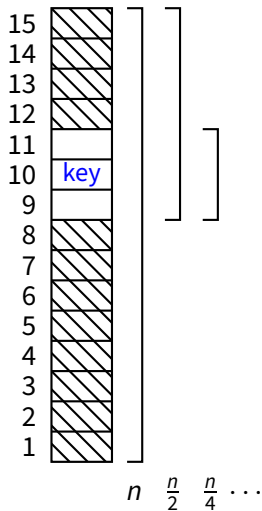


**BINARYSEARCH**(A, key)

```

1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE

```



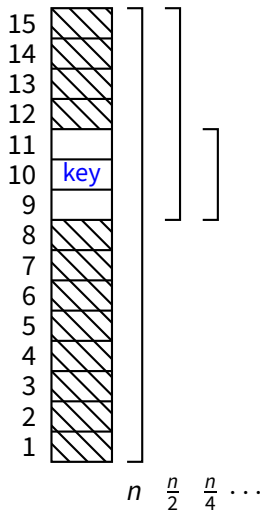
**BINARYSEARCH**(A, key)

```

1  first = 1
2  last = length(A)
3  while first ≤ last
4      middle = ⌈(first + last)/2⌉
5      if A[middle] == key
6          return TRUE
7      elseif first = last
8          return FALSE
9      elseif A[middle] > key
10         last = middle - 1
11     else first = middle + 1
12 return FALSE

```

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



- A slightly different problem:

**Input:** two *sorted* sequences  $A = \langle a_1, a_2, \dots, a_n \rangle$  and  $B = \langle b_1, b_2, \dots, b_m \rangle$ , where  $a_1 \leq a_2 \leq \dots \leq a_n$  and  $b_1 \leq b_2 \leq \dots \leq b_m$

**Output:** a sequence  $X = \langle x_1, x_2, \dots, x_\ell \rangle$  such that

- ▶ every element of  $A$  appears once in  $X$
- ▶ every element of  $B$  appears once in  $X$
- ▶ every element of  $X$  appears in  $A$  or in  $B$  or in both

**MERGESIMPLE2**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not BINARYSEARCH( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not BINARYSEARCH( $A, B[i]$ )
6          and not BINARYSEARCH( $B[1..i-1], B[i]$ )
7          output  $B[i]$ 
```

**MERGESIMPLE2**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not BINARYSEARCH( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not BINARYSEARCH( $A, B[i]$ )
6          and not BINARYSEARCH( $B[1..i-1], B[i]$ )
7          output  $B[i]$ 
```

$$T(n) = \sum_{i=1}^n O(\log i) =$$

**MERGESIMPLE2**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not BINARYSEARCH( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not BINARYSEARCH( $A, B[i]$ )
6          and not BINARYSEARCH( $B[1..i-1], B[i]$ )
7          output  $B[i]$ 
```

$$T(n) = \sum_{i=1}^n O(\log i) = O(n \log n)$$

# A Better Merge Algorithm

**MERGESIMPLE2**( $A, B$ )

```
1  for  $i = 1$  to  $\text{length}(A)$ 
2      if not BINARYSEARCH( $A[1..i-1], A[i]$ )
3          output  $A[i]$ 
4  for  $i = 1$  to  $\text{length}(B)$ 
5      if not BINARYSEARCH( $A, B[i]$ )
6      and not BINARYSEARCH( $B[1..i-1], B[i]$ )
7          output  $B[i]$ 
```

$$T(n) = \sum_{i=1}^n O(\log i) = O(n \log n)$$

Better than  $O(n^2)$ , but can we do even better than  $O(n \log n)$ ?



# An Even Better Merge Algorithm

- *Intuition: A and B are sorted*

e.g.

$A = \langle 3, 7, 12, 13, 34, 37, 70, 75, 80 \rangle$

$B = \langle 1, 5, 6, 7, 34, 35, 40, 41, 43 \rangle$

# An Even Better Merge Algorithm

- *Intuition: A and B are sorted*

e.g.

$A = \langle 3, 7, 12, 13, 34, 37, 70, 75, 80 \rangle$

$B = \langle 1, 5, 6, 7, 34, 35, 40, 41, 43 \rangle$

so just like in **BINARYSEARCH** I can avoid looking for an element  $x$  if the *first* element I see is  $y > x$

# An Even Better Merge Algorithm

- *Intuition: A and B are sorted*

e.g.

$A = \langle 3, 7, 12, 13, 34, 37, 70, 75, 80 \rangle$

$B = \langle 1, 5, 6, 7, 34, 35, 40, 41, 43 \rangle$

so just like in **BINARYSEARCH** I can avoid looking for an element  $x$  if the *first* element I see is  $y > x$

- High-level algorithm strategy

- ▶ step through every position  $i$  of  $A$  and every position  $j$  of  $B$
- ▶ output  $a_i$  and advance  $i$  if  $a_i \leq b_j$  or if  $j$  is beyond the end of  $B$
- ▶ output  $b_j$  and advance  $j$  if  $a_i \geq b_j$  or if  $i$  is beyond the end of  $A$

# MERGE Algorithm

A

3	7	12	13	34	37	70	75	80
---	---	----	----	----	----	----	----	----

B

1	5	6	7	34	35	40	41	43
---	---	---	---	----	----	----	----	----

# MERGE Algorithm

$i = 1$



A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----

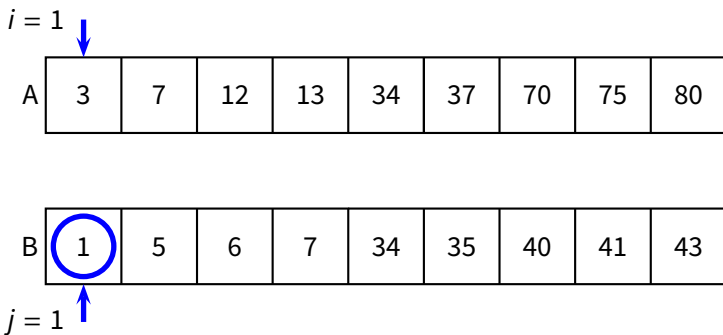
B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 1$



Output:

# MERGE Algorithm



Output:

# MERGE Algorithm

$i = 1$



A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----

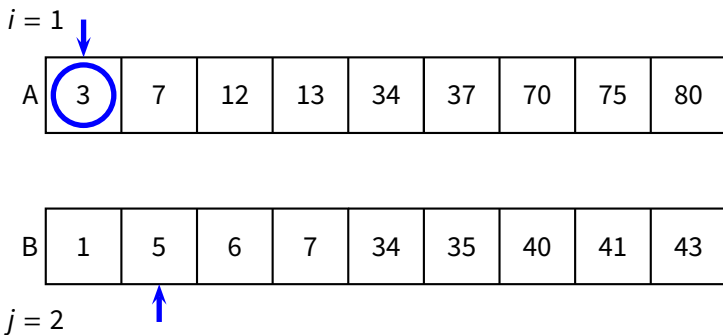
B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 2$



Output: 1

# MERGE Algorithm




Output: 1



# MERGE Algorithm

$i = 2$



A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----

B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

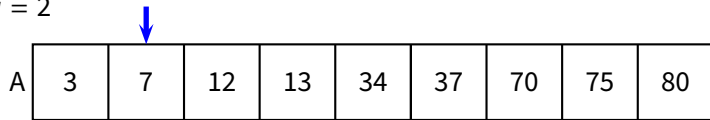
$j = 2$



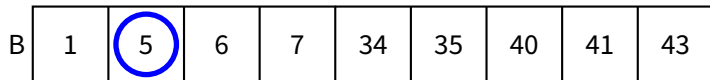
Output: 13

# MERGE Algorithm

$i = 2$



A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----




B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 2$

Output: 13

# MERGE Algorithm

$i = 2$



A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----

B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

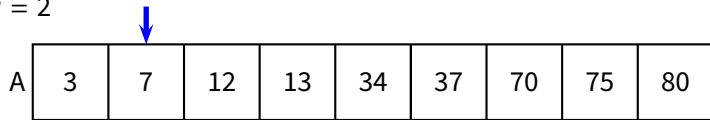
$j = 3$



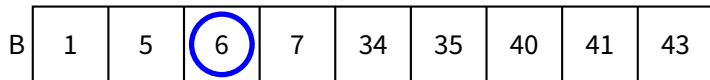
Output: 1 3 5

# MERGE Algorithm

$i = 2$



A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----




B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 3$

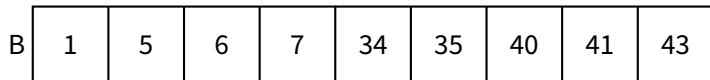
Output: 1 3 5

# MERGE Algorithm

$i = 2$



A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----



B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 4$

Output: 1 3 5 6

# MERGE Algorithm

$i = 2$

A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----


B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 4$

Output: 1 3 5 6

# MERGE Algorithm

$i = 3$

									
A	3	7	12	13	34	37	70	75	80

B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 5$

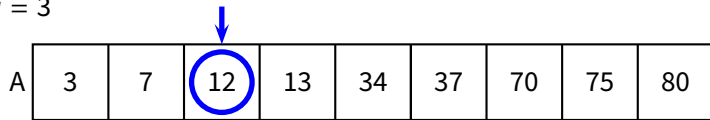


Output: 1 3 5 6 7

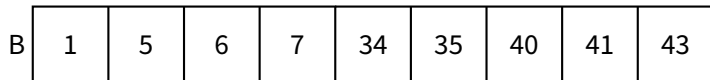
# MERGE Algorithm

$i = 3$

A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----



B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----




$j = 5$

Output: 1 3 5 6 7



# MERGE Algorithm

$i = 4$

									
A	3	7	12	13	34	37	70	75	80

B	1	5	6	7	34	35	40	41	43

$j = 5$

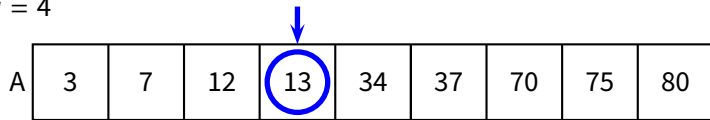


Output: 1 3 5 6 7 12

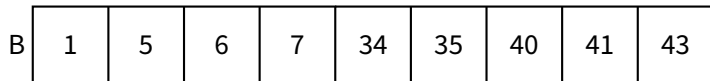
# MERGE Algorithm

$i = 4$

A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----



B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----



$j = 5$

Output: 1 3 5 6 7 12

# MERGE Algorithm

$i = 5$

A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----



B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 5$



Output: 1 3 5 6 7 12 13

# MERGE Algorithm

$i = 5$

A	3	7	12	13	34	37	70	75	80
---	---	---	----	----	----	----	----	----	----



B	1	5	6	7	34	35	40	41	43
---	---	---	---	---	----	----	----	----	----

$j = 5$



Output: 1 3 5 6 7 12 13...

## MERGE Algorithm (2)

**MERGE**( $A, B$ )

```
1   $i, j = 1$ 
2   $X = \emptyset$ 
3  while  $i \leq \text{length}(A)$  or  $j \leq \text{length}(B)$ 
4      if  $i > \text{length}(A)$ 
5           $X = X \circ B[j]$            // appends  $B[j]$  to  $X$ 
6           $j = j + 1$ 
7      elseif  $j > \text{length}(B)$ 
8           $X = X \circ A[i]$ 
9           $i = i + 1$ 
10     elseif  $A[i] < B[j]$ 
11          $X = X \circ A[i]$ 
12          $i = i + 1$ 
13     else  $X = X \circ B[j]$ 
14          $j = j + 1$ 
15 return  $X$ 
```

```
MERGE( $A, B$ )
1   $i, j = 1$ 
2   $X = \emptyset$ 
3  while  $i \leq \text{length}(A)$  or  $j \leq \text{length}(B)$ 
4      if  $i > \text{length}(A)$ 
5           $X = X \circ B[j]$            // appends  $B[j]$  to  $X$ 
6           $j = j + 1$ 
7      elseif  $j > \text{length}(B)$ 
8           $X = X \circ A[i]$ 
9           $i = i + 1$ 
10     elseif  $A[i] < B[j]$ 
11          $X = X \circ A[i]$ 
12          $i = i + 1$ 
13     else  $X = X \circ B[j]$ 
14          $j = j + 1$ 
15     return  $X$ 
```

- This algorithm is incorrect! (Exercise: fix it)

**MERGE**( $A, B$ )

1  $i, j = 1$

2  $X = \emptyset$

3 **while**  $i \leq \text{length}(A)$  **or**  $j \leq \text{length}(B)$

4     **if**  $i \leq \text{length}(A)$  **and**  $(j > \text{length}(B)$  **or**  $A[i] < B[j])$

5          $X = X \circ A[i]$

6          $i = i + 1$

7     **else**  $X = X \circ B[j]$

8          $j = j + 1$

9 **return**  $X$

**MERGE**( $A, B$ )

1  $i, j = 1$

2  $X = \emptyset$

3 **while**  $i \leq \text{length}(A)$  **or**  $j \leq \text{length}(B)$

4     **if**  $i \leq \text{length}(A)$  **and**  $(j > \text{length}(B)$  **or**  $A[i] < B[j])$

5          $X = X \circ A[i]$

6          $i = i + 1$

7     **else**  $X = X \circ B[j]$

8          $j = j + 1$

9 **return**  $X$

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



**MERGE**( $A, B$ )

1  $i, j = 1$

2  $X = \emptyset$

3 **while**  $i \leq \text{length}(A)$  **or**  $j \leq \text{length}(B)$

4     **if**  $i \leq \text{length}(A)$  **and**  $(j > \text{length}(B)$  **or**  $A[i] < B[j])$

5          $X = X \circ A[i]$

6          $i = i + 1$

7     **else**  $X = X \circ B[j]$

8          $j = j + 1$

9 **return**  $X$

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

- Can we do better?

**MERGE**( $A, B$ )

1  $i, j = 1$

2  $X = \emptyset$

3 **while**  $i \leq \text{length}(A)$  **or**  $j \leq \text{length}(B)$

4     **if**  $i \leq \text{length}(A)$  **and**  $(j > \text{length}(B)$  **or**  $A[i] < B[j])$

5          $X = X \circ A[i]$

6          $i = i + 1$

7     **else**  $X = X \circ B[j]$

8          $j = j + 1$

9 **return**  $X$

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

- Can we do better? No!

**MERGE**( $A, B$ )

1  $i, j = 1$

2  $X = \emptyset$

3 **while**  $i \leq \text{length}(A)$  **or**  $j \leq \text{length}(B)$

4     **if**  $i \leq \text{length}(A)$  **and**  $(j > \text{length}(B)$  **or**  $A[i] < B[j])$

5          $X = X \circ A[i]$

6          $i = i + 1$

7     **else**  $X = X \circ B[j]$

8          $j = j + 1$

9 **return**  $X$

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

■ Can we do better? No!

- ▶ we have to output  $n = \text{length}(A) + \text{length}(B)$  elements

- So now we have a *linear-complexity* merge procedure
  - ▶ merges two *sorted* sequences
  - ▶ *produces a sorted sequence*

- So now we have a *linear-complexity* merge procedure
  - ▶ merges two *sorted* sequences
  - ▶ *produces a sorted sequence*
  
- Perhaps we could use it to implement a sort algorithm

- So now we have a *linear-complexity* merge procedure
  - ▶ merges two *sorted* sequences
  - ▶ *produces a sorted sequence*
- Perhaps we could use it to implement a sort algorithm
- Idea
  - ▶ use a variant of **MERGE** that outputs *all* elements of its input sequences
    - ▶ i.e., without removing duplicates
  - ▶ assume that two parts,  $A_L \circ A_R = A$ , and that  $A_L$  and  $A_R$  are sorted

- So now we have a *linear-complexity* merge procedure
  - ▶ merges two *sorted* sequences
  - ▶ *produces a sorted sequence*
- Perhaps we could use it to implement a sort algorithm
- Idea
  - ▶ use a variant of **MERGE** that outputs *all* elements of its input sequences
    - ▶ i.e., without removing duplicates
  - ▶ assume that two parts,  $A_L \circ A_R = A$ , and that  $A_L$  and  $A_R$  are sorted
  - ▶ use **MERGE** to combine  $A_L$  and  $A_R$  into a sorted sequence

- So now we have a *linear-complexity* merge procedure
  - ▶ merges two *sorted* sequences
  - ▶ *produces a sorted sequence*
- Perhaps we could use it to implement a sort algorithm
- Idea
  - ▶ use a variant of **MERGE** that outputs *all* elements of its input sequences
    - ▶ i.e., without removing duplicates
  - ▶ assume that two parts,  $A_L \circ A_R = A$ , and that  $A_L$  and  $A_R$  are sorted
  - ▶ use **MERGE** to combine  $A_L$  and  $A_R$  into a sorted sequence
  - ▶ this suggests a recursive algorithm





## **MERGESORT**( $A$ )

```
1  if  $length(A) == 1$ 
2      return  $A$ 
3   $m = \lfloor length(A)/2 \rfloor$ 
4   $A_L = \mathbf{MERGESORT}(A[1..m])$ 
5   $A_R = \mathbf{MERGESORT}(A[m + 1..length(A)])$ 
6  return MERGE( $A_L, A_R$ )
```

## **MERGESORT**( $A$ )

```
1  if  $length(A) == 1$ 
2      return  $A$ 
3   $m = \lfloor length(A)/2 \rfloor$ 
4   $A_L = \mathbf{MERGESORT}(A[1..m])$ 
5   $A_R = \mathbf{MERGESORT}(A[m+1..length(A)])$ 
6  return MERGE( $A_L, A_R$ )
```

- The complexity of **MERGESORT** is

## **MERGESORT**(*A*)

```
1  if length(A) == 1
2      return A
3  m =  $\lfloor \text{length}(A)/2 \rfloor$ 
4  AL = MERGESORT(A[1..m])
5  AR = MERGESORT(A[m + 1..length(A)])
6  return MERGE(AL, AR)
```

- The complexity of **MERGESORT** is

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

## **MERGESORT**(A)

```
1  if length(A) == 1
2      return A
3   $m = \lfloor \text{length}(A)/2 \rfloor$ 
4   $A_L = \mathbf{MERGESORT}(A[1..m])$ 
5   $A_R = \mathbf{MERGESORT}(A[m+1..length(A)])$ 
6  return MERGE( $A_L, A_R$ )
```

- The complexity of **MERGESORT** is

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

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

- **MERGESORT** exemplifies the *divide and conquer* strategy

- **MERGESORT** exemplifies the *divide and conquer* strategy
- *General strategy*: given a problem  $P$  on input data  $A$ 
  - ▶ *divide* the input  $A$  into parts  $A_1, A_2, \dots, A_k$ , usually *disjoint*, surely with  $|A_i| < |A| = n$
  - ▶ *solve* problem  $P$  for the individual  $k$  parts
  - ▶ *combine* the partial solutions to obtain the solution for  $A$

- **MERGESORT** exemplifies the *divide and conquer* strategy
- *General strategy*: given a problem  $P$  on input data  $A$ 
  - ▶ **divide** the input  $A$  into parts  $A_1, A_2, \dots, A_k$ , usually *disjoint*, surely with  $|A_i| < |A| = n$
  - ▶ **solve** problem  $P$  for the individual  $k$  parts
  - ▶ **combine** the partial solutions to obtain the solution for  $A$
- Complexity analysis

$$T(n) = T_{\text{divide}} + \sum_{i=1}^k T(|A_i|) + T_{\text{combine}}$$

we might analyze this formula another time...



## A Divide-and-Conquer Merge

**MERGER**( $A, B$ )

1 **if**  $length(A) == 0$

2     **return**  $B$

3 **if**  $length(B) == 0$

4     **return**  $A$

5 **if**  $A[1] < B[1]$

6     **return**  $A[1] \circ \mathbf{MERGER}(A[2..length(A)], B)$

7 **else return**  $B[1] \circ \mathbf{MERGER}(A, B[2..length(B)])$

## A Divide-and-Conquer Merge

**MERGER**( $A, B$ )

1 **if**  $length(A) == 0$

2     **return**  $B$

3 **if**  $length(B) == 0$

4     **return**  $A$

5 **if**  $A[1] < B[1]$

6     **return**  $A[1] \circ \mathbf{MERGER}(A[2..length(A)], B)$

7 **else return**  $B[1] \circ \mathbf{MERGER}(A, B[2..length(B)])$

- Again, this algorithm is a bit incorrect (Exercise: Fix it.)

## A Divide-and-Conquer Merge

**MERGER**( $A, B$ )

1 **if**  $length(A) == 0$

2     **return**  $B$

3 **if**  $length(B) == 0$

4     **return**  $A$

5 **if**  $A[1] < B[1]$

6     **return**  $A[1] \circ \mathbf{MERGER}(A[2..length(A)], B)$

7 **else return**  $B[1] \circ \mathbf{MERGER}(A, B[2..length(B)])$

- Again, this algorithm is a bit incorrect (Exercise: Fix it.)
- The complexity of **MERGER** is

## A Divide-and-Conquer Merge

```
MERGER(A, B)
1  if length(A) == 0
2      return B
3  if length(B) == 0
4      return A
5  if A[1] < B[1]
6      return A[1] ◦ MERGER(A[2..length(A)], B)
7  else return B[1] ◦ MERGER(A, B[2..length(B)])
```

- Again, this algorithm is a bit incorrect (Exercise: Fix it.)
- The complexity of **MERGER** is

$$T(n) = C_1 + T(n - 1)$$

## A Divide-and-Conquer Merge

```
MERGER(A, B)
1  if length(A) == 0
2      return B
3  if length(B) == 0
4      return A
5  if A[1] < B[1]
6      return A[1] ◦ MERGER(A[2..length(A)], B)
7  else return B[1] ◦ MERGER(A, B[2..length(B)])
```

- Again, this algorithm is a bit incorrect (Exercise: Fix it.)
- The complexity of **MERGER** is

$$T(n) = C_1 + T(n - 1) = C_1 n$$

## A Divide-and-Conquer Merge

```
MERGER(A, B)
1  if length(A) == 0
2      return B
3  if length(B) == 0
4      return A
5  if A[1] < B[1]
6      return A[1] ◦ MERGER(A[2..length(A)], B)
7  else return B[1] ◦ MERGER(A, B[2..length(B)])
```

- Again, this algorithm is a bit incorrect (Exercise: Fix it.)
- The complexity of **MERGER** is

$$T(n) = C_1 + T(n - 1) = C_1 n = O(n)$$

- Can we do better?

```
MERGER(A, B)
1  if length(A) == 0
2      return B
3  if length(B) == 0
4      return A
5  if A[1] < B[1]
6      return A[1] ◦ MERGER(A[2..length(A)], B)
7  else return B[1] ◦ MERGER(A, B[2..length(B)])
```

- Again, this algorithm is a bit incorrect (Exercise: Fix it.)
- The complexity of **MERGER** is

$$T(n) = C_1 + T(n - 1) = C_1 n = O(n)$$

- Can we do better? No! (We knew that already)

# Divide-and-Conquer Multiplication



# Divide-and-Conquer Multiplication

- Going back to multiplication...

# Divide-and-Conquer Multiplication

- Going back to multiplication...

$$x = \boxed{X_L} \boxed{X_R} \quad \text{and} \quad y = \boxed{Y_L} \boxed{Y_R}$$

# Divide-and-Conquer Multiplication

- Going back to multiplication...

$$x = \boxed{X_L} \boxed{X_R} \quad \text{and} \quad y = \boxed{Y_L} \boxed{Y_R}$$

which means  $x = 2^{\ell/2}x_L + x_R$  and  $y = 2^{\ell/2}y_L + y_R$ , so...

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

we reduced the problem of multiplying two numbers of  $\ell$  bits into the problem of multiplying *four* numbers of  $\ell/2$  bits...

# Divide-and-Conquer Multiplication

- Going back to multiplication...

$$x = \boxed{X_L} \boxed{X_R} \quad \text{and} \quad y = \boxed{Y_L} \boxed{Y_R}$$

which means  $x = 2^{\ell/2}x_L + x_R$  and  $y = 2^{\ell/2}y_L + y_R$ , so...

$$\begin{aligned}xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R\end{aligned}$$

we reduced the problem of multiplying two numbers of  $\ell$  bits into the problem of multiplying *four* numbers of  $\ell/2$  bits...

$$T(\ell) = 4T(\ell/2) + O(\ell)$$

# Divide-and-Conquer Multiplication

- Going back to multiplication...

$$x = \boxed{X_L} \boxed{X_R} \quad \text{and} \quad y = \boxed{Y_L} \boxed{Y_R}$$

which means  $x = 2^{\ell/2}x_L + x_R$  and  $y = 2^{\ell/2}y_L + y_R$ , so...

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

we reduced the problem of multiplying two numbers of  $\ell$  bits into the problem of multiplying *four* numbers of  $\ell/2$  bits...

$$T(\ell) = 4T(\ell/2) + O(\ell)$$

$$T(\ell) = \Theta(\ell^2)$$

## Divide-and-Conquer Multiplication (2)

## Divide-and-Conquer Multiplication (2)

- Again, we have

$$\begin{aligned}xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^\ell x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

## Divide-and-Conquer Multiplication (2)

- Again, we have

$$\begin{aligned}xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^\ell x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

but notice that  $x_L y_R + x_R y_L = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$ , so



## Divide-and-Conquer Multiplication (2)

- Again, we have

$$\begin{aligned}xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^\ell x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

but notice that  $x_L y_R + x_R y_L = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$ , so

$$xy = 2^\ell x_L y_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R) + x_R y_R$$

## Divide-and-Conquer Multiplication (2)

- Again, we have

$$\begin{aligned}xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^\ell x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

but notice that  $x_L y_R + x_R y_L = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$ , so

$$xy = 2^\ell x_L y_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R) + x_R y_R$$

Only 3 multiplications:  $x_L y_L$ ,  $(x_L + x_R)(y_R + y_L)$ , and  $x_R y_R$

## Divide-and-Conquer Multiplication (2)

- Again, we have

$$\begin{aligned}xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^\ell x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

but notice that  $x_L y_R + x_R y_L = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$ , so

$$xy = 2^\ell x_L y_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R) + x_R y_R$$

Only 3 multiplications:  $x_L y_L$ ,  $(x_L + x_R)(y_R + y_L)$ , and  $x_R y_R$

$$T(\ell) = 3T(\ell/2) + O(\ell)$$

## Divide-and-Conquer Multiplication (2)

- Again, we have

$$\begin{aligned}xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^\ell x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

but notice that  $x_L y_R + x_R y_L = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$ , so

$$xy = 2^\ell x_L y_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R) + x_R y_R$$

Only 3 multiplications:  $x_L y_L$ ,  $(x_L + x_R)(y_R + y_L)$ , and  $x_R y_R$

$$T(\ell) = 3T(\ell/2) + O(\ell)$$

which, as we will see, leads to a much better complexity

$$T(\ell) = O(\ell^{\log_2 3}) = O(\ell^{1.59})$$

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?
- Idea: first sort, then pick the element in the middle

```
SIMPLEMEDIAN( $A$ )
```

```
1  $X = \mathbf{MERGESORT}(A)$ 
```

```
2 return  $X[\lfloor \text{length}(A)/2 \rfloor]$ 
```

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?
- Idea: first sort, then pick the element in the middle

```
SIMPLEMEDIAN( $A$ )
```

```
1  $X = \mathbf{MERGESORT}(A)$ 
```

```
2 return  $X[\lfloor \text{length}(A)/2 \rfloor]$ 
```

- Is it correct?



- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?
- Idea: first sort, then pick the element in the middle

```
SIMPLEMEDIAN( $A$ )
```

```
1  $X = \mathbf{MERGESORT}(A)$ 
```

```
2 return  $X[\lfloor \text{length}(A)/2 \rfloor]$ 
```

- Is it correct? Yes

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?
- Idea: first sort, then pick the element in the middle

## **SIMPLEMEDIAN**( $A$ )

```
1  $X = \text{MERGESORT}(A)$   
2 return  $X[\lfloor \text{length}(A)/2 \rfloor]$ 
```

- Is it correct? Yes
- How long does it take?

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?
- Idea: first sort, then pick the element in the middle

## **SIMPLEMEDIAN**( $A$ )

```
1  $X = \text{MERGESORT}(A)$   
2 return  $X[\lfloor \text{length}(A)/2 \rfloor]$ 
```

- Is it correct? Yes
- How long does it take?  $T(n) = T_{\text{MERGESORT}}(n) = O(n \log n)$

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?
- Idea: first sort, then pick the element in the middle

## **SIMPLEMEDIAN**( $A$ )

```
1  $X = \text{MERGESORT}(A)$   
2 return  $X[\lfloor \text{length}(A)/2 \rfloor]$ 
```

- Is it correct? Yes
- How long does it take?  $T(n) = T_{\text{MERGESORT}}(n) = O(n \log n)$
- Can we do better?

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are smaller than  $m$  and half are bigger than  $m$ 
  - ▶ e.g., what is the median of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?
- Idea: first sort, then pick the element in the middle

## **SIMPLEMEDIAN**( $A$ )

```
1  $X = \text{MERGESORT}(A)$   
2 return  $X[\lfloor \text{length}(A)/2 \rfloor]$ 
```

- Is it correct? Yes
- How long does it take?  $T(n) = T_{\text{MERGESORT}}(n) = O(n \log n)$
- Can we do better? Intuitively, yes...

## Computing the Median (2)

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are less than or equal to  $m$  and the other half are greater than or equal to  $m$

## Computing the Median (2)

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are less than or equal to  $m$  and the other half are greater than or equal to  $m$
- Generalizing, the ***k-smallest*** element of a sequence  $A$  is a value  $v \in A$  such that  $k$  elements of  $A$  are less than or equal to  $v$  and  $n - k$  are greater or equal to  $v$

## Computing the Median (2)

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are less than or equal to  $m$  and the other half are greater than or equal to  $m$
- Generalizing, the ***k-smallest*** element of a sequence  $A$  is a value  $v \in A$  such that  $k$  elements of  $A$  are less than or equal to  $v$  and  $n - k$  are greater or equal to  $v$

E.g.,

- ▶ for  $k = 1$ , the minimum of  $A$



## Computing the Median (2)

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are less than or equal to  $m$  and the other half are greater than or equal to  $m$
- Generalizing, the ***k-smallest*** element of a sequence  $A$  is a value  $v \in A$  such that  $k$  elements of  $A$  are less than or equal to  $v$  and  $n - k$  are greater or equal to  $v$

E.g.,

- ▶ for  $k = 1$ , the minimum of  $A$
- ▶ for  $k = \lfloor |A|/2 \rfloor$ , the median of  $A$

## Computing the Median (2)

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are less than or equal to  $m$  and the other half are greater than or equal to  $m$
- Generalizing, the ***k-smallest*** element of a sequence  $A$  is a value  $v \in A$  such that  $k$  elements of  $A$  are less than or equal to  $v$  and  $n - k$  are greater or equal to  $v$

E.g.,

- ▶ for  $k = 1$ , the minimum of  $A$
- ▶ for  $k = \lfloor |A|/2 \rfloor$ , the median of  $A$
- ▶ what is the *6th smallest* element of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?

## Computing the Median (2)

- The *median* of a sequence  $A$  is a value  $m \in A$  such that half the values in  $A$  are less than or equal to  $m$  and the other half are greater than or equal to  $m$
- Generalizing, the ***k-smallest*** element of a sequence  $A$  is a value  $v \in A$  such that  $k$  elements of  $A$  are less than or equal to  $v$  and  $n - k$  are greater or equal to  $v$

E.g.,

- ▶ for  $k = 1$ , the minimum of  $A$
- ▶ for  $k = \lfloor |A|/2 \rfloor$ , the median of  $A$
- ▶ what is the *6th smallest* element of  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$ ?  
the 6th smallest element of  $A$ —a.k.a.  $select(A, 6)$ —is 8

# k-Smallest Element

- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_v$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_V$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

E.g.,  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

and we must compute the 7th smallest value in  $A$

- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_V$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

E.g.,  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

and we must compute the 7th smallest value in  $A$

we pick a splitting value, say  $v = 5$

- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_v$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

E.g.,  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

and we must compute the 7th smallest value in  $A$

we pick a splitting value, say  $v = 5$

$$A_L = \langle 2, 4, 1 \rangle$$



- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_v$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

E.g.,  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

and we must compute the 7th smallest value in  $A$

we pick a splitting value, say  $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle$$

- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_V$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

E.g.,  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

and we must compute the 7th smallest value in  $A$

we pick a splitting value, say  $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_V = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_v$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

E.g.,  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

and we must compute the 7th smallest value in  $A$

we pick a splitting value, say  $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

Now, where is the 7th smallest value of  $A$ ?

- Idea: we split the sequence  $A$  in three parts based on a *chosen value*  $v \in A$ 
  - ▶  $A_L$  contains the set of elements that are *less than*  $v$
  - ▶  $A_v$  contains the set of elements that are *equal to*  $v$
  - ▶  $A_R$  contains the set of elements that are *greater than*  $v$

E.g.,  $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$

and we must compute the 7th smallest value in  $A$

we pick a splitting value, say  $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

Now, where is the 7th smallest value of  $A$ ?

*It is the 2nd smallest value of  $A_R$*

We use  $select(A, k)$  to denote the  $k$ -smallest element of  $A$

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_V| \\ select(A_R, k - |A_L| - |A_V|) & \text{if } k > |A_L| + |A_V| \end{cases}$$

We use  $select(A, k)$  to denote the  $k$ -smallest element of  $A$

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_V| \\ select(A_R, k - |A_L| - |A_V|) & \text{if } k > |A_L| + |A_V| \end{cases}$$

- Computing  $A_L$ ,  $A_V$ , and  $A_R$  takes  $O(n)$  steps

We use  $select(A, k)$  to denote the  $k$ -smallest element of  $A$

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_V| \\ select(A_R, k - |A_L| - |A_V|) & \text{if } k > |A_L| + |A_V| \end{cases}$$

- Computing  $A_L$ ,  $A_V$ , and  $A_R$  takes  $O(n)$  steps
- How do we pick  $v$ ?

We use  $select(A, k)$  to denote the  $k$ -smallest element of  $A$

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_v| \\ select(A_R, k - |A_L| - |A_v|) & \text{if } k > |A_L| + |A_v| \end{cases}$$

- Computing  $A_L$ ,  $A_v$ , and  $A_R$  takes  $O(n)$  steps
- How do we pick  $v$ ?
- Ideally, we should pick  $v$  so as to obtain  $|A_L| \approx |A_R| \approx |A|/2$ 
  - ▶ so, ideally we should pick  $v = median(A)$ , but...



We use  $select(A, k)$  to denote the  $k$ -smallest element of  $A$

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_v| \\ select(A_R, k - |A_L| - |A_v|) & \text{if } k > |A_L| + |A_v| \end{cases}$$

- Computing  $A_L, A_v$ , and  $A_R$  takes  $O(n)$  steps
- How do we pick  $v$ ?
- Ideally, we should pick  $v$  so as to obtain  $|A_L| \approx |A_R| \approx |A|/2$ 
  - ▶ so, ideally we should pick  $v = median(A)$ , but...
- We pick *a random element of  $A$*

**SELECTION**( $A, k$ )

```
1   $v = A[\text{random}(1 \dots |A|)]$ 
2   $A_L, A_V, A_R = \emptyset$ 
3  for  $i = 1$  to  $|A|$ 
4      if  $A[i] < v$ 
5           $A_L = A_L \cup A[i]$ 
6      elseif  $A[i] == v$ 
7           $A_V = A_V \cup A[i]$ 
8      else  $A_R = A_R \cup A[i]$ 
9  if  $k \leq |A_L|$ 
10     return SELECTION( $A_L, k$ )
11 elseif  $k > |A_L| + |A_V|$ 
12     return SELECTION( $A_R, k - |A_L| - |A_V|$ )
13 else return  $v$ 
```