

## Chapter 2

### *Getting Started*

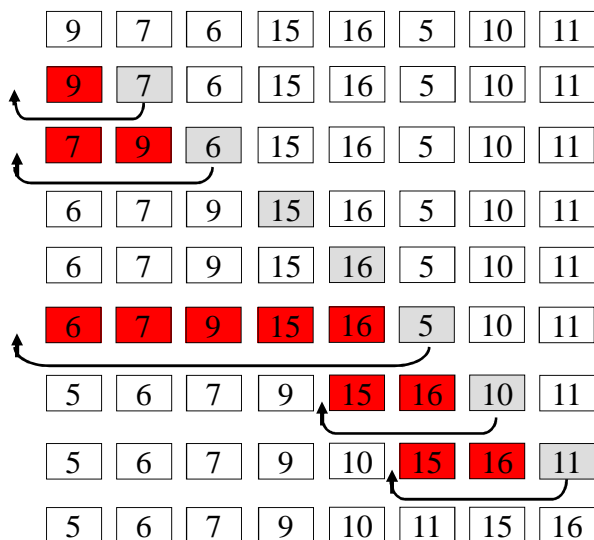
1

## Outlines: Getting Started

- ❖ Sequential Approach
  - [Insertion Sort](#)
- ❖ Divide-and-Conquer Approach
  - [Merge Sort](#)

2

## Insertion Sort Execution Example



3

## An Example: Insertion Sort

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

next key  
go left  
find place for key  
**shift sorted right**  
go left

put key in place

4

## An Example: Insertion Sort

30	10	40	20
1	2	3	4

$i = \emptyset$	$j = \emptyset$	$key = \emptyset$
$A[j] = \emptyset$	$A[j+1] = \emptyset$	

```

➔ InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
        }
        A[j+1] = key
    }
}

```

5

## An Example: Insertion Sort

30	10	40	20
1	2	3	4

$i = 2$	$j = 1$	$key = 10$
$A[j] = 30$	$A[j+1] = 10$	

```

➔ InsertionSort(A, n) {
    for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
        }
        A[j+1] = key
    }
}

```

6

## An Example: Insertion Sort

30	30	40	20
1	2	3	4

$i = 2$	$j = 1$	$key = 10$
$A[j] = 30$	$A[j+1] = 30$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

⇒

7

## An Example: Insertion Sort

30	30	40	20
1	2	3	4

$i = 2$	$j = 1$	$key = 10$
$A[j] = 30$	$A[j+1] = 30$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

⇒

8

## An Example: Insertion Sort

30	30	40	20
1	2	3	4

$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 30$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

9

## An Example: Insertion Sort

30	30	40	20
1	2	3	4

$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 30$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

10

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 10$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

11

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 10$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

12

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$	$j = 0$	$key = 40$
$A[j] = \emptyset$	$A[j+1] = 10$	



```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

13

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$	$j = 0$	$key = 40$
$A[j] = \emptyset$	$A[j+1] = 10$	



```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

14

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
  
```

⇒

15

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
  
```

⇒

16



## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

⇒

17

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 4$	$j = 2$	$key = 40$
$A[j] = 30$	$A[j+1] = 40$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

⇒

18

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
  
```

→

19

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}
  
```

→

20

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 20$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

21

## An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 20$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

22

## An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 40$	

⇒

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

23

## An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 40$	

⇒

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

24

## An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 3$	$key = 20$
$A[j] = 40$	$A[j+1] = 40$	



```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

25

## An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	



```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

26

## An Example: Insertion Sort

10	30	40	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 40$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

27

## An Example: Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 30$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

⇒

28

## An Example: Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$	$j = 2$	$key = 20$
$A[j] = 30$	$A[j+1] = 30$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

29

## An Example: Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$	$A[j+1] = 30$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

⇒

30

## An Example: Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$		$A[j+1] = 30$

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

→

31

## An Example: Insertion Sort

10	20	30	40
1	2	3	4

$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$		$A[j+1] = 20$

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

⇒

32



## An Example: Insertion Sort

10	20	30	40
1	2	3	4

$i = 4$	$j = 1$	$key = 20$
$A[j] = 10$	$A[j+1] = 20$	

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

Done!

33

## Insertion Sort

```

InsertionSort(A, n) {
  for i = 2 to n {
    key = A[i]
    j = i - 1;
    while (j > 0) and (A[j] > key) {
      A[j+1] = A[j]
      j = j - 1
    }
    A[j+1] = key
  }
}

```

How many times will this while loop execute?

34

## Insertion Sort

Statement	Effort
<code>InsertionSort(A, n) {</code>	
<code>for i = 2 to n {</code>	$c_1 n$
<code>key = A[i]</code>	$c_2 (n-1)$
<code>j = i - 1</code>	$c_3 (n-1)$
<code>while (j &gt; 0) and (A[j] &gt; key) {</code>	$c_4 C$
<code>A[j+1] = A[j]</code>	$c_5 (C - (n-1))$
<code>j = j - 1</code>	$c_6 (C - (n-1))$
<code>A[j+1] = key</code>	$c_7 (n-1)$
<code>}}</code>	

$C = t_2 + t_3 + \dots + t_n$  where  $t_i$  is number of **while** expression evaluations for the  $i^{\text{th}}$  for loop iteration

$$\begin{aligned} \text{Body of the while statement is executed} &= (t_2 - 1) + (t_3 - 1) + \dots + (t_n - 1) \\ &= t_2 + t_3 + \dots + t_n - (n-1) = C - (n-1) \end{aligned}$$

35

## Analyzing Insertion Sort

$$\begin{aligned} T(n) &= c_1 n + c_2 (n-1) + c_3 (n-1) + c_4 C + c_5 (C - (n-1)) + c_6 (C - (n-1)) + c_7 (n-1) \\ &= c_8 C + c_9 n + c_{10} \end{aligned}$$

- What can  $C$  be?
  - **Best case** -- inner loop body never executed (**array is sorted**)
    - $t_i = 1 \rightarrow T(n)$  is a linear function =  $a.n + b$
  - **Worst case** -- inner loop body executed for all previous elements (**array sorted in reverse order**)
    - $t_i = i \rightarrow T(n)$  is a quadratic function =  $a.n^2 + b.n + c$
  - *If  $T$  is a quadratic function, which terms in the above equation matter?*

$$\begin{aligned} &c_1 n \\ &c_2 (n-1) \\ &c_3 (n-1) \\ &c_4 C \\ &c_5 (C - (n-1)) \\ &c_6 (C - (n-1)) \\ &c_7 (n-1) \end{aligned}$$

36

## Analyzing Insertion Sort (Cont.)

- ❖ Best Case
  - If A is sorted:  $O(n)$  comparisons
- ❖ Worst Case
  - If A is reversed sorted:  $O(n^2)$  comparisons
- ❖ Average Case
  - If A is randomly sorted:  $O(n^2)$  comparisons

37

## Divide-and-Conquer

- ❖ Recursive in structure
  - *Divide* the problem into several smaller sub-problems that are similar to the original but smaller in size
  - *Conquer* the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - *Combine* the solutions to create a solution to the original problem

38

## An Example: Merge Sort

- ❖ *Divide*: Divide the  $n$ -element sequence to be sorted into two subsequences of  $n/2$  elements each
- ❖ *Conquer*: Sort the two subsequences recursively using merge sort.
- ❖ *Combine*: Merge the two sorted subsequences to produce the sorted answer.

39

## Merge Sort

```
MergeSort(A, left, right) {  
  if (left < right) {  
    mid = floor((left + right) / 2);  
    MergeSort(A, left, mid);  
    MergeSort(A, mid+1, right);  
    Merge(A, left, mid, right);  
  }  
}  
  
// Merge() takes two sorted subarrays of A and  
// merges them into a single sorted subarray of A.  
// It requires  $O(n)$  time
```

40

## Merge

```

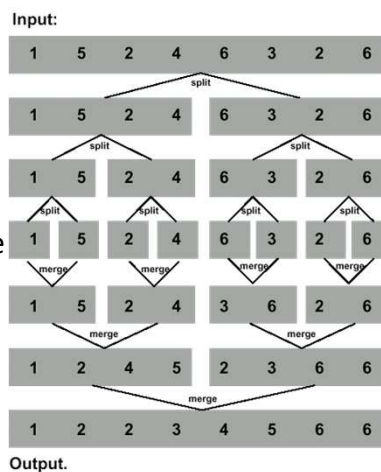
Merge (A, p, q, r)
n1 = q - p + 1
n2 = r - q
Create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if (L[i] ≤ R[j]) {
        A[k] = L[i]
        i = i + 1
    }
    else {
        A[k] = R[j]
        j = j + 1
    }

```

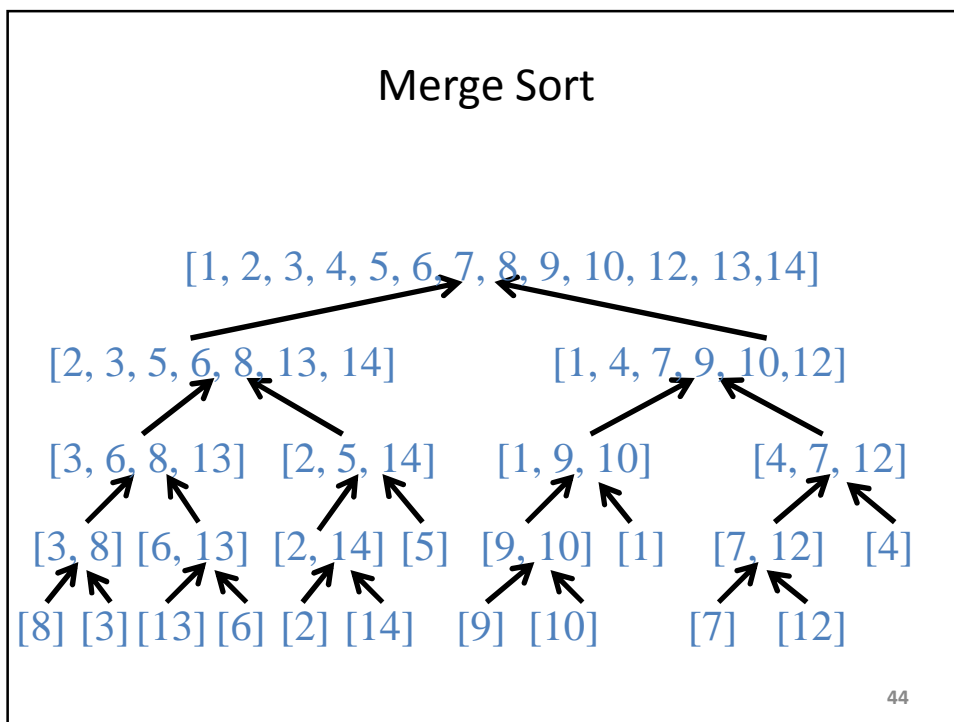
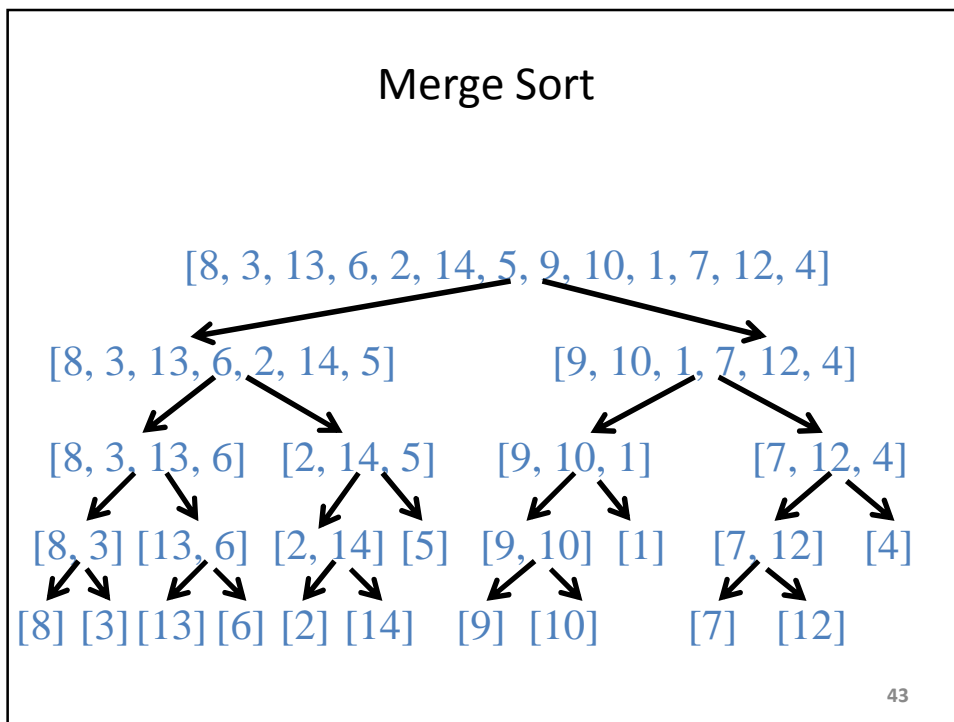
41

## Merge Sort Revisited

- ❖ To sort  $n$  numbers
  - if  $n = 1$  done!
  - recursively sort 2 lists of numbers  $\lfloor n/2 \rfloor$  and  $\lceil n/2 \rceil$  elements
  - merge 2 sorted lists in  $O(n)$  time
- ❖ Strategy
  - break problem into similar (smaller) subproblems
  - recursively solve subproblems
  - combine solutions to answer



42



## Analysis of Merge Sort

Statement	Effort
MergeSort(A, left, right) {	T(n)
if (left < right) {	Θ(1)
mid = floor((left + right) / 2);	Θ(1)
MergeSort(A, left, mid);	T(n/2)
MergeSort(A, mid+1, right);	T(n/2)
Merge(A, left, mid, right);	Θ(n)
}	
}	

- ❖ So  $T(n) = \Theta(1)$  when  $n = 1$ , and  
 $2T(n/2) + \Theta(n) + \Theta(1)$  when  $n > 1$
- ❖ Solving this recurrence (*how?*) gives  $T(n) = n \log n$
- ❖ This expression is a *recurrence*

45

## Review: Analysis of Merge Sort

- ❖ Divide: computing the middle takes  $O(1)$
- ❖ Conquer: solving 2 sub-problem takes  $2T(n/2)$
- ❖ Combine: merging  $n$ -element takes  $O(n)$
- ❖ Total:
 

$T(n) = O(1)$	if $n = 1$
$T(n) = 2T(n/2) + O(n) + O(1)$	if $n > 1$
- ⇒  $T(n) = O(n \lg n)$
- ❖ Solving this recurrence (*how?*) gives  $T(n) = O(n \lg n)$
- ❖ This expression is a *recurrence*
- ❖  $T(n)$  denote the number of operations required by an algorithm to solve a given class of problems.

46

## Analysis of Merge Sort

Assume  $n=2^k$  for  $k \geq 1$

$$T(n) = 2 T(n/2) + bn + c$$

$$T(n/2) = 2T((n/2)/2) + b(n/2) + c$$

$$= 2[2T(n/4) + b(n/2) + c] + bn + c$$

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

$$= 4 T(n/4) + 2bn + (1 + 2) c = 2^2 T(n/2^2) + 2bn + (2^0 + 2^1)c$$

$$= 4 [2T((n/4)/2) + b(n/4) + c] + 2bn + (1+2)c$$

$$= 8 T(n/8) + 3bn + (1+2+4)c$$

$$= 2^3 T(n/2^3) + 3bn + (2^0 + 2^1 + 2^2)c$$

$$\vdots$$

$$= 2^k T(n/2^k) + kbn + (2^0 + 2^1 + \dots + 2^{k-1})c$$

$$T(1) = a, \text{ since } n=2^k \rightarrow \log n = (\log 2^k) = k$$

$$T(n) = 2^k \cdot a + kbn + (2^0 + 2^1 + \dots + 2^{k-1})c$$

$$= n \cdot a + \log n \cdot bn + c \cdot \left( \sum_{i=0}^{k-1} 2^i = 2^{(k-1)+1} - 1 = 2^k - 1 = n - 1 \right)$$

$$= b \cdot n \log n + (a + c) n - c$$

$$= O(n \log n) \quad \text{Worst case}$$

## Summary

### ❖ Sequential Approach

#### ➤ Insertion Sort:

Example

Time Complexity (Worst, Best, & Average Cases)

Space Complexity

Advantages and Disadvantages

### ❖ Divide-and-Conquer Approach

#### ➤ Merge Sort:

Example

Time Complexity (Worst Case)

Space Complexity

Advantages and Disadvantages