# Ch.19 Divide and Conquer

SNU Internet DataBase Lab.

# BIRD'S-EYE VIEW

- Divide and conquer algorithms
  - Decompose a problem instance into several smaller independent instances
  - May be effectively run on a parallel computer
  - Min-max problem, matrix multiplication and so on

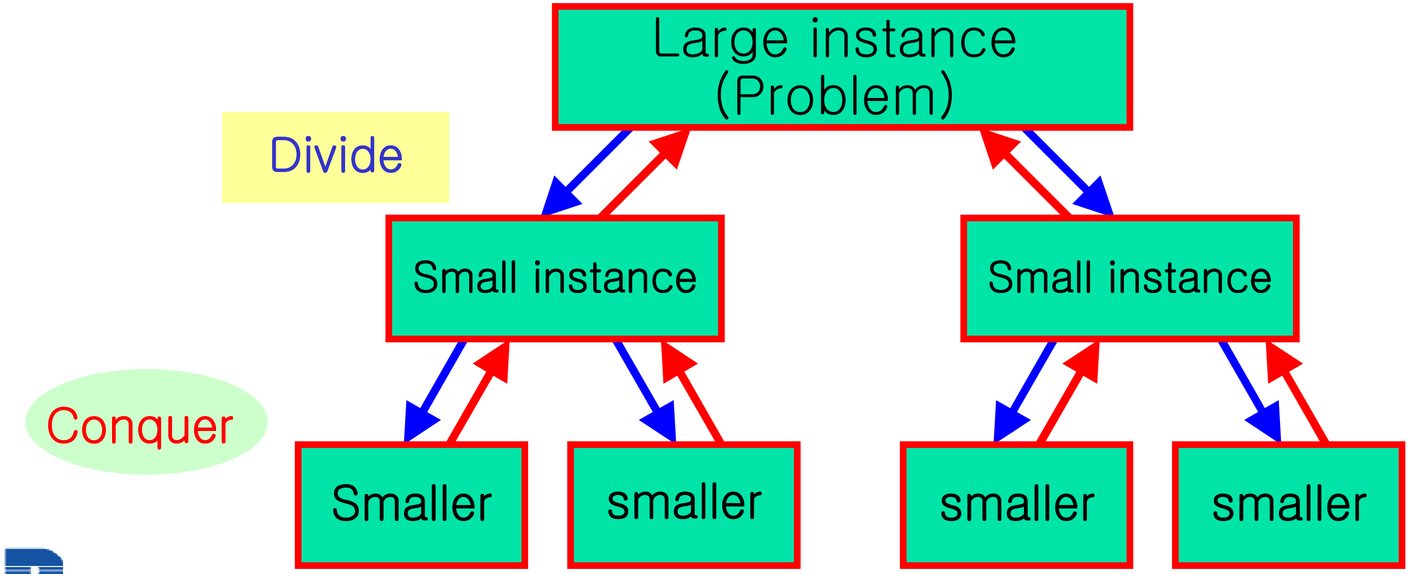
- This chapter
  - Develops the mathematics needed to analyze the complexity of divide and conquer algorithms
  - Proves that the divide and conquer algorithms for the min-max and sorting problems are optimal

# Table of contents


- The **Divide and Conquer method**
  - **Solving a small instance**
  - **Solving a large instance**
- Applications
  - Divide and Conquer Sorting
    - Insertion Sort
    - Selection Sort
    - Bubble Sort
    - Merge Sort
    - Quick Sort

# Divide and Conquer (1)

- Distinguish between small and large instances
- Small instances solved differently from large ones
- All instances are non-overlapping




SNU Internet DataBase Lab.

# Divide and Conquer (2)

- A small instance is solved using some direct/simple strategy
  - Sort a list that has $n$ ($\leq 10$) elements
    - Use count, insertion, bubble, or selection sort
  - Find the minimum of $n$ ($\leq 2$) elements
    - When $n = 0$, there is no minimum element
    - When $n = 1$, the single element is the minimum
    - When $n = 2$, compare the two elements and determine which is smaller

- A large instance is solved as follows:
  - Divide the large instance into $k$ ($\geq 2$) smaller instances
  - Solve the smaller instances somehow
  - Combine the results of the smaller instances to obtain the result for the original large instance

# Table of contents

- The Method
  - Divide and Conquer
    - Solving a small instance
    - Solving a large instance
- Applications
  - **Divide and Conquer Sorting**
    - **Insertion Sort**
    - **Selection Sort**
    - **Bubble Sort**
    - Merge Sort
    - Quick Sort

SNU Internet DataBase Lab.

# Divide and Conquer Sorting (1)

- Sort n elements into nondecreasing order
- Divide-and-conquer sorting algorithm
  - If n is 1,
    - Terminate
  - Otherwise,
    - Partition the large instance of n elements into two or more small instances
    - Sort each small instances
    - Combine the sorted small instances into a single sorted instance

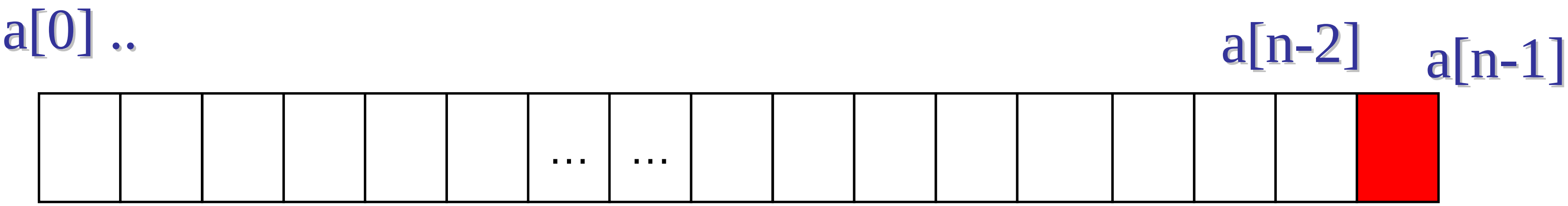
- Divide-and-conquer algorithms have best complexity when a large instance is divided into small instances of approximately the same size
  - When $k = 2$ and $n = 24$, divide into two small instances of size 12 each
  - When $k = 2$ and $n = 25$, divide into two small instances of size 13 and 12, respectively

SNU Internet DataBase Lab.

# Divide and Conquer Sorting (2)

- Partitioning Schemes
  - Partitioning the n elements into two unbalanced collections (i.e., n-1elements & 1 element)
    - All three sort methods in this manner take $O(n^2)$ time
      - Insertion sort
      - Selection sort
      - Bubble sort
  - Partitioning the n elements into two balanced collections (i.e., n/k element & the rest elements into 2 groups)
    - The following methods in this manner take O(n log n) time
      - Merge sort
      - Quick sort

# Insertion Sort by Divide & Conquer




- k = 2 divide-and-conquer sorting method
  - Complexity is $O(n^2)$
- Divide Phase
  - First n - 1 elements (a[0:n-2]) define the first small instance
  - Last element (a[n-1]) defines the second small instance
  - a[0: n-2] is sorted recursively
- Conquer Phase
  - Combining is done by inserting a[n-1] into the sorted a[0:n-2]
- Here we show the recursive solution, but normally implemented non-recursively

SNU Internet DataBase Lab.

# Example for Insertion Sort

Original Array →

| | | | | |
|---|---|---|---|---|
| 4 | 3 | 5 | 1 | 2 |

Divide it into two arrays →

| | | | | |
|---|---|---|---|---|
| 4 | 3 | 5 | 1 | |

| |
|---|
| 2 |

Sort the 1st array recursively →

| | | | |
|---|---|---|---|
| 1 | 3 | 4 | 5 |

| |
|---|
| 2 |

Conquer the two arrays
: Insert the 2nd array Into the 1st array →

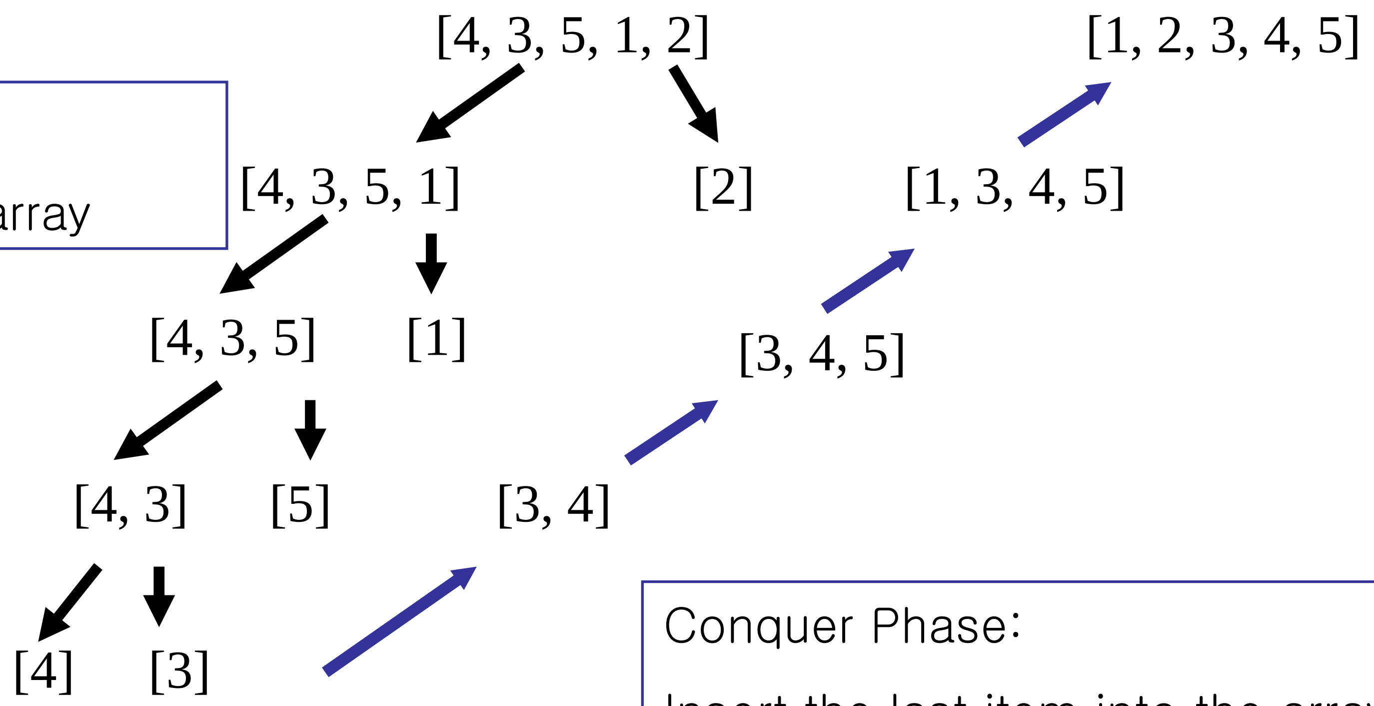
| | | | | |
|---|---|---|---|---|
| 1 | 3 | 4 | 5 | 2 |
| 1 | 3 | 4 | 2 | 5 |
| 1 | 3 | 2 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |

SNU Internet DataBase Lab.

# Insertion Sort Example

Divide Phase:

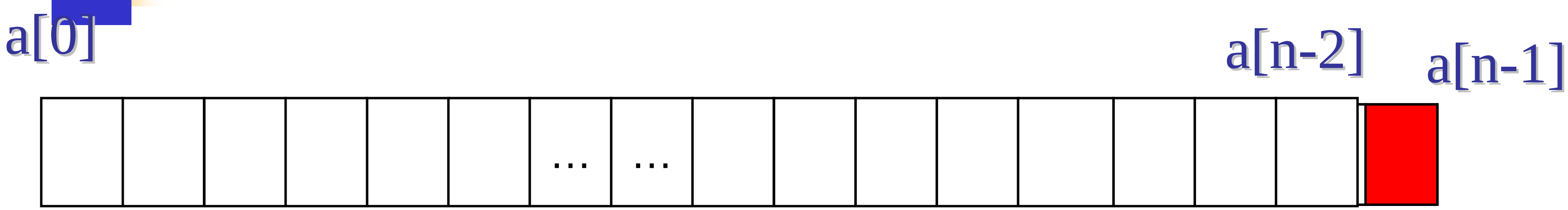
Just split the array




Conquer Phase:

Insert the last item into the array

# Selection Sort by Divide & Conquer




- k = 2 divide-and-conquer sorting method
  - Complexity is $O(n^2)$
- Divide Phase: To divide a large collection into two smaller instances
  - First find the largest element & The largest element defines one small instance
  - The remaining n-1 elements define the second small instance
  - The second small instance is sorted recursively
- Conquer Phase: Append the first smaller instance (largest element) to the right end of the sorted second small instance
- Here we show the recursive solution, but normally implemented non-recursively

SNU Internet DataBase Lab.

# Example for Selection Sort

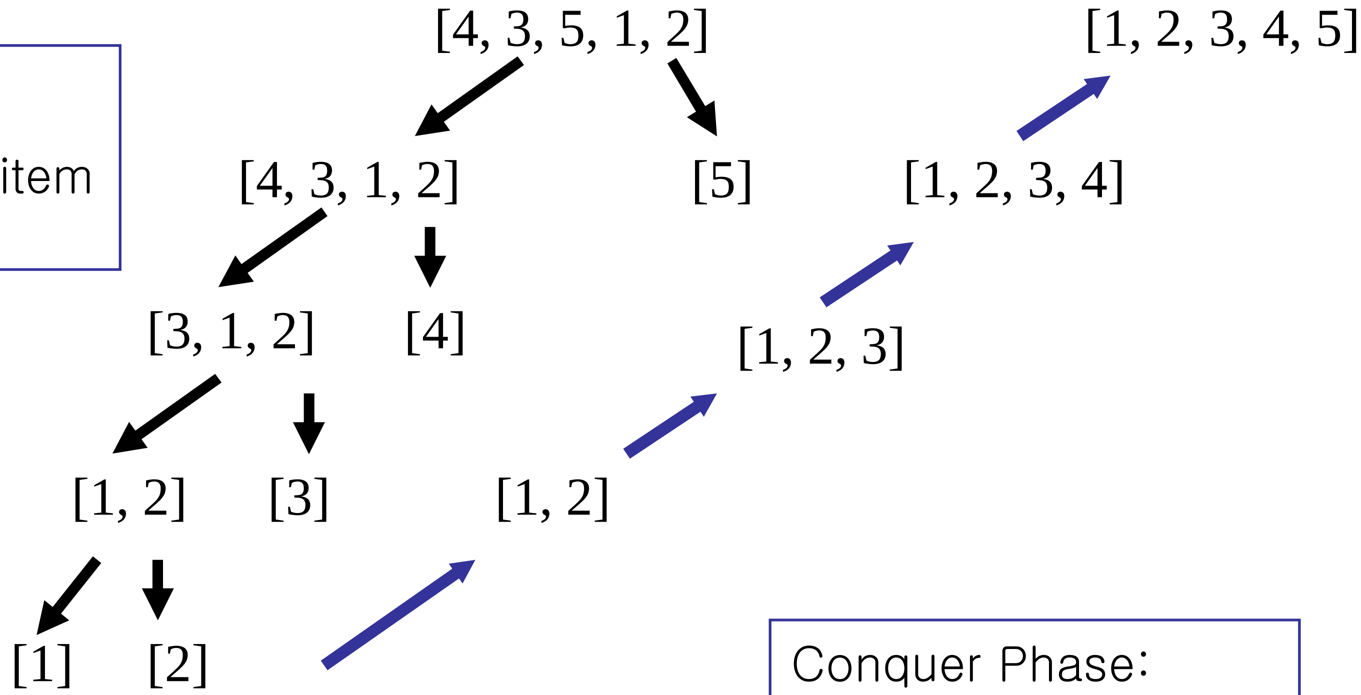
| | |
|---|---|
| Original Array → | 4 3 5 1 2 |
| Find the largest element → | 4 3 **5** 1 2 |
| Divide it into two arrays → | 4 3 1 2    **5** |
| Sort the 1st array recursively → | 1 2 3 4    **5** |
| Conquer the two arrays<br>: Just append the 2nd array to the right of the 1st array → | 1 2 3 4 **5** |

SNU Internet DataBase Lab.

# Selection Sort Example

Divide Phase:

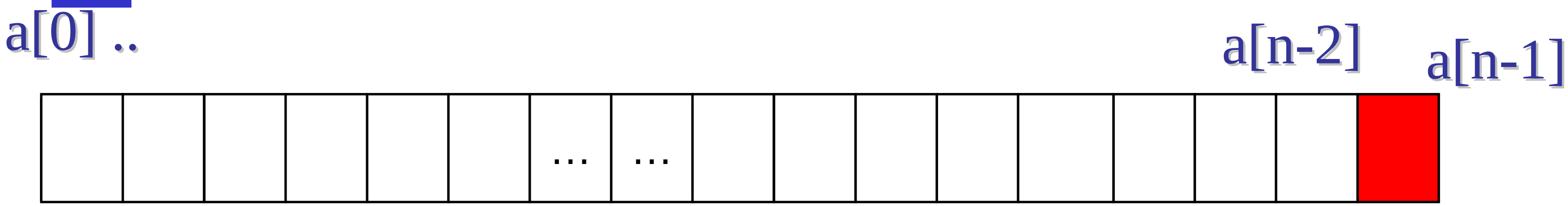
Move the largest item by max process




Conquer Phase:

Merge two arrays

# Bubble Sort by Divide & Conquer




- k = 2 divide-and-conquer sorting method
  - Complexity is $O(n^2)$

- Divide Phase: To divide a large collection into two smaller instances
  - First find the largest element by bubbling (a series of swapping)
  - The largest element defines one small instance
  - The remaining n-1 elements define the second small instance
- Conquer Phase: Merge two arrays
- Here we show the recursive solution, but normally implemented non-recursively

SNU Internet DataBase Lab.

# Example for Bubble Sort

Original Array →

| | | | | |
|---|---|---|---|---|
| 4 | 3 | 5 | 1 | 2 |

Move the largest element to the right end by bubbling process →

| | | | | |
|---|---|---|---|---|
| 3 | 4 | 5 | 1 | 2 |
| 3 | 4 | 5 | 1 | 2 |
| 3 | 4 | 1 | 5 | 2 |
| 3 | 4 | 1 | 2 | 5 |

Divide it into two arrays →

| | | | | | |
|---|---|---|---|---|---|
| 3 | 4 | 1 | 2 | | 5 |

Sort the 1st array recursively →

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | | 5 |

Conquer the two arrays
: Just append the 2nd array to the right of the 1st array →

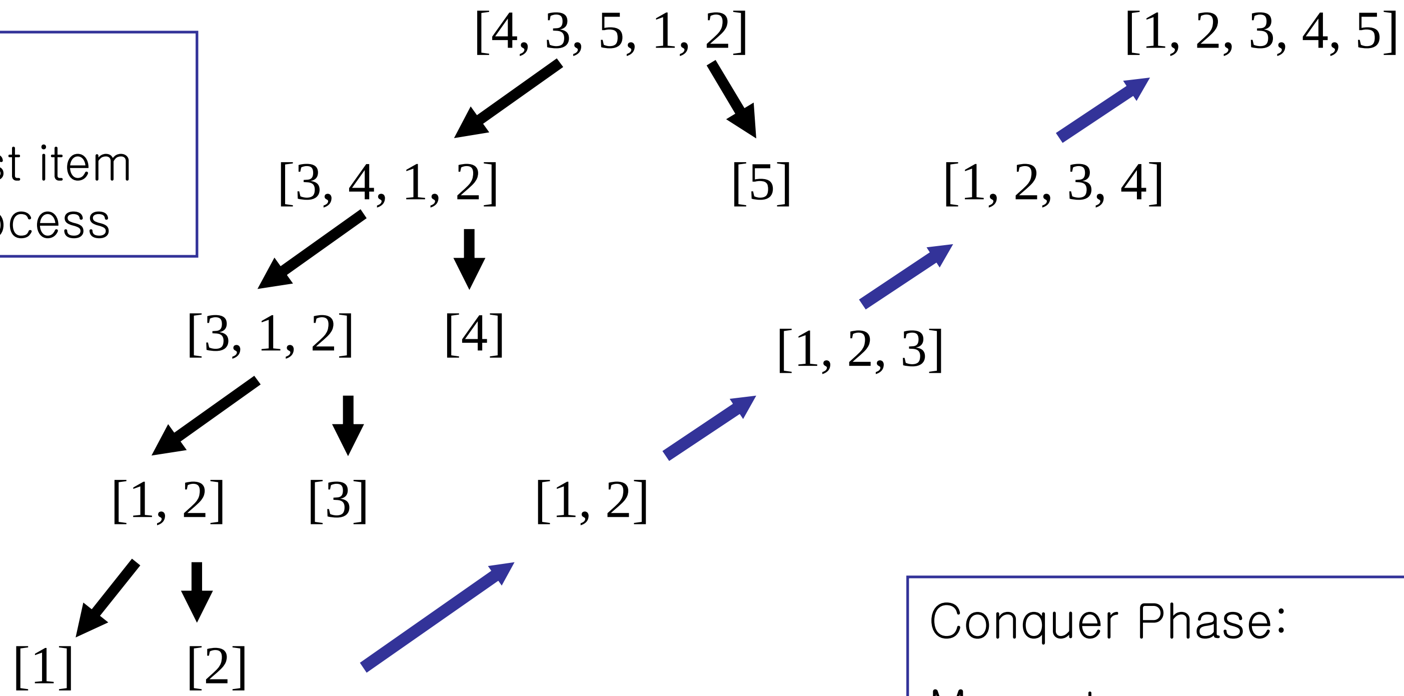
| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

SNU Internet DataBase Lab.

# Bubble Sort Example

Divide phase:

Move the largest item by Bubbling process




Conquer Phase:

Merge two arrays

SNU Internet DataBase Lab.

# Table of contents

- The Method
  - Divide and Conquer
    - Solving a small instance
    - Solving a large instance
- Applications
  - Divide and Conquer Sorting
    - Insertion Sort
    - Selection Sort
    - Bubble Sort
    - **Merge Sort**
    - **Quick Sort**

# Merge Sort by Divide and Conquer (1)

- k = 2 divide-and-conquer sorting method

- Divide Step
  - First ceil(n/2) elements define one of the smaller instances
  - Remaining floor(n/2) elements define the second smaller instance
- Conquer Step
  - Each of the two smaller instances is sorted recursively
  - The sorted smaller instances are combined using a merge process

- The complexity of merge sort is O(n log n)
- Here we show the recursive solution, but normally implemented non-recursively

# Merge Sort by Divide and Conquer (2)

- An Example for Merge Process
  - A large instance divided into two instances (A and B) and sort them

    A = (2, 5, 6) B = (1, 3, 8, 9, 10) C = ( )

  - Compare smallest elements of A and B and merge smaller into C

    A = (2, 5, 6) B = (3, 8, 9, 10) C = (1)
    A = (5, 6) B = (3, 8, 9, 10) C = (1, 2)
    A = (5, 6) B = (8, 9, 10) C = (1, 2, 3)
    A = (6) B = (8, 9, 10) C = (1, 2, 3, 5)
    A = ( ) B = (8, 9, 10) C = (1, 2, 3, 5, 6)

  - When one of A and B becomes empty, append the other list to C

    A = ( ) B = ( ) C = (1, 2, 3, 5, 6, 8, 9, 10)

- O(1) time needed to move an element into C
- Total time is O(n + m), where n and m are the number of elements initially in A and B

SNU Internet DataBase Lab.

# Merge Sort: Example (1)

- Divide Phase

[8, 3, 13, 6, 2, 14, 5, 9, 10, 1, 7, 12, 4]

[8, 3, 13, 6, 2, 14, 5] [9, 10, 1, 7, 12, 4]

[8, 3, 13, 6] [2, 14, 5] [9, 10, 1] [7, 12, 4]

[8, 3] [13, 6] [2, 14] [5] [9, 10] [1] [7, 12] [4]

[8] [3] [13] [6] [2] [14] [9] [10] [7] [12]

# Merge Sort: Example (2)

- Conquer Phase: all sub components are sorted

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13,14]

[2, 3, 5, 6, 8, 13, 14]     [1, 4, 7, 9, 10,12]

[3, 6, 8, 13]   [2, 5, 14]   [1, 9, 10]   [4, 7, 12]

[3, 8] [6, 13]  [2, 14] [5]  [9, 10] [1]  [7, 12] [4]

[8] [3] [13] [6] [2] [14]  [9] [10]  [7] [12]

SNU Internet DataBase Lab.

# Merge Sort Analysis

- Number of leaf nodes is n
- Number of nonleaf nodes is n-1

- Downward pass over the recursion tree
  - Divide large instances into small ones
  - O(1) time at each node
  - O(n) total time at all nodes
- Upward pass over the recursion tree
  - Merge pairs of sorted lists
  - O(n) time merging at each level that has a nonleaf node
  - Number of levels is O(log n)
  - Total time is O(n log n)

SNU Internet DataBase Lab.

# Quick Sort by Divide and Conquer

- Small instance has n <= 1.
  - So, Every small instance is a sorted instance
- To sort a large instance, select a pivot element from out of the n elements
- Partition the n elements into 3 groups left, middle and right
  - The middle group contains only the pivot element
  - All elements in the left group are <= pivot
  - All elements in the right group are >= pivot
- Sort left and right groups recursively
- combine left group, middle group and right group

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 8 | 5 | 11 | 10 | 4 | 1 | 9 | 7 | 3 |

Use 6 as the pivot

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 4 | 1 | 3 | 6 | 7 | 9 | 10 | 11 | 8 |

Sort left and right groups recursively

SNU Internet DataBase Lab.

# Quick Sort by Divide and Conquer: Choice of Pivot

- Leftmost element in list that is to be sorted
  - When sorting a[6:20], use a[6] as the pivot
  - Text implementation does this

- Randomly select one of the elements to be sorted as the pivot
  - When sorting a[6:20], generate a random number r in the range [6, 20].
  - Use a[r] as the pivot

- Median-of-Three rule. From the leftmost, middle, and rightmost elements of the list to be sorted, select the one with median key as the pivot
  - When sorting a[6:20], examine a[6], a[13] ((6+20)/2), and a[20].
  - Select the element with median (i.e., middle) key

  - If a[6].key = 30, a[13].key = 2, and a[20].key = 10, a[20] becomes the pivot
  - If a[6].key = 3, a[13].key = 2, and a[20].key = 10, a[6] becomes the pivot
  - If a[6].key = 30, a[13].key = 25, and a[20].key = 10, a[13] becomes the pivot

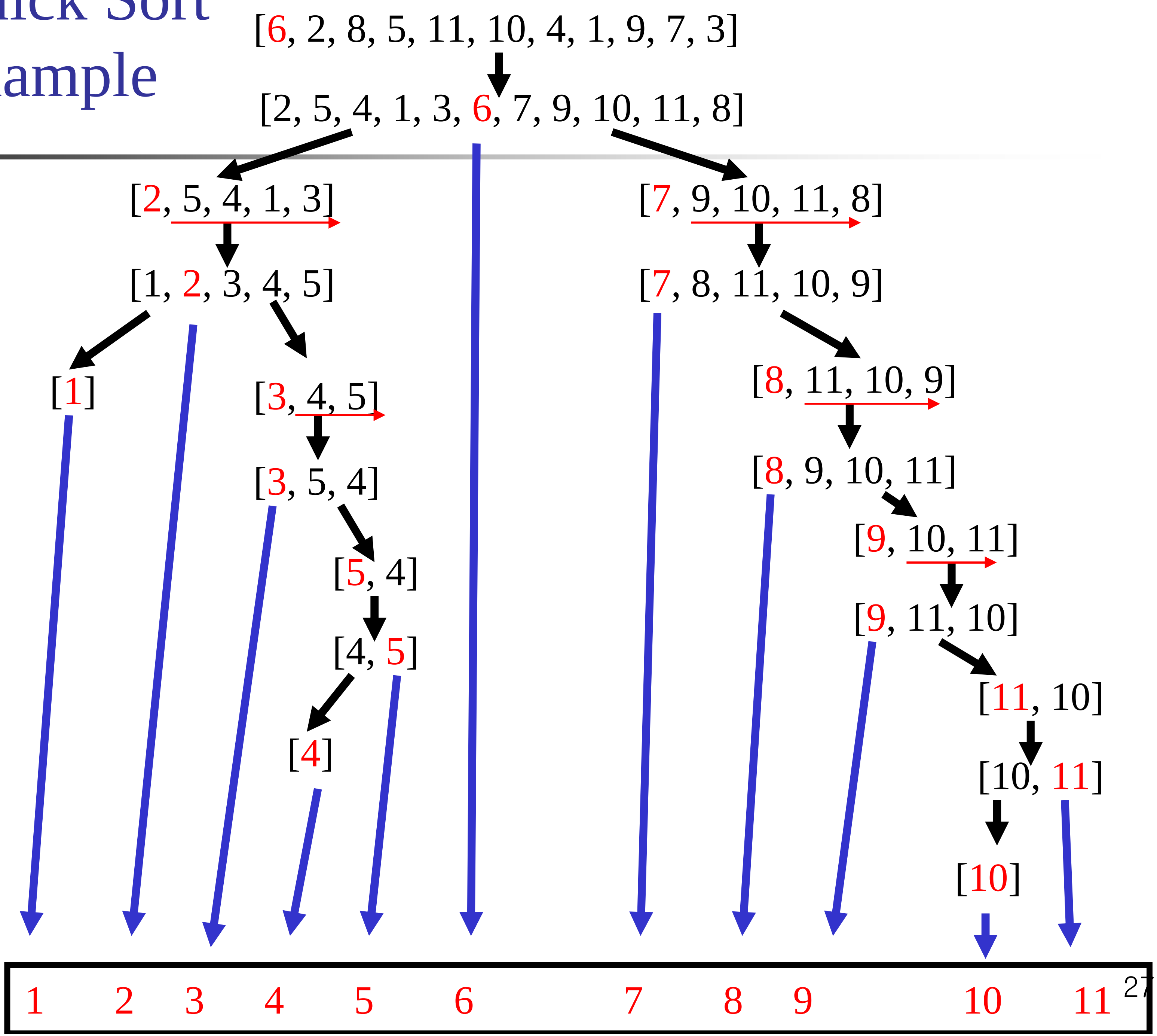
# Quick Sort by Divide and Conquer: Partitioning

- Sort a = [6, 2, 8, 5, 11, 10, 4, 1, 9, 7, 3]
- Suppose the leftmost element “6” is the pivot
- When another array b is available:
  - Scan a from left to right (omit the pivot in this scan), placing elements <= pivot at the left end of b and the remaining elements at the right end of b
  - The pivot is placed at the remaining position of the b

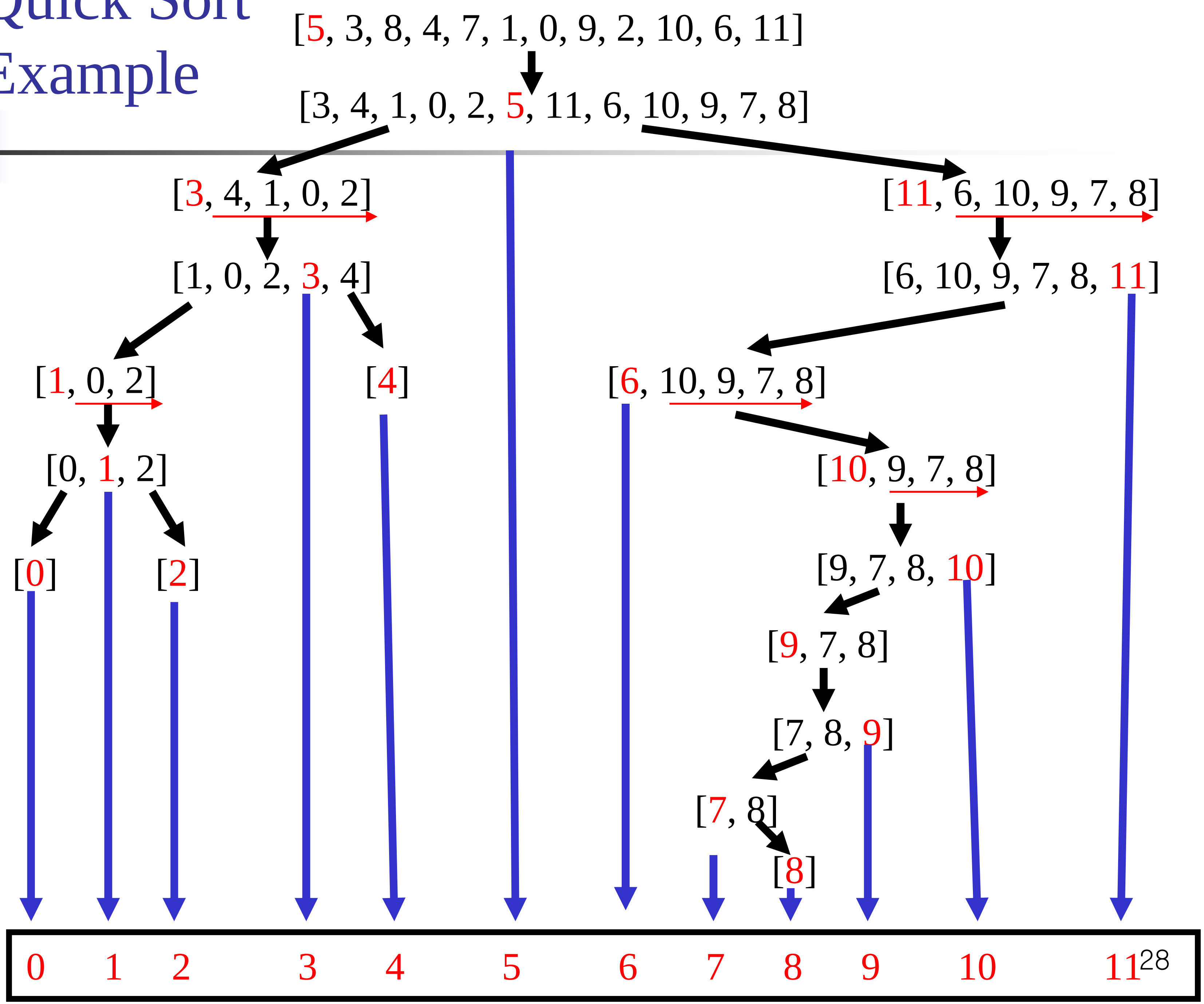
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | 6 | 2 | 8 | 5 | 11 | 10 | 4 | 1 | 9 | 7 | 3 |
| b | 2 | 5 | 4 | 1 | 3 | 6 | 7 | 9 | 10 | 11 | 8 |

Sort left and right groups recursively

# Quick Sort Example



# Quick Sort Example



# Time Complexity of Quick Sort

- $O(n)$ time to partition an array of $n$ elements
- Let $t(n)$ be the time needed to sort $n$ elements
  - $t(0) = t(1) = c$, where $c$ is a constant
  - When $t > 1$,

    $t(n) = t(|left|) + t(|right|) + dn$, where $d$ is a constant
  - $t(n)$ is maximum when either $|left| = 0$ or $|right| = 0$ following each partitioning
- Overall Time Complexity
  - The worst-case computing time for quick sort is $\Theta(n^2)$
    - When left is always empty
  - The best-case computing time for quick sort is $\Theta(n*\log n)$
    - When left and right are always of about the same size
  - The average complexity of quick sort is also $\Theta(n*\log n)$
    - Theorem 19.2 in your textbook

SNU Internet DataBase Lab.

# BIRD'S-EYE VIEW

- Divide and conquer algorithms
  - Decompose a problem instance into several smaller independent instances
  - May be effectively run on a parallel computer
  - Min-max problem, matrix multiplication and so on

- This chapter
  - Develops the mathematics needed to analyze the complexity of divide and conquer algorithms
  - Proves that the divide and conquer algorithms for the min-max and sorting problems are optimal

# Table of contents

- The **Divide and Conquer method**
  - **Solving a small instance**
  - **Solving a large instance**
- Applications
  - Divide and Conquer Sorting
    - Insertion Sort
    - Selection Sort
    - Bubble Sort
    - Merge Sort
    - Quick Sort