Data structure and algorithms
Spring 2025

Sorting Algorithms I

Lecturer: Do Thuy Duong

Outline

- Introduction to Sorting
- Selection Sort
- Inserting Sort
- Bubble Sort
- Merge Sort

Introduction to Sorting

- Definition: Rearrange a sequence of elements into numerical order based on the sort key(s).
- Two major types:
  - Internal sorting: During sorting, the data is in main memory (RAM)
  - External sorting:
    - During sorting, some data is in RAM, some other is in secondary (external) storage.
    - Used for large amount of data.

Introduction to Sorting

- **Stability**: Whether data with **equal key** values maintain their relative input order in the output.
  - (**4**, 2) (**3**, 7) (**3**, 1) (**5**, 6) input sequence
  - (**3**, 7) (**3**, 1) (**4**, 2) (**5**, 6) (order maintained) output 1
  - (**3**, 1) (**3**, 7) (**4**, 2) (**5**, 6) (order changed) output 2
- **Efficiency**: A measure of the relative efficiency (**time complexity**) of a sort.
    - Usually based on number of comparisons and moves during sorting.
- During sorting, the data are traversed many times. Each such traversal is called a **sort pass**.

Selection Sort - Definition

1. **Select** the largest (or smallest) number from unsorted range.
2. Swap it to the end (or the beginning) of the unsorted list.
3. Reduce the range by one and repeat step 1 until range = 1

0	1	2	3
3	7	2	5

0	1	2	3
3	5	2	7

Iteration 1: Find largest among A[0..3]. It is **A[1]=7**
So: swap A[3] and A[1].
We don't need to consider **A[3]** anymore.

0	1	2	3
3	5	2	7

0	1	2	3
3	2	5	7

Iteration 2: Find largest among A[0..2]. It is **A[1]=5**
So: swap A[2] and A[1].
We don't need to consider **A[2]** anymore.

0	1	2	3
3	2	5	7

0	1	2	3
2	3	5	7

Iteration 3: Find largest among A[0..1]. It is **A[0]=3**
So: swap A[0] and A[1].
We don't need to consider **A[1]** anymore.

Selection Sort – Pseudo code

Code

```
Algorithm SelectionSort(A, n):
    Input: An array A[0..n-1] storing n integers.
    Output: A is sorted in ascending order

    for i← n-1 down to 1 do
        Max ← A[i]
        Pos ← i
        for j← i-1 down to 0 do
            if Max < A[j] then
                Max ← A[j]
                Pos ← j

        A[Pos] ← A[i]
        A[i] ← Max
```

Selection Sort – Time complexity

Code

$O(n^2)$

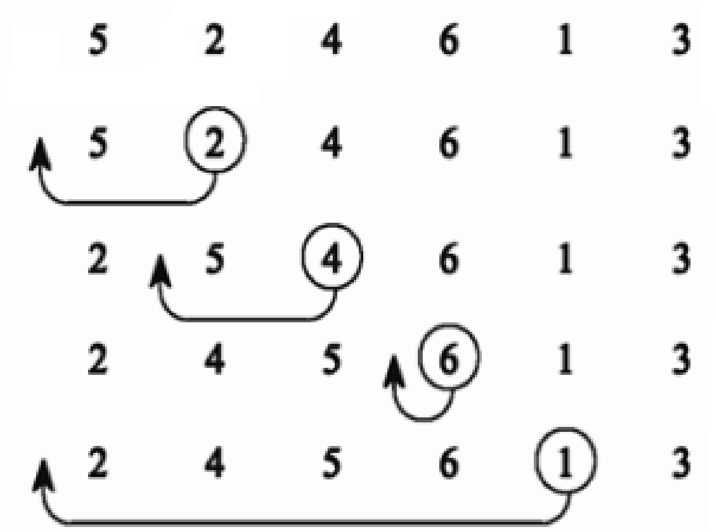
```
Algorithm SelectionSort(A, n):
    Input: An array A[0..n-1] storing n integers.
    Output: A is sorted in ascending order

    for i← n-1 down to 1 do                    O(n).O(n)=O(n²)
O(1)  Max ← A[i]
      Pos ← i
      for j← i-1 down to 0 do                 O(n).O(1)=O(n)
          if Max < A[j] then                   O(1)
             Max ← A[j]
             Pos ← j             O(1)

      A[Pos] ← A[i]
      A[i] ← Max            O(1)
```

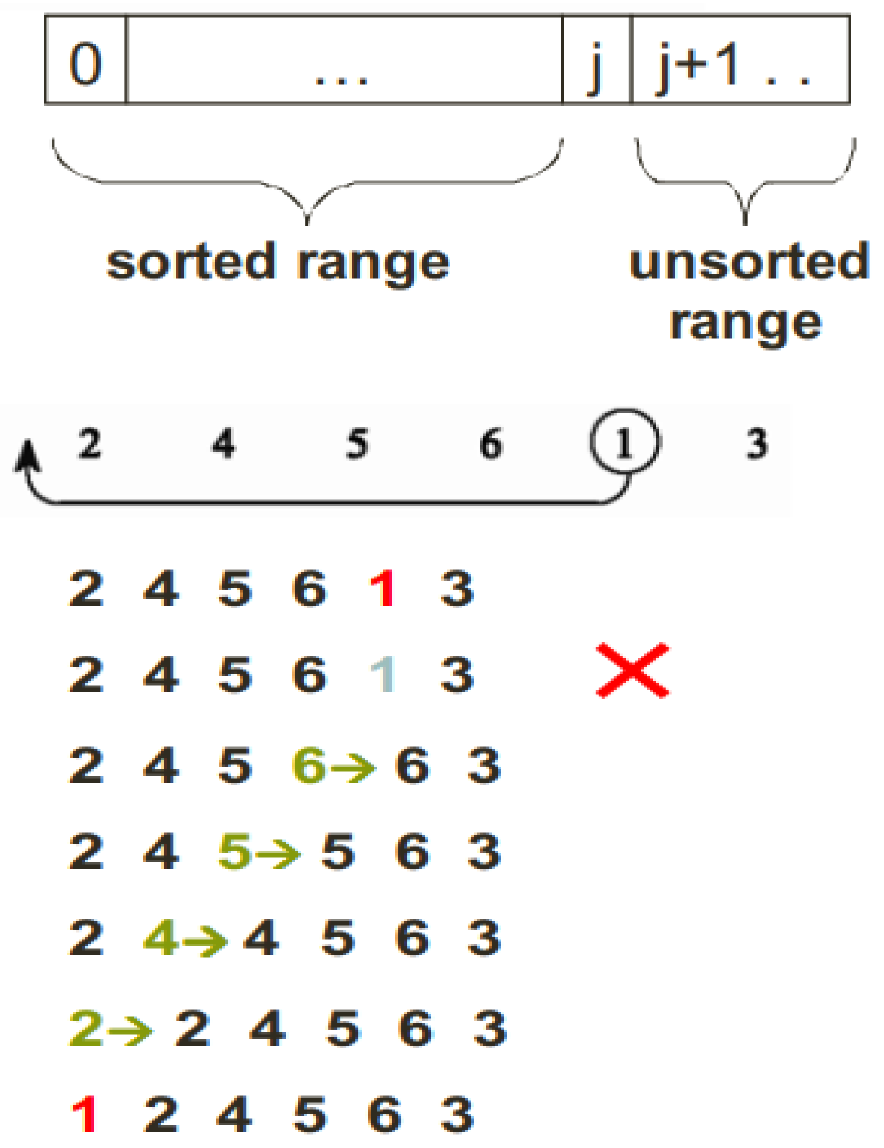
Insertion Sort - Definition

- Idea: Sort a set of elements by inserting unsorted elements into existing sorted list.
    1. Consider the 1st element as sorted range
    2. Compare the next element and **Insert** it in the correct order of the sorted range.
    3. Increase the sorted range and repeat step 2.



Insertion Sort - Example

- Sort A[0,1,2…,n-1]
- Consider A[j]
- Move A[0,1,…,j-1] to the right until correct position for A[j] is found



Insertion Sort – Pseudo code

Code

Algorithm InsertionSort(A, *n*):
Input: An array A[0..n-1] storing n integers.
Output: A is sorted in ascending order

for $j \leftarrow 1$ **to** $n-1$ **do**
$Key \leftarrow A[j]$
$i \leftarrow j-1$

while $i>=0$ **and** $A[i]>Key$
$A[i+1] \leftarrow A[i]$
$i \leftarrow i-1$

$A[i+1] \leftarrow Key$

Insertion Sort – Time complexity

Code

$O(n^2)$

Algorithm InsertionSort(A, *n*):
Input: An array A[0..n-1] storing n integers.
Output: A is sorted in ascending order

for $j \leftarrow 1$ **to** $n-1$ **do** $O(n).O(n)=O(n^2)$
  $Key \leftarrow A[j]$
  $i \leftarrow j-1$ $O(1)$

  while $i>=0$ **and** $A[i]>Key$ $O(n).O(1)=O(n)$
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i-1$ $O(1)$

  $A[i+1] \leftarrow Key$ $O(1)$

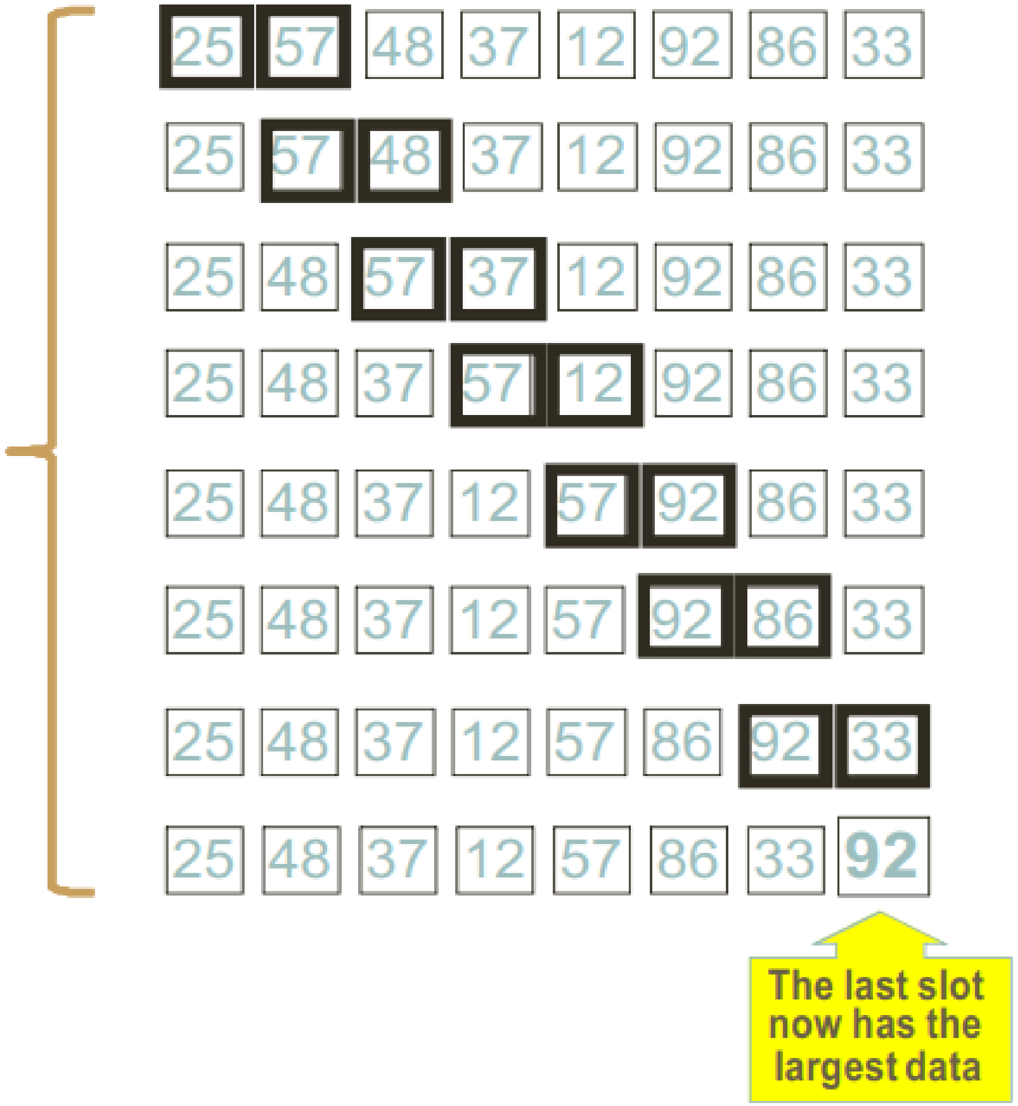
Bubble Sort - Definition

- Idea: Bubble Sort is similar to bubbles in water, the bigger ones will raise faster to the surface
- Algorithm:

  1. Scan the array from left to right, **exchange pairs** of elements that are **out-of-order**.
  2. Repeat the above process for (N-1) time where N is the number of elements in the array.

Bubble Sort - Example

- (1st pass)→



Bubble Sort - Example

(2nd pass)

25	48	37	12	57	86	33	92
25	48	37	12	57	86	33	92
25	37	48	12	57	86	33	92
25	37	12	48	57	86	33	92
25	37	12	48	57	86	33	92
25	37	12	48	57	86	33	92
25	37	12	48	57	33	**86**	92

This slot now has the 2nd largest data

(3rd pass)

25	48	37	12	57	33	86	92
25	48	37	12	57	33	86	92
25	37	48	12	57	33	86	92
25	37	12	48	57	33	86	92
25	37	12	48	57	33	86	92
25	37	12	48	33	**57**	86	92

This slot now has the 3rd largest data

Bubble Sort - Example

Result:

- Original: 25 57 48 37 12 92 86 33
- After pass 1: 25 48 37 12 57 86 33 92
- After pass 2: 25 37 12 48 57 33 86 92
- After pass 3: 25 12 37 48 33 57 86 92
- After pass 4: 12 25 37 33 48 57 86 92
- After pass 5: 12 25 33 37 48 57 86 92
- After pass 6: 12 25 33 37 48 57 86 92
- After pass 7: 12 25 33 37 48 57 86 92

Bubble Sort – Pseudo code

Code

```
Algorithm BubbleSort(A, n):
    Input: An array A[0..n-1] storing n integers.
    Output: A is sorted in ascending order

    for i ← 0 to n-2 do
     for j ← 0 to n-i-2 do
       if A[j] > A[j+1] then
          Tmp ← A[j]
          A[j] ← A[j+1]
          A[j+1] ← Tmp
```

Bubble Sort - Optimized

Result:

- Original: 25 57 48 37 12 92 86 33
- After pass 1: 25 48 37 12 57 86 33 92
- After pass 2: 25 37 12 48 57 33 86 92
- After pass 3: 25 12 37 48 33 57 86 92
- After pass 4: 12 25 37 33 48 57 86 92
- After pass 5: 12 25 33 37 48 57 86 92
- After pass 6: 12 25 33 37 48 57 86 92
- After pass 7: 12 25 33 37 48 57 86 92

Note: if the result of each pass is examined carefully, only 5 passes are needed for the sorting.

Bubble Sort - Optimized

Code

```
Algorithm BubbleSort2(A, n):
    Input: An array A[0..n-1] storing n integers.
    Output: A is sorted in ascending order

    isSorted ← False
    while isSorted = False
        isSorted ← True
        for j ← 0 to n-2 do
            if A[j] > A[j+1] then
                Tmp ← A[j]
                A[j] ← A[j+1]
                A[j+1] ← Tmp
                isSorted ← False
```

Bubble Sort – Time complexity

- Worst case Time Complexity: $O(n^2)$

Merge Sort - Definition

- Based on **Divide and Conquer Approach**:
  - **Divide:** Divide the data into 2 or more disjoint subsets
  - **Recursion:** Solve the sub-problems associated with the subsets
  - **Conquer:** Take the solutions to the sub-problems and "merge" these solutions into a solution to the Big original problem.

Merge Sort - Algorithm

• **Divide**: If S has at leas two elements (nothing needs to be done if S has zero or one elements), remove all the elements from S and put them into two sequences, S1 and S2, each containing about half of the elements of S.
(i.e. S1 contains the first $\lceil n/2 \rceil$ elements and S2 contains the remaining $\lfloor n/2 \rfloor$ elements.

• **Recursion**: Recursive sort sequences S1 and S2.

• **Conquer**: Put back the elements into S by merging the sorted sequences S1 and S2 into a unique sorted sequence.

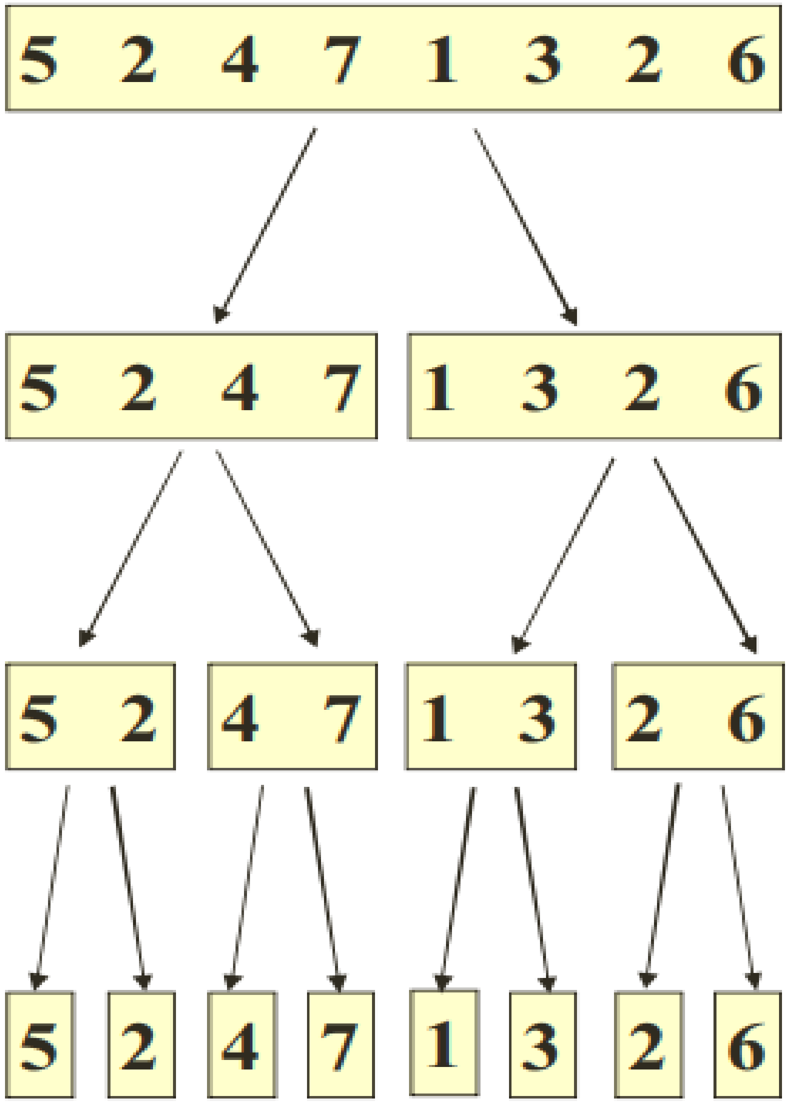
Merge Sort - Example

At the beginning, a Mr. MergeSort is called to sort:

Then 2 other Mr. MergeSorts are called to sort:

Then 4 other Mr. MergeSorts are called to sort:

Then 8 other Mr. MergeSorts are called to sort:



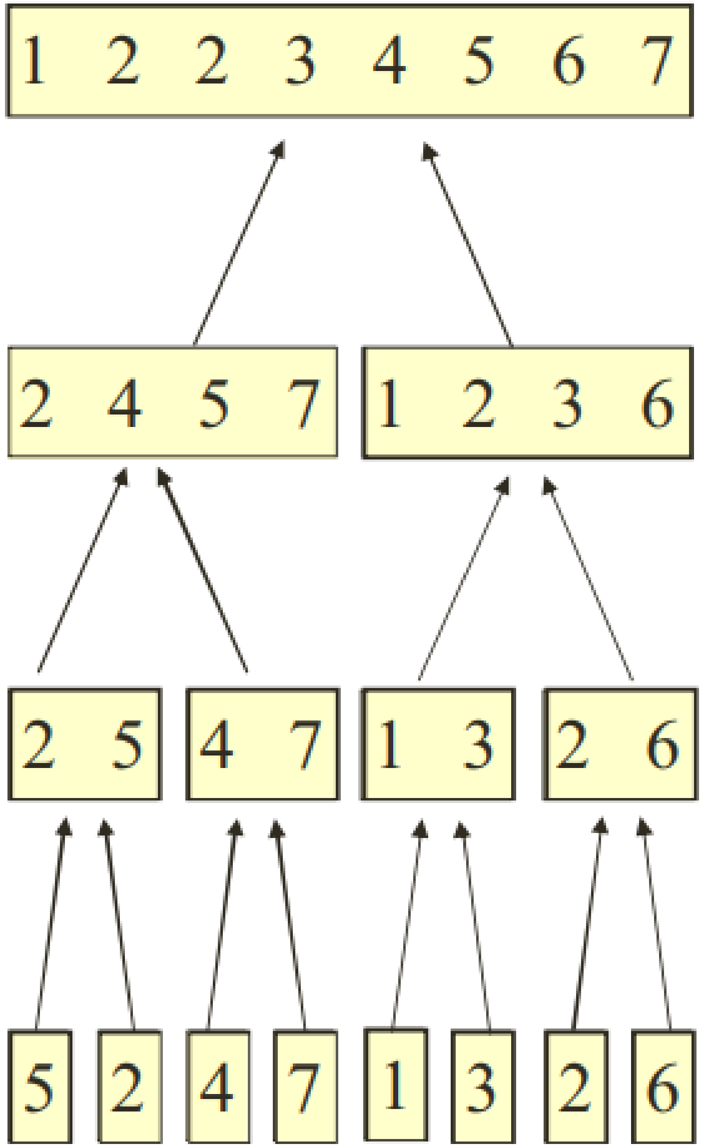
Merge Sort – Example

Then the first Mr. MergeSort succeeds and returns.

Then each of the 2 Mr. MergeSorts returns the merged numbers.

Then the 4 Mr. MergeSorts returns the merged numbers.

Then the 8 Mr. MergeSorts return.



Merge Sort – Pseudo code [1]

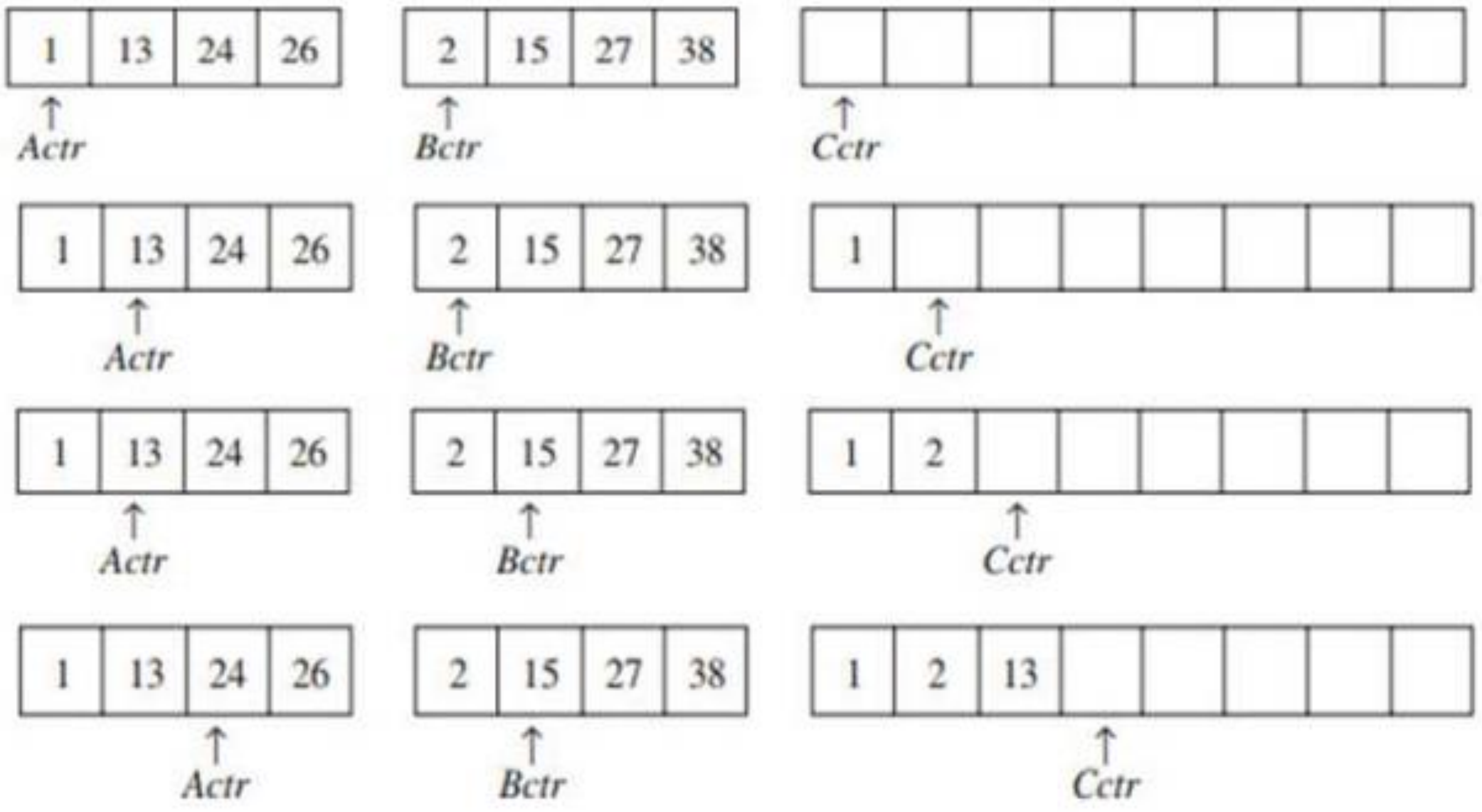
Code

```
Algorithm MergeSort(A, left, right):
Input: An array A storing integer elements.
Output: A is sorted in ascending order from left to right

    if left < right then
        mid ← (left + right) / 2
        MergeSort(A, left, mid)
        MergeSort(A, mid+1, right)
        Merge(A, left, mid, right)
```

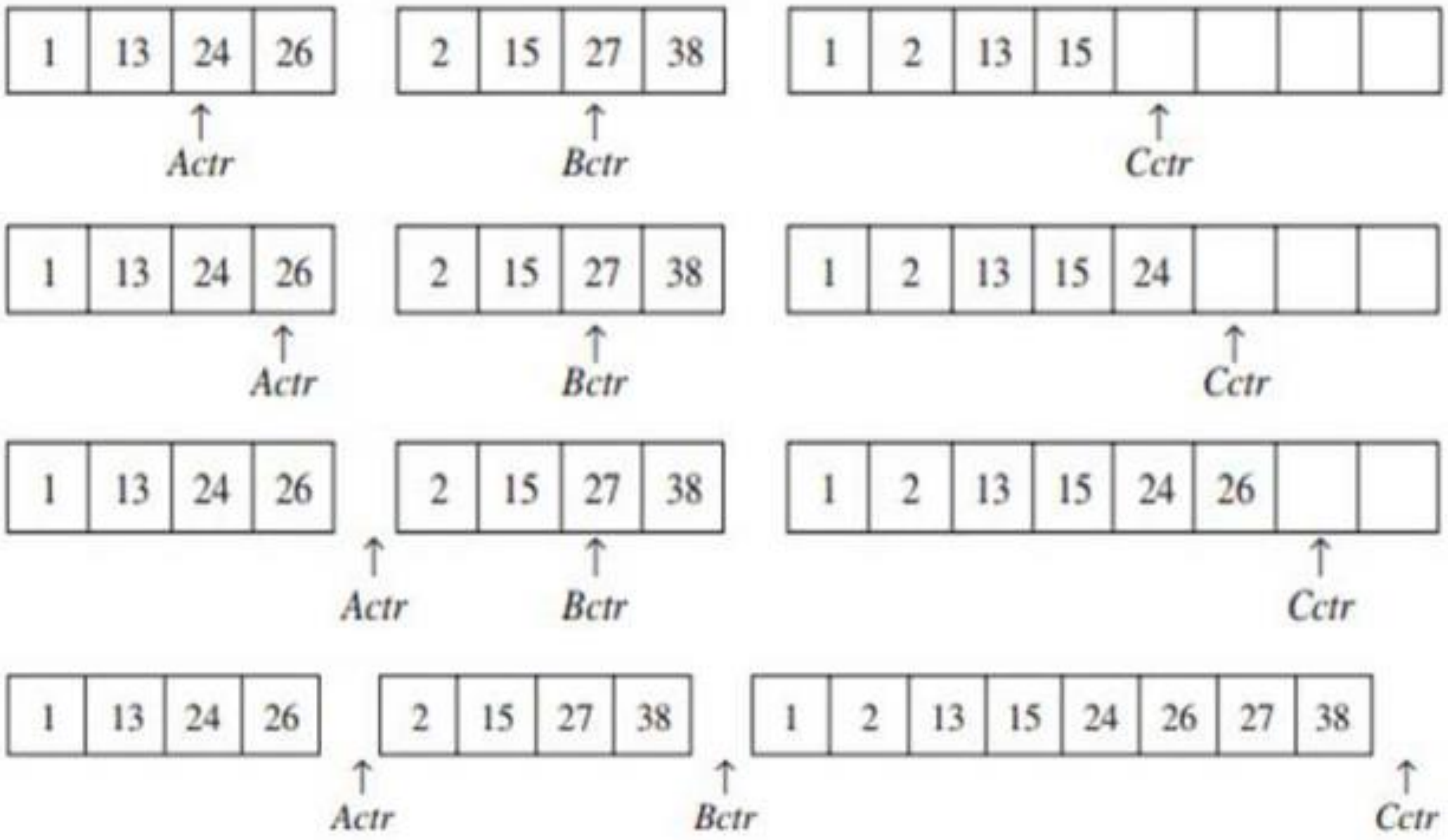
Merge Sort

- How to Merge?



Merge Sort

- How to Merge?



Merge – Pseudo code

Code

```
Algorithm Merge(A, left, mid, right):
Input: Two sorted arrays A[left…mid] and A[mid+1…right]
Output: Sorted array A[left…right]

c[0…right-left] is a new array
i ← left        j ← mid+1       k ← 0

while i<mid and j<right
   if a[i]<a[j] then
      c[k] ← a[i]   i ← i+1
   else
      c[k] ← a[j]   j ← j+1
   k ← k+1
for t ← i to mid do
  c[k] ← a[t]   k ← k+1
for t ← j to right do
  c[k] ← a[t]   k ← k+1

a[left…right] ← c[0…k-1]
```

Merge Sort – Time complexity

$$T(n) = \begin{cases} O(1) & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) & if\ n > 1 \end{cases}$$

- **Apply recursion-trees method**
  - Assume that $n=2^k$ (k>0)
  - Construct the recursion tree representing the recurrence.
  - The fully expand recursion tree has **k+1** levels
  - Each level contributes a total cost of **c.n**, where c is a constant.

Merge Sort

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

$$= 2\ [2\ T(n/4) + n/2] + n$$

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

$$= 4\ [2\ T(n/8) + n/4] + 2n$$

$$= 8\ T(n/8) + 3n$$

$$= 16\ T(n/16) + 4n$$

$= 2^k\ T(n/2^k) + k\ n$ [*this is the Eureka! line*]

$n/2^k = 1$ *OR* $n = 2^k$ *OR* $\log_2 n = k$

$$= 2^k\ T(n/2^k) + k\ n$$

$$= 2^{\log_2 n}\ T(1) + (\log_2 n)\ n$$

$= n + n \log_2 n$ [*remember that $T(1) = 1$*]

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

Sorting Algorithms

- Selection sort

Time complexity: $\mathbf{O(n^2)}$. Stability: **No**

- Insertion sort

Time complexity: $\mathbf{O(n^2)}$. Stability: **Yes**

- Bubble sort

Time complexity: $\mathbf{O(n^2)}$. Stability: **Yes**

- Merge sort

Time complexity: $\mathbf{O(}\boldsymbol{n}\mathbf{log}\boldsymbol{n}\mathbf{)}$. Stability: **Yes**

Tutorial and next topic

Preparing for the tutorial:

- Practice with examples and exercises in Tutorial 3

Preparing for next topic:

- Read textbook chapter 7 Sorting (7.5, 7.7 & 7.11)