# Algorithms for Sorting

ESC108A Elements of Computer Science and Engineering

B. Tech. 2017

**Course Leaders:**

**Roopa G.**

**Ami Rai E.**

**Chaitra S.**

## Objectives

- At the end of this lecture, student will be able to
  - apply the general strategies of sorting in a list of elements
  - identify the various requirements that demand different strategies of sorting

# Sorting

- Sorting
  - placing the data into a particular order such as ascending or descending

- Practical Examples
  – A bank sorts all checks by account number so that it can prepare individual bank statements at the end of each month
  – Telephone companies sort their lists of accounts by last name and, within that, by first name to make it easy to find phone numbers

## Sorting Algorithms

- A sorting algorithm
  - An algorithm that puts elements of a list in a certain order
- The most used orders are numerical order
- Efficient sorting is important to optimize the use of other algorithms that require sorted lists to work correctly

# List of Various Sorting Algorithms

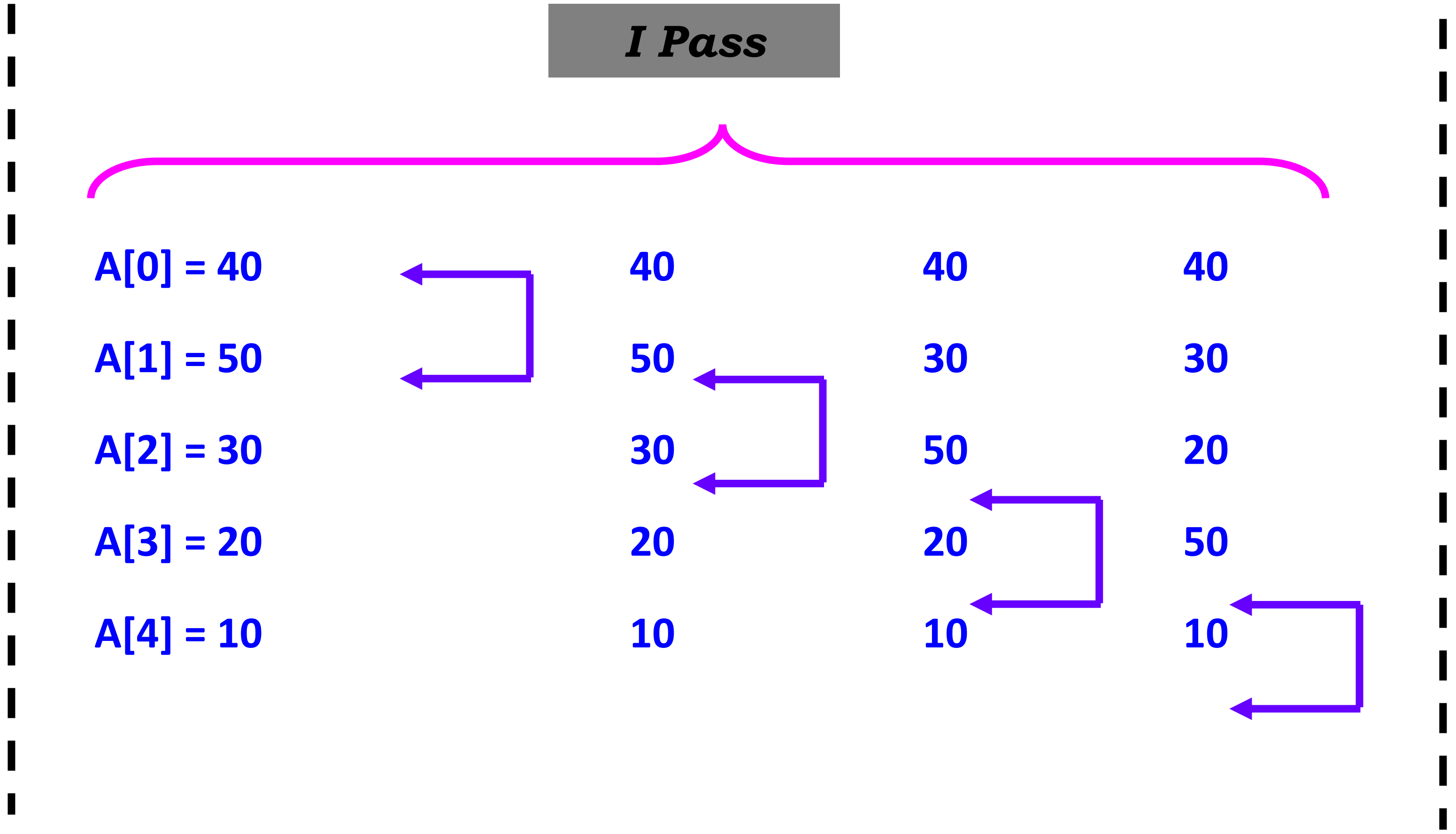
- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge sort
- Quick sort

# Bubble Sort

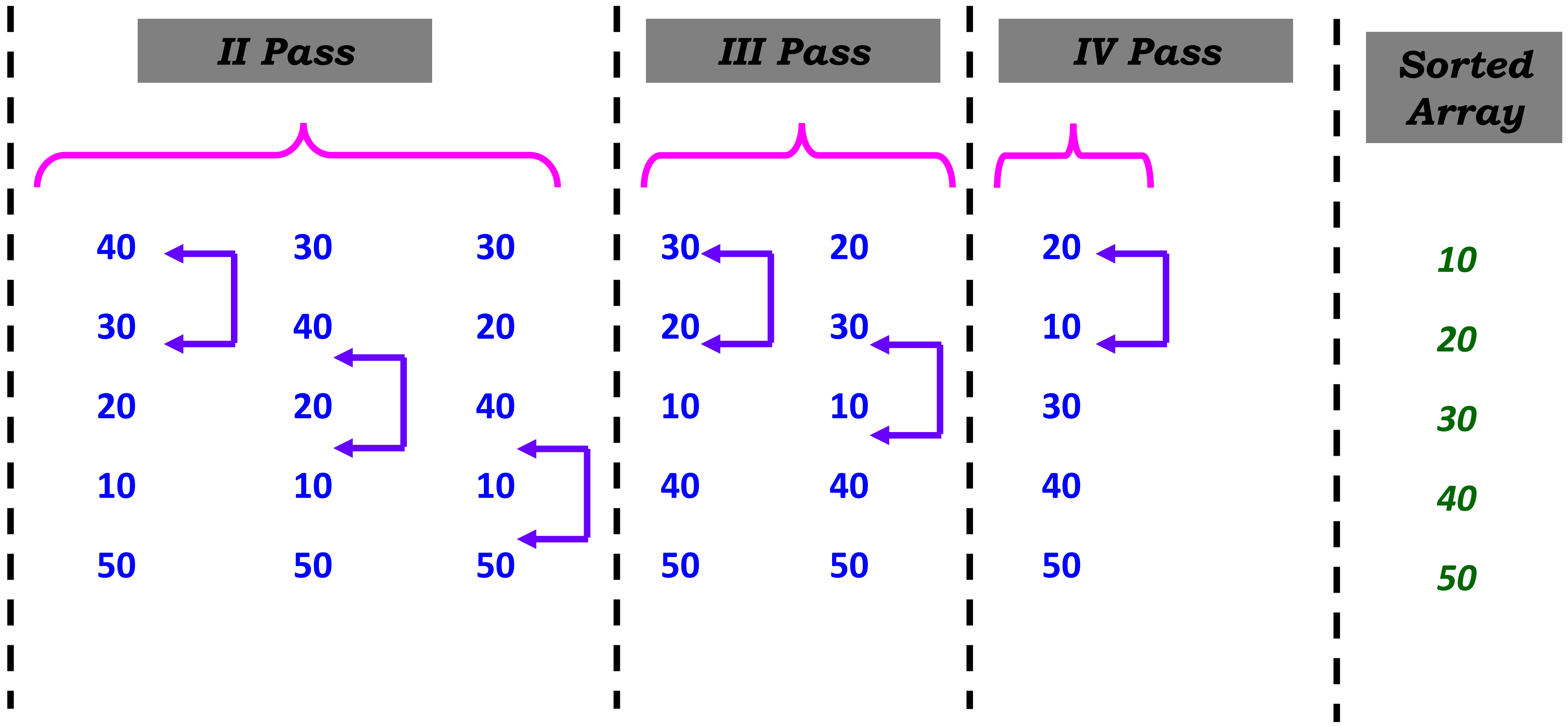
- Also known as exchange sort or sinking sort or comparison sort
  - the smaller values gradually “bubble” their way upward to the top of the array like air bubbles rising in water, while the larger values sink to the bottom of the array

- It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order
- The pass through the list is repeated until no swaps are needed, which means the list is sorted

# Trace of a Bubble Sort



# Trace of a Bubble Sort contd.



# Worst Case Performance

- Bubble sort has worst-case complexity $O(n^2)$ on lists of size $n$
  - Each element is moved no more than one step each time
  - No element can be more than a distance of $n-1$ away from its final sorted position, so we use at most $n - 1 = O(n)$ operations to move an element to its final sorted position, and use no more than $(n - 1)^2 = O(n^2)$ operations in the worst case
  - On a list where the smallest element is at the bottom, each pass through the list will only move it up by one step, so we will take $n- 1$ passes to move it to its final sorted position.
  - As each pass traverses the whole list a pass will take $n - 1 = O(n)$ operations

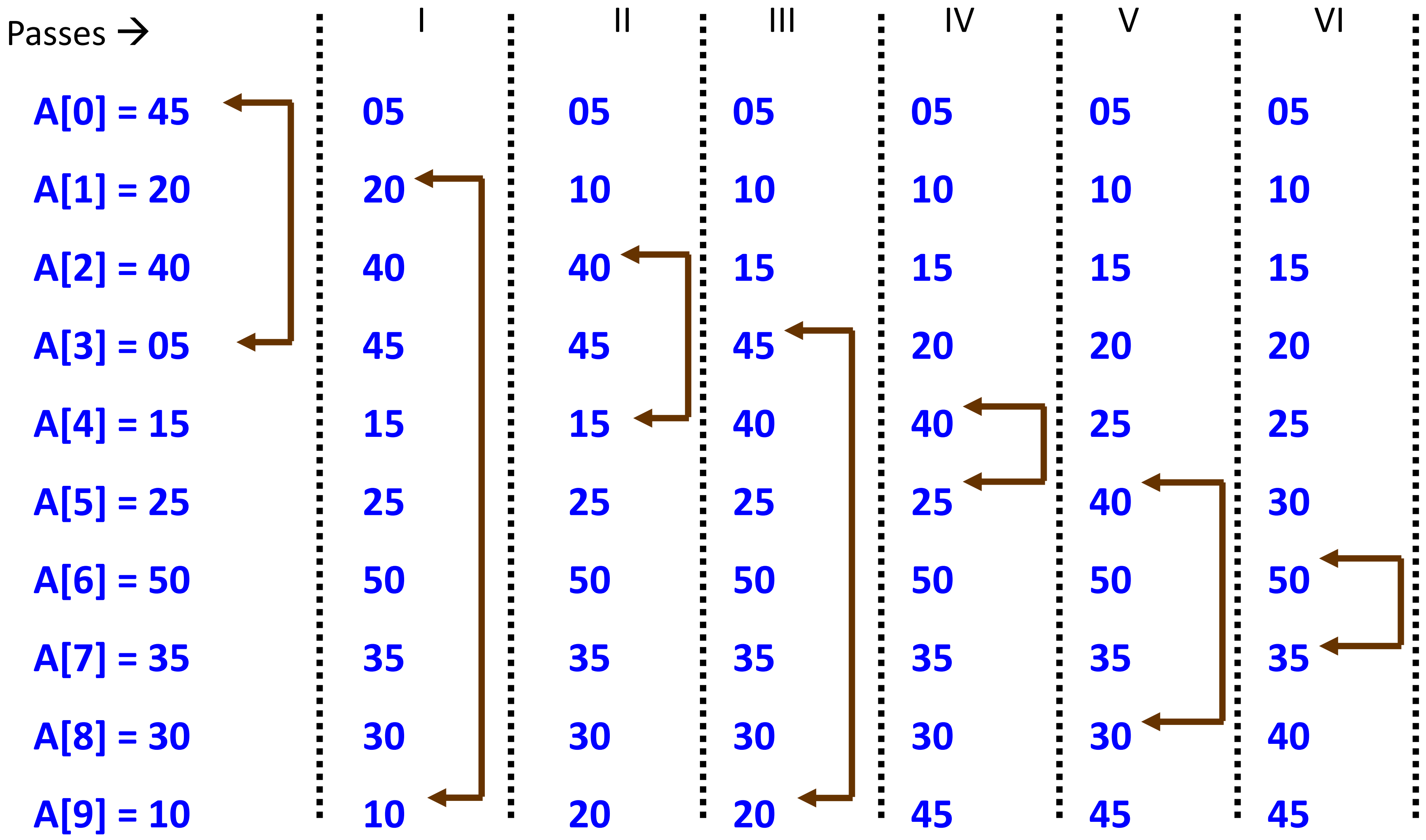
## Best Case Performance

- When a list is already sorted, bubble sort will pass through the list once, and find that it does not need to swap any elements. This means the list is already sorted
- Thus bubble sort will take O(n) time when the list is completely sorted
- It will also use considerably less time if the elements in the list are not too far from their sorted places

# Selection Sort

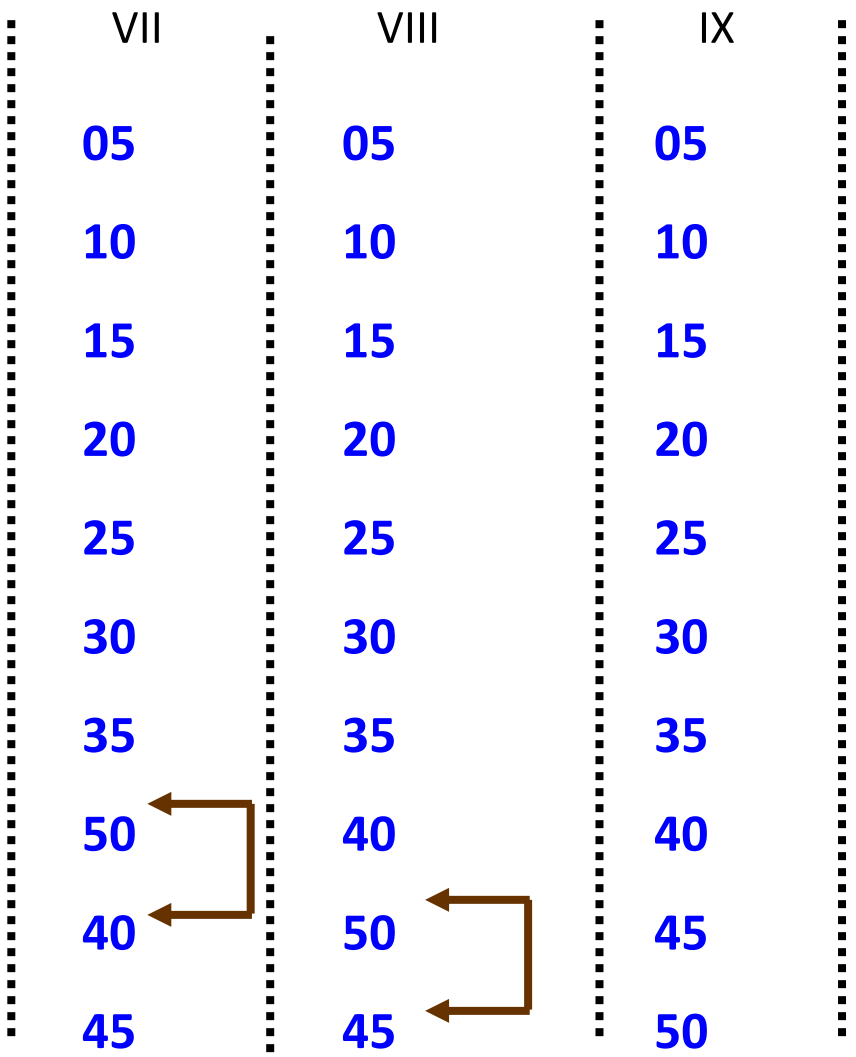
- A sorting algorithm, specifically an in-place comparison sort
- Noted for its simplicity
- Has performance advantages over more complicated algorithms in certain situations
- It works as follows:
  1. Find the minimum value in the list
  2. Swap it with the value in the first position
  3. Repeat the steps above for remainder of the list (starting at the second position)

# Trace of a Selection Sort



| Passes → | I | II | III | IV | V | VI |
|---|---|---|---|---|---|---|
| A[0] = 45 | 05 | 05 | 05 | 05 | 05 | 05 |
| A[1] = 20 | 20 | 10 | 10 | 10 | 10 | 10 |
| A[2] = 40 | 40 | 40 | 15 | 15 | 15 | 15 |
| A[3] = 05 | 45 | 45 | 45 | 20 | 20 | 20 |
| A[4] = 15 | 15 | 15 | 40 | 40 | 25 | 25 |
| A[5] = 25 | 25 | 25 | 25 | 25 | 40 | 30 |
| A[6] = 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| A[7] = 35 | 35 | 35 | 35 | 35 | 35 | 35 |
| A[8] = 30 | 30 | 30 | 30 | 30 | 30 | 40 |
| A[9] = 10 | 10 | 20 | 20 | 45 | 45 | 45 |

# Trace of a Selection Sort contd.



| VII | VIII | IX | ***Sorted Array*** |
|---|---|---|---|
| 05 | 05 | 05 | *05* |
| 10 | 10 | 10 | *10* |
| 15 | 15 | 15 | *15* |
| 20 | 20 | 20 | *20* |
| 25 | 25 | 25 | *25* |
| 30 | 30 | 30 | *30* |
| 35 | 35 | 35 | *35* |
| 50 | 40 | 40 | *40* |
| 40 | 50 | 45 | *45* |
| 45 | 45 | 50 | *50* |

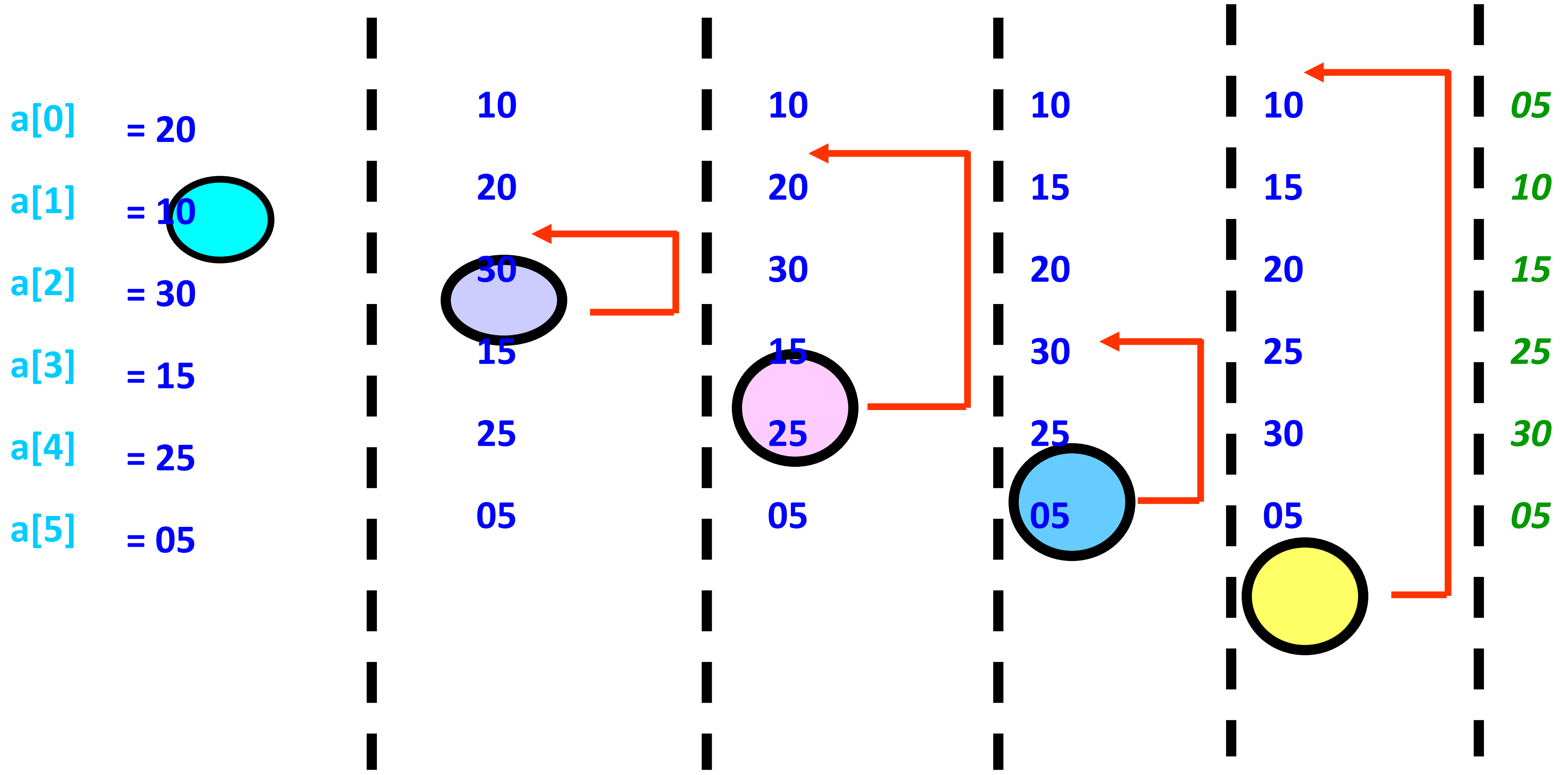
## Analysis

- Easy to analyze since none of the loops depend on the data in the array
- Selecting the lowest element requires scanning all *n* elements (takes *n* - 1 comparisons) and then swapping it into the first position
- Finding the next lowest element requires scanning the remaining *n*-1 elements and so on, for a total of (*n* - 1) + (*n* - 2) + ... + 2 + 1 = O($n^2$) comparisons
- Each of these scans requires one swap for a total of *n* - 1 swaps (the final element is already in place)
- Thus, the comparisons dominate the running time, which is O($n^2$)

# Insertion Sort

- A simple sorting algorithm, a comparison sort in which the sorted array (or list) is built one entry at a time
- Efficient on small data sets
- Efficient on data sets which are already substantially sorted
- Stable (does not change the relative order of elements with equal keys)
- It is an online algorithm, in that it can sort a list as it receives it

# Trace: Insertion Sort



## Best Case: Insertion Sort

- In a sorted array, the implementation of insertion sort takes O(*n*) time
- In each iteration, the first remaining element of the input is only compared with the last element of the sorted subsection of the array
- Thus, if an array is sorted or nearly sorted, insertion sort will significantly outperform quick sort

## Worst Case: Insertion Sort

- The worst case is an array sorted in reverse order, as every execution of the inner loop will have to scan and shift the entire sorted section of the array before inserting the next element
- Insertion sort takes O($n^2$) time in the worst case as well as in the average case, which makes it impractical for sorting large numbers of elements
- Insertion sort's inner loop is very fast, which often makes it one of the fastest algorithms for sorting small numbers of elements, typically less than 10 or so

## Time Complexities

| Algorithm | Best | Average | Worst |
|---|---|---|---|
| Bubble Sort | O(n) | -- | O(n2) |
| Selection Sort | O(n2) | O(n2) | O(n2) |
| Insertion Sort | O(n) | O(n + d) | O(n2) |

## Summary

- A sorting algorithm is an algorithm that puts elements of a list in a certain order
- Bubble sort is a sorting algorithm, comparing two items at a time and swapping them if they are in the wrong order
- Selection sort is a comparison sort in which the minimum element is exchanged with the first position repeatedly
- Insertion sort is a comparison sort in which the sorted array (or list) is built one entry at a time