Sorting an Array – Simple Algorithms

Lecture 33
Sections 9.3, 9.4

Robb T. Koether

Hampden-Sydney College

Wed, Nov 28, 2018

Outline

Sorting an Array

- The Problem
  - Arrange the elements of an array into ascending (or descending) order.
- The Algorithm
  - There are many different sorting algorithms.
  - Some are efficient.
  - Some are not efficient.

Efficiency of Sorting Algorithms

- Simple Sorting Algorithms
  - They run in $O(n^2)$ time.
  - They are relatively inefficient.
  - They are suitable for short lists only (at most, thousands of elements).

Efficiency of Sorting Algorithms

- Complex Sorting Algorithms
  - They run in $O(n \log n)$ time.
  - They are very efficient.
  - They are suitable for long lists (up to hundreds of millions of elements).

Sorting Algorithms

- We will study three simple sorting algorithms.
  - Bubble sort
  - Selection sort
  - Insertion sort
- And we will study one complex sorting algorithm.
  - The Merge sort

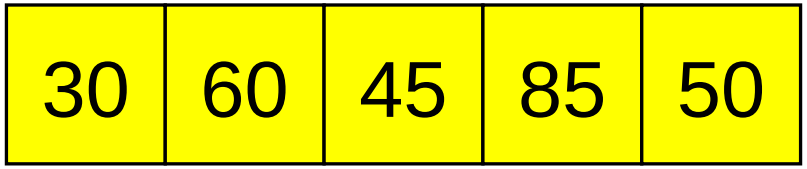
Outline

Outline

The Bubble Sort Algorithm

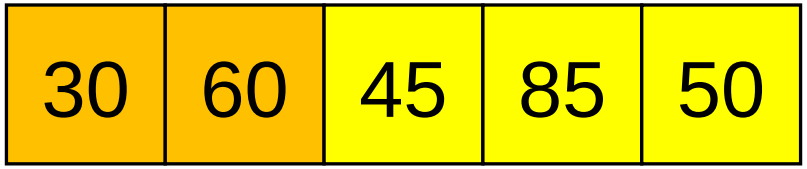
- Make $n - 1$ passes.
- On each pass, compare each element to its successor.
- If they are out of order, then swap them.

The Bubble Sort Algorithm



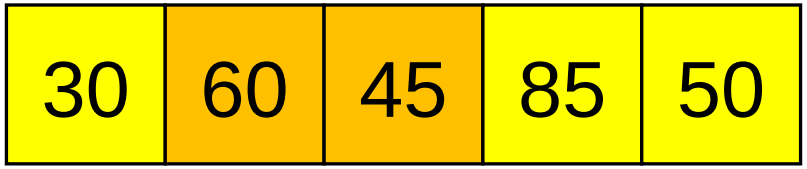
Begin with the array

The Bubble Sort Algorithm



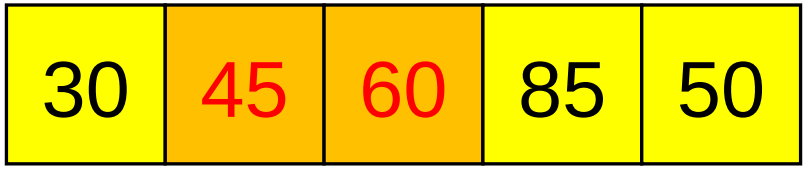
Compare `a[0]` to `a[1]`

The Bubble Sort Algorithm



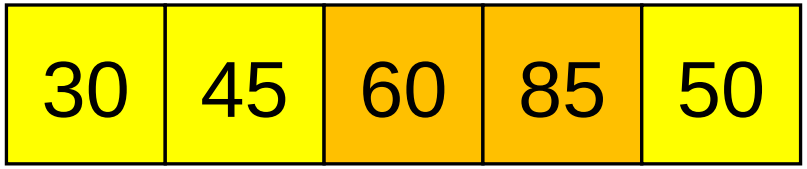
Compare `a[1]` to `a[2]`

The Bubble Sort Algorithm



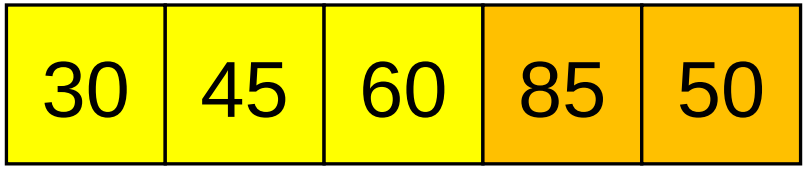
Swap `a[1]` and `a[2]`

The Bubble Sort Algorithm



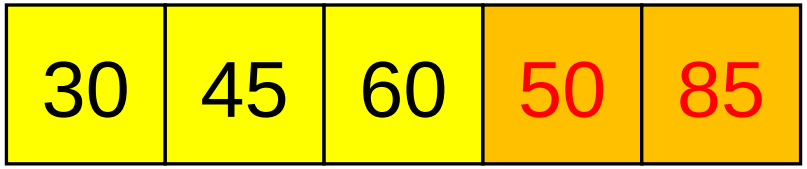
Compare `a[2]` to `a[3]`

The Bubble Sort Algorithm



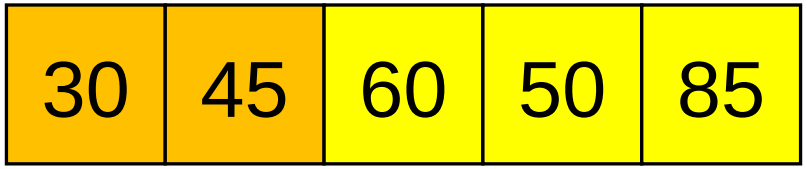
Compare `a[3]` to `a[4]`

The Bubble Sort Algorithm



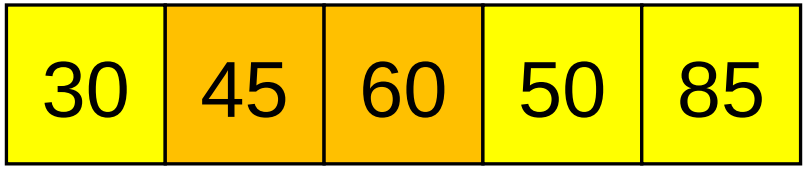
Swap `a[3]` and `a[4]`

The Bubble Sort Algorithm



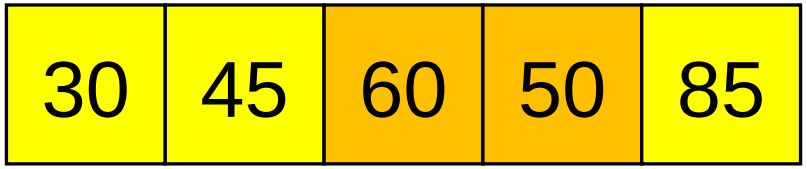
Compare `a[0]` to `a[1]`

The Bubble Sort Algorithm



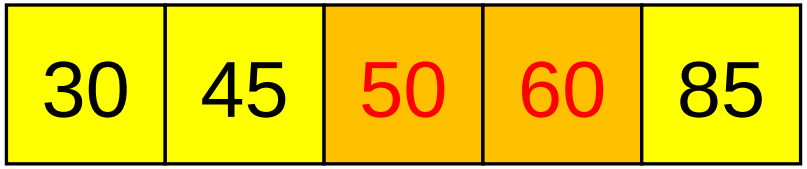
Compare `a[1]` to `a[2]`

The Bubble Sort Algorithm



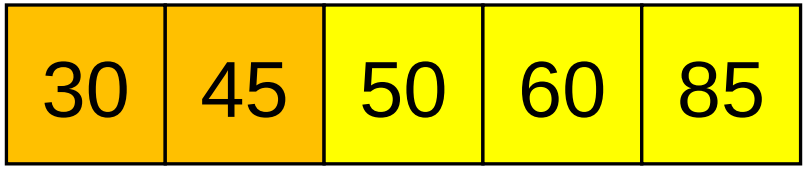
Compare `a[2]` to `a[3]`

The Bubble Sort Algorithm



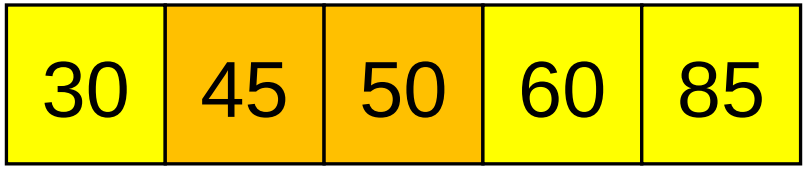
Swap `a[2]` and `a[3]`

The Bubble Sort Algorithm



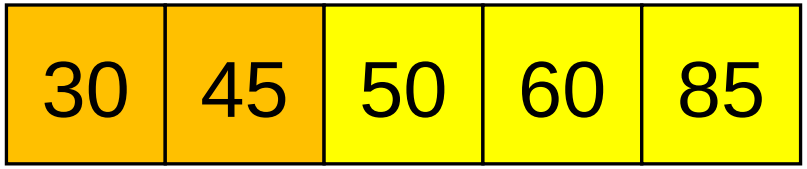
Compare `a[0]` to `a[1]`

The Bubble Sort Algorithm



Compare `a[1]` to `a[2]`

The Bubble Sort Algorithm



Compare `a[0]` to `a[1]`

Outline

Efficiency of the Bubble Sort

- The average case requires $\frac{1}{2}(n^2 - n)$ comparisons.
- The best case requires $\frac{1}{2}(n^2 - n)$ comparisons.
- The worst case requires $\frac{1}{2}(n^2 - n)$ comparisons.
- The average case requires approximately $\frac{1}{4}(n^2 - n)$ swaps, or $\frac{3}{4}(n^2 - n)$ transfers (3 transfers per swap).

Examples of a Bubble Sort

- Examples
  - `BubbleSort.cpp`
  - `BubbleSortTimer.cpp`
  - `date.cpp`
  - `rational.cpp`
  - `point.cpp`

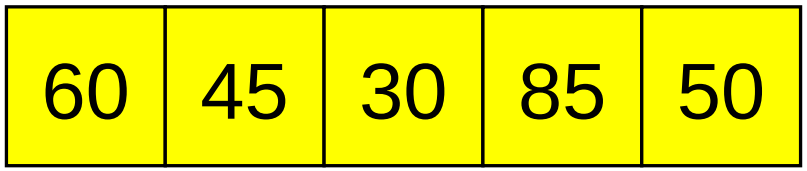
Outline

Outline

The Selection Sort Algorithm

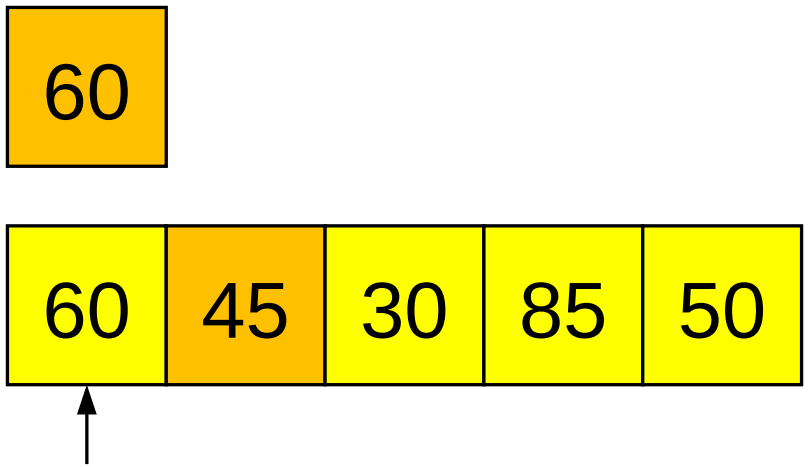
- Make $n - 1$ passes (numbered 0 through $n - 2$).
- On pass k, locate the smallest element among positions k through $n - 1$.
- At the end of pass k, swap the smallest element with the element in position k.

The Selection Sort Algorithm



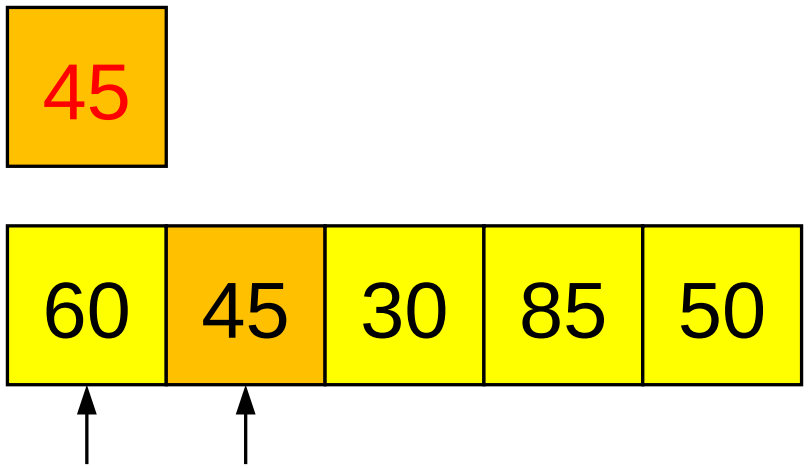
Begin with the array

The Selection Sort Algorithm



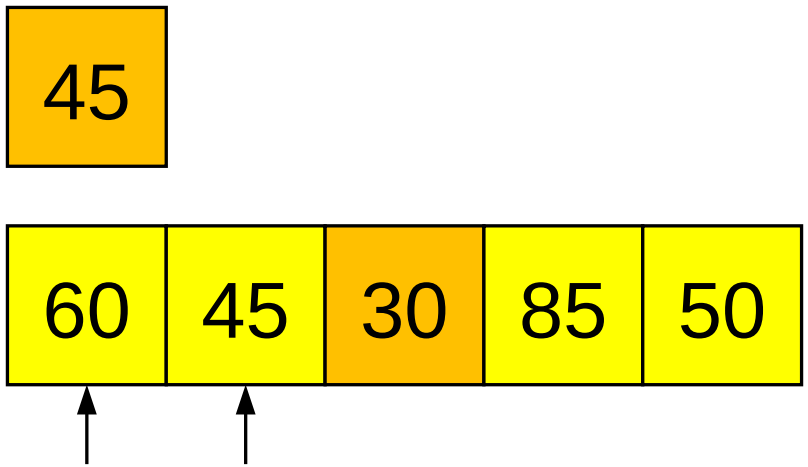
Move `a[0]` to `temp` and compare to `a[1]`

The Selection Sort Algorithm



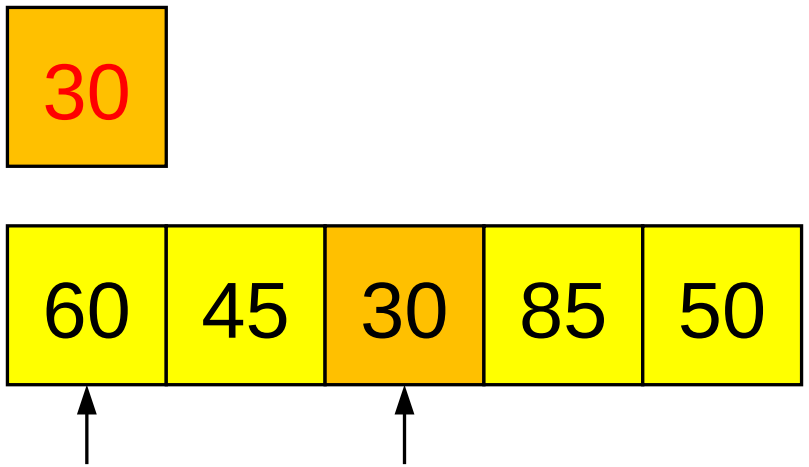
Move `a[1]` to `temp`

The Selection Sort Algorithm



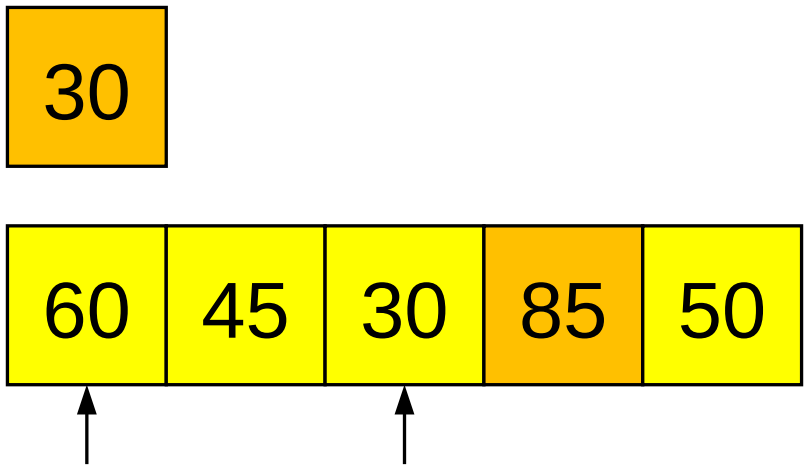
Compare `a[2]` to `temp`

The Selection Sort Algorithm



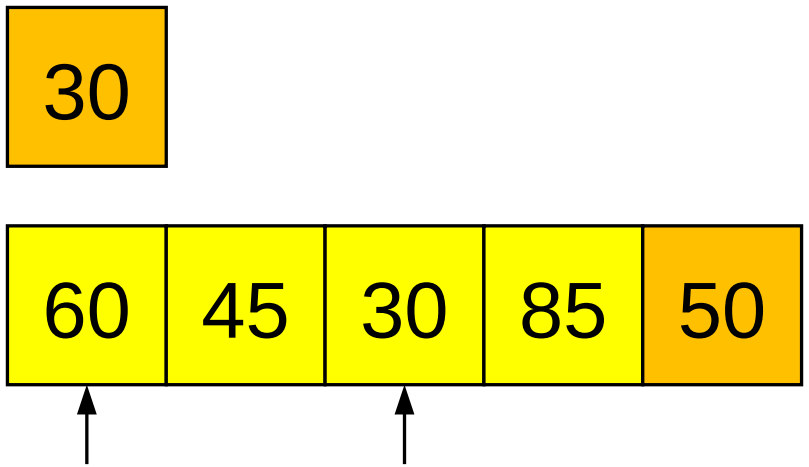
Move `a[2]` to `temp`

The Selection Sort Algorithm



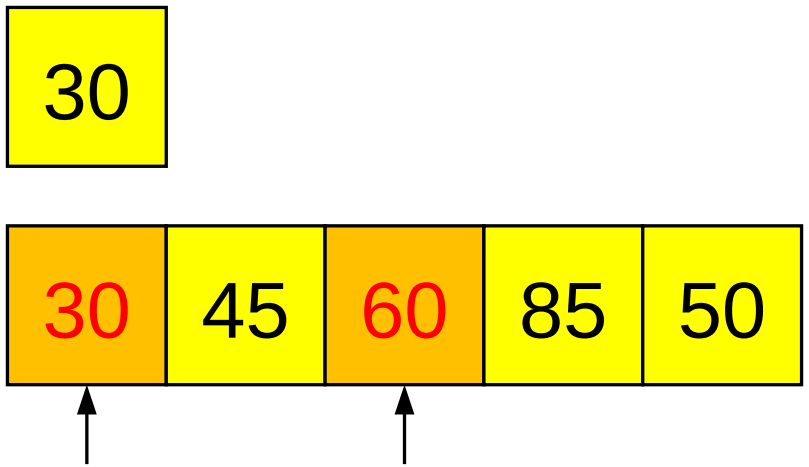
Compare `a[3]` to `temp`

The Selection Sort Algorithm



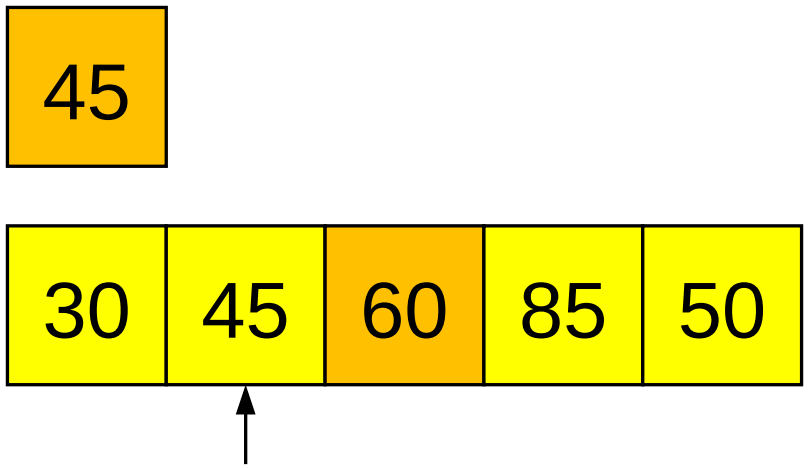
Compare `a[4]` to `temp`

The Selection Sort Algorithm



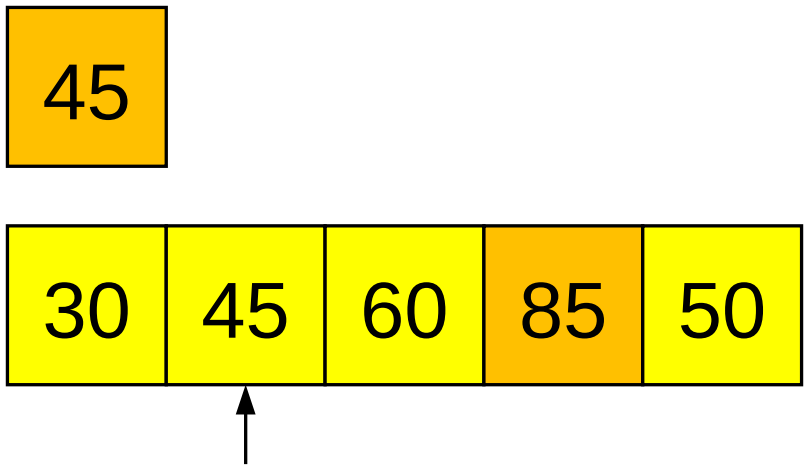
Swap `a[0]` and `a[2]`

The Selection Sort Algorithm



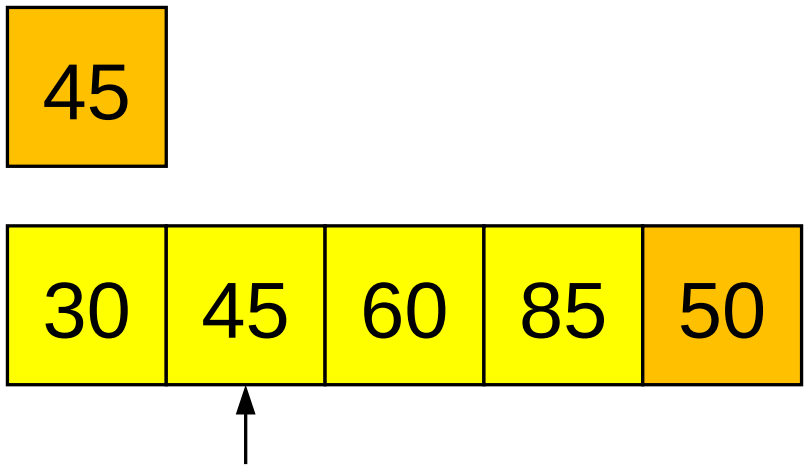
Move `a[1]` to `temp` and compare to `a[2]`

The Selection Sort Algorithm



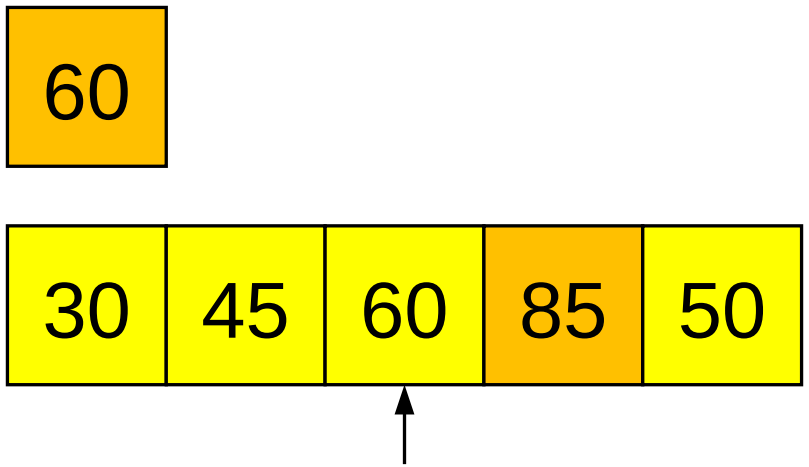
Compare `temp` to `a[3]`

The Selection Sort Algorithm



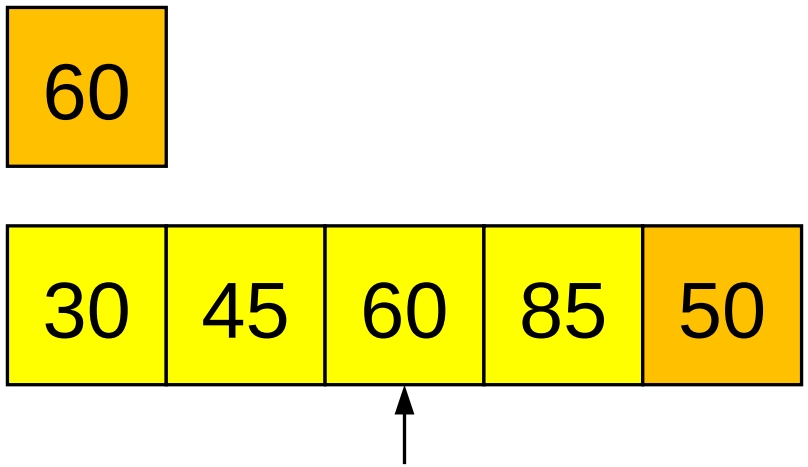
Compare `temp` to `a[4]`

The Selection Sort Algorithm



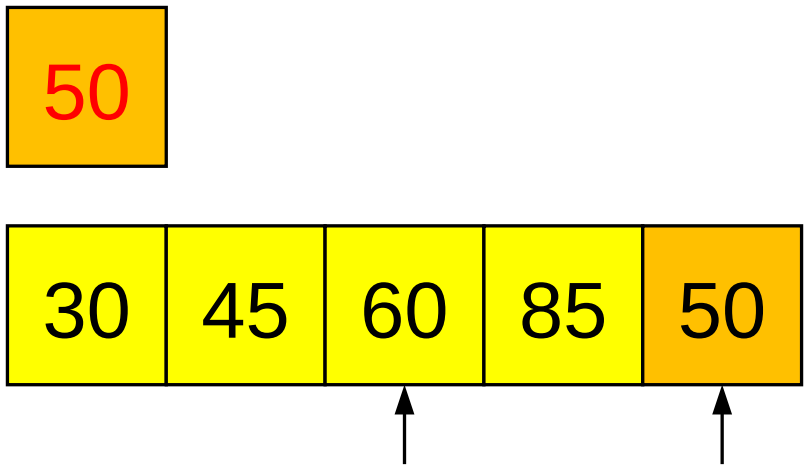
Move `a[2]` to `temp` and compare to `a[3]`

The Selection Sort Algorithm



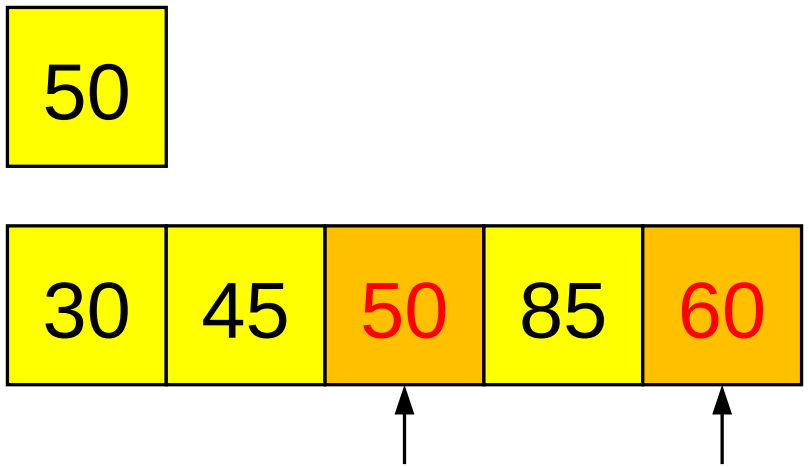
Compare `temp` to `a[4]`

The Selection Sort Algorithm



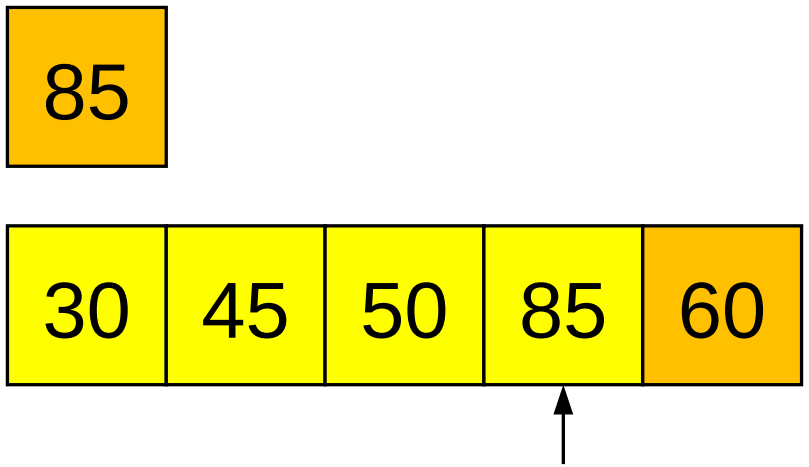
Move `a[4]` to `temp`

The Selection Sort Algorithm



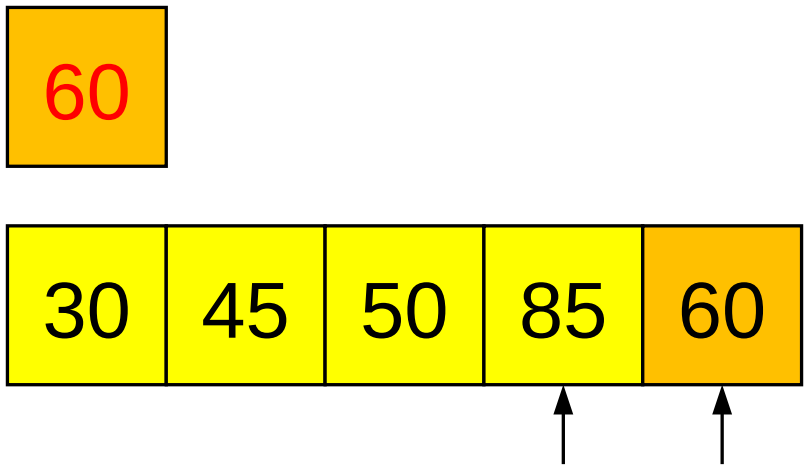
Swap `a[2]` and `a[4]`

The Selection Sort Algorithm



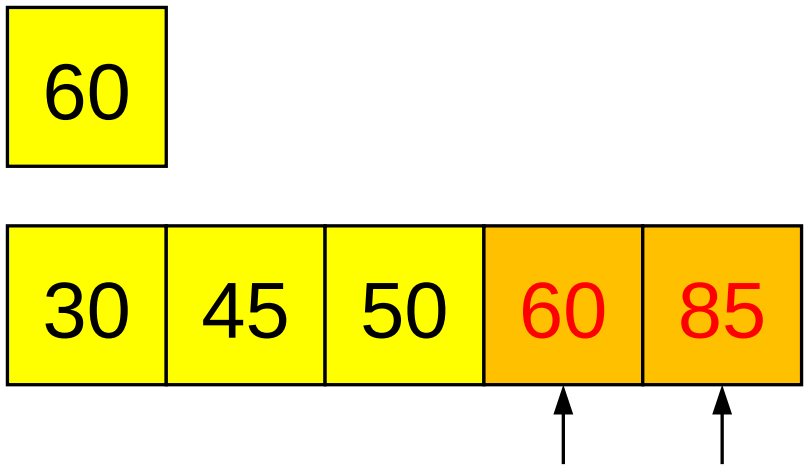
Move `a[3]` to `temp` and compare to `a[4]`

The Selection Sort Algorithm



Move `a[4]` to `temp`

The Selection Sort Algorithm



Swap `a[3]` and `a[4]`

Outline

Efficiency of the Selection Sort

- The average case requires $\frac{1}{2}(n^2 - n)$ comparisons.
- The best case requires $\frac{1}{2}(n^2 - n)$ comparisons.
- The worst case requires $\frac{1}{2}(n^2 - n)$ comparisons.
- The average case requires $3(n - 1)$ transfers.

Examples of a Selection Sort

- Examples
  - `SelectionSort.cpp`
  - `SelectionSortTimer.cpp`

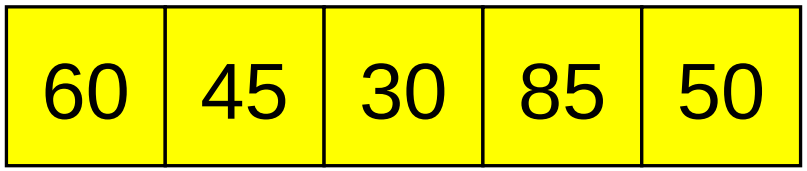
Outline

Outline

The Insertion Sort Algorithm

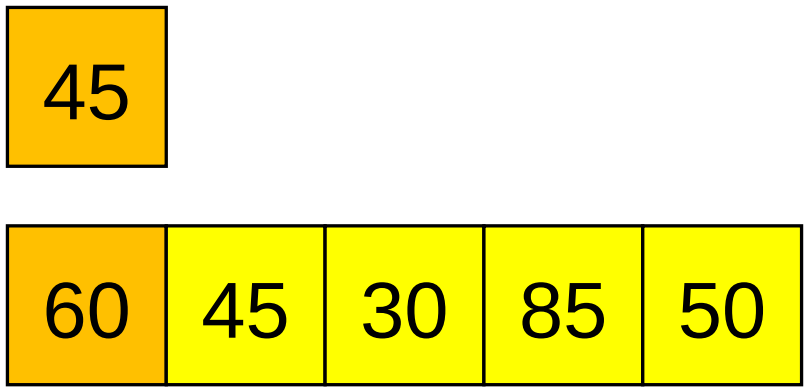
- Make $n - 1$ passes (numbered 1 through $n - 1$).
- On pass k, insert the element in position k into the list consisting of the elements in positions 0 through $k - 1$.
  - Copy element in position k element to `temp`.
  - Compare to element in position $k - 1$ to `temp`.
  - If element is greater than `temp`, shift element up one position.
  - Continue in the manner through elements in positions $k - 2$ to 0.
  - If all elements are greater than `temp`, then copy `temp` to position 0.

The Insertion Sort Algorithm



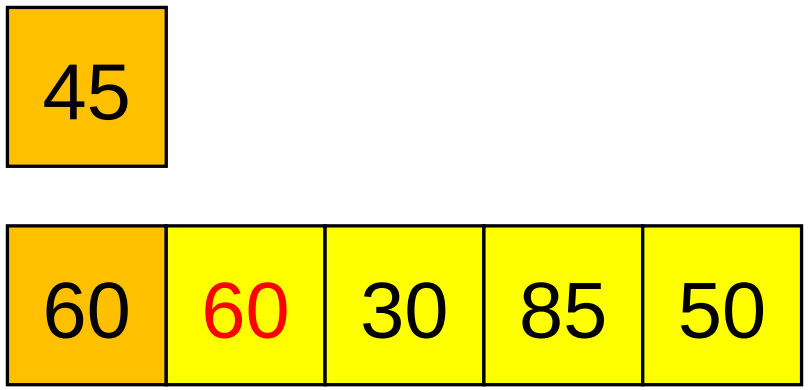
Begin with the array

The Insertion Sort Algorithm



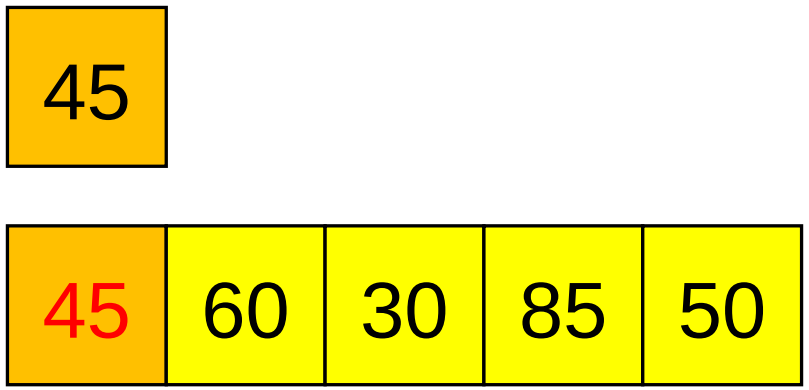
Move `a[1]` to `temp` and compare to `a[0]`

The Insertion Sort Algorithm



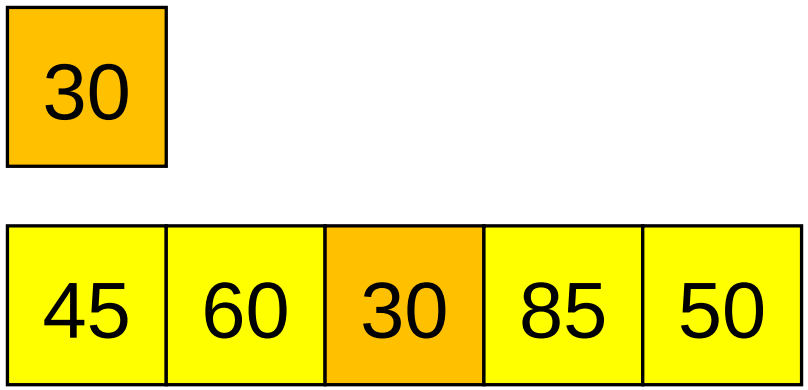
Copy `a[0]` to `a[1]`

The Insertion Sort Algorithm



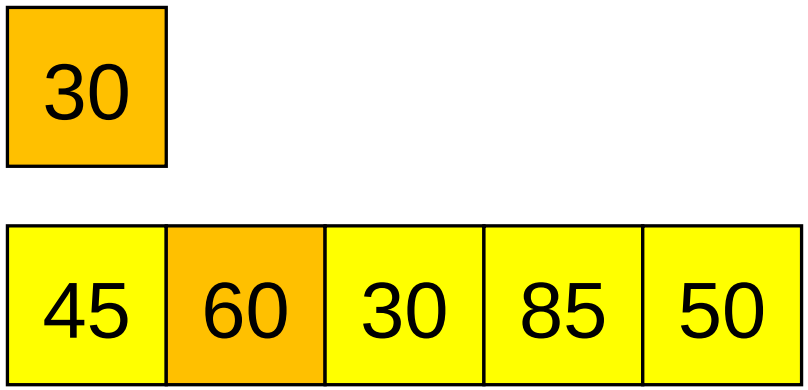
Move `temp` to `a[0]`

The Insertion Sort Algorithm



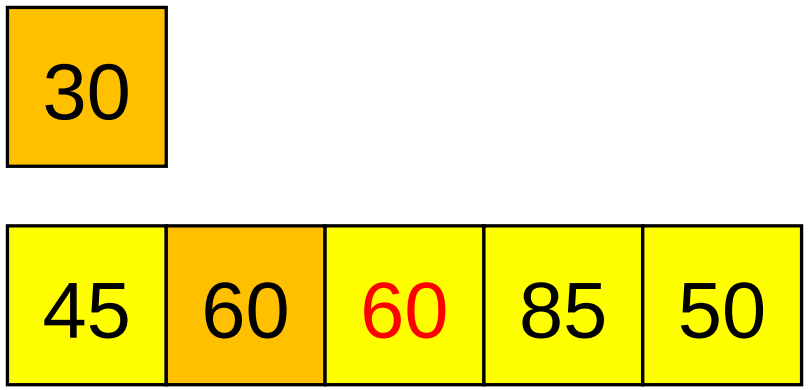
Move `a[2]` to `temp`

The Insertion Sort Algorithm



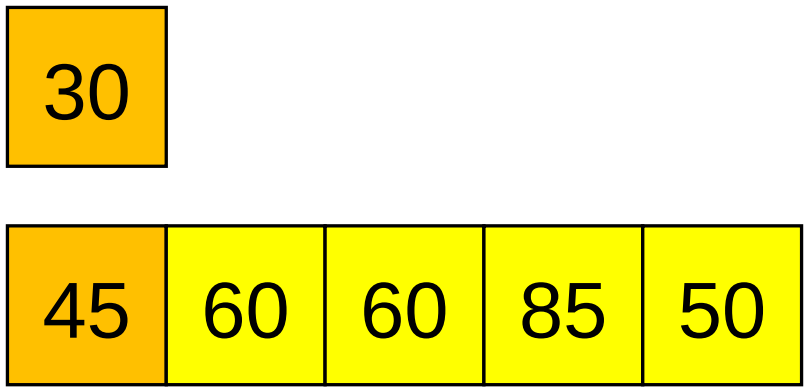
Compare `temp` to `a[1]`

The Insertion Sort Algorithm



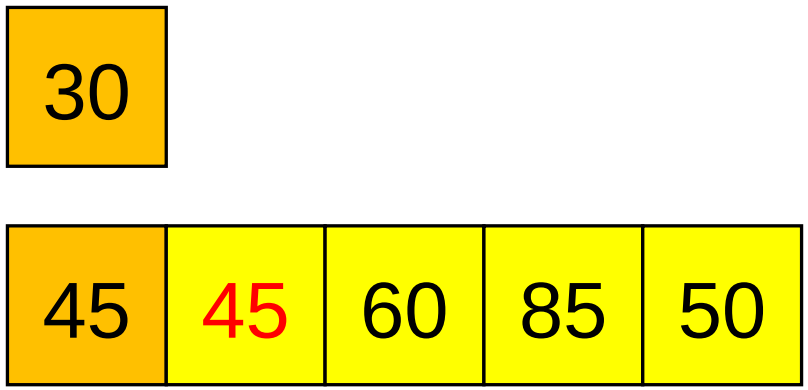
Copy `a[1]` to `a[2]`

The Insertion Sort Algorithm



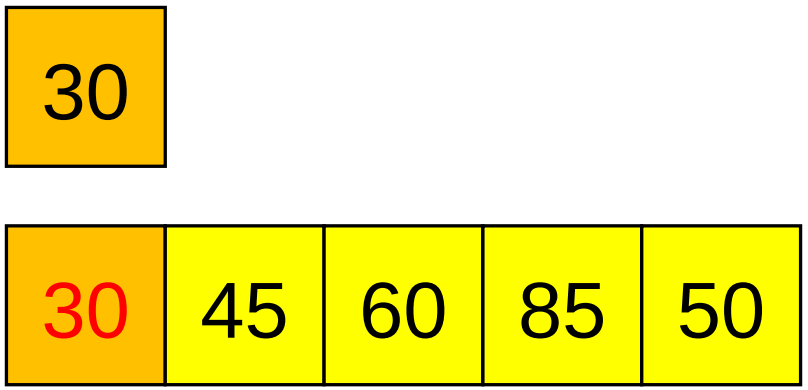
Compare `temp` to `a[0]`

The Insertion Sort Algorithm



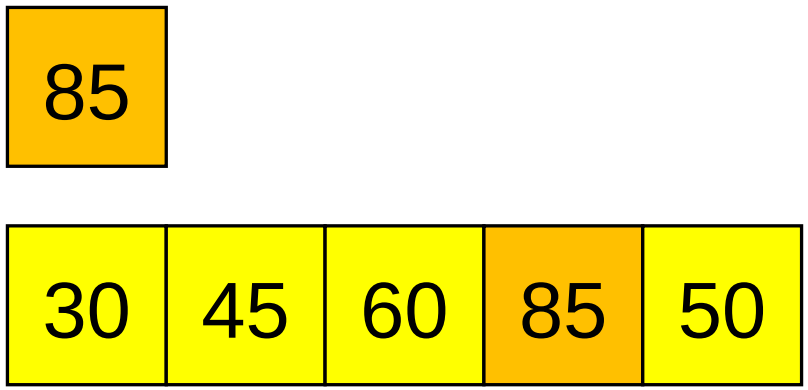
Copy `a[0]` to `a[1]`

The Insertion Sort Algorithm



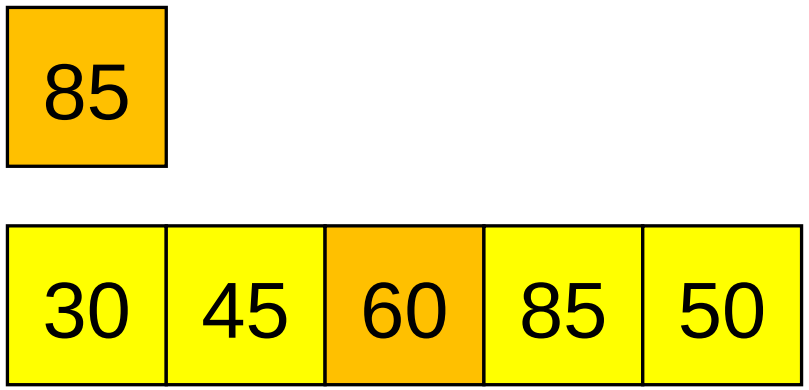
Move `temp` to `a[0]`

The Insertion Sort Algorithm



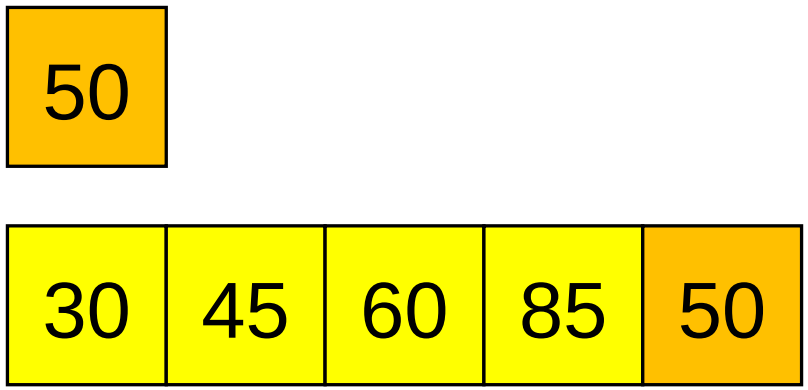
Move `a[3]` to `temp`

The Insertion Sort Algorithm



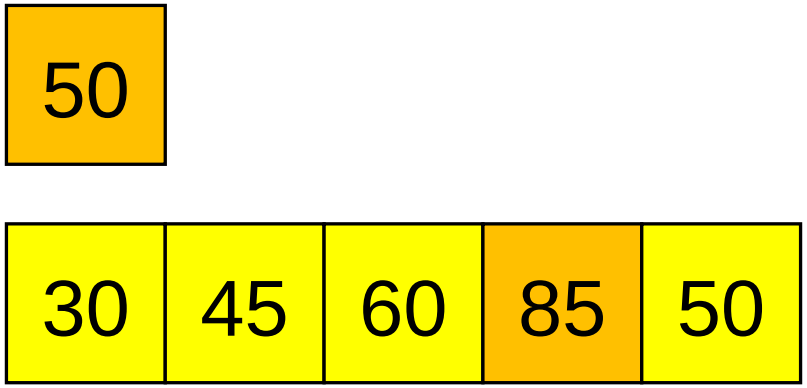
Compare `temp` to `a[2]`

The Insertion Sort Algorithm



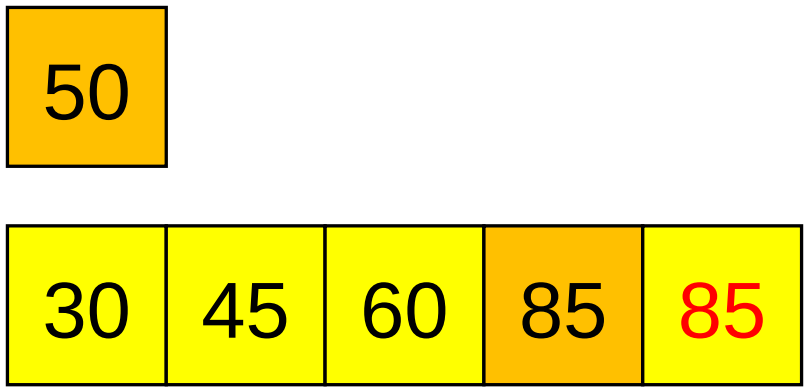
Move `a[4]` to `temp`

The Insertion Sort Algorithm



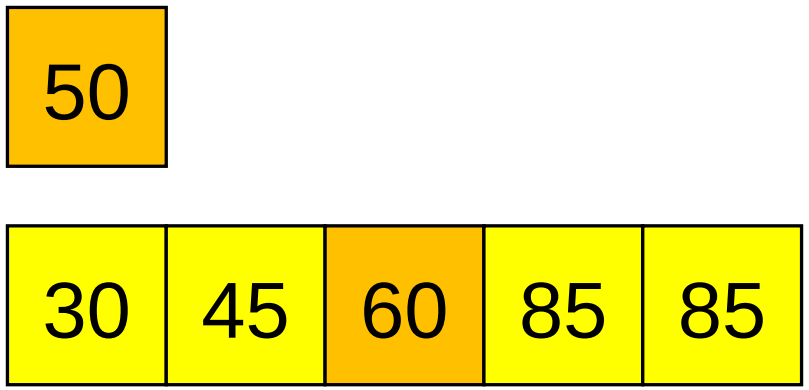
Compare `temp` to `a[3]`

The Insertion Sort Algorithm



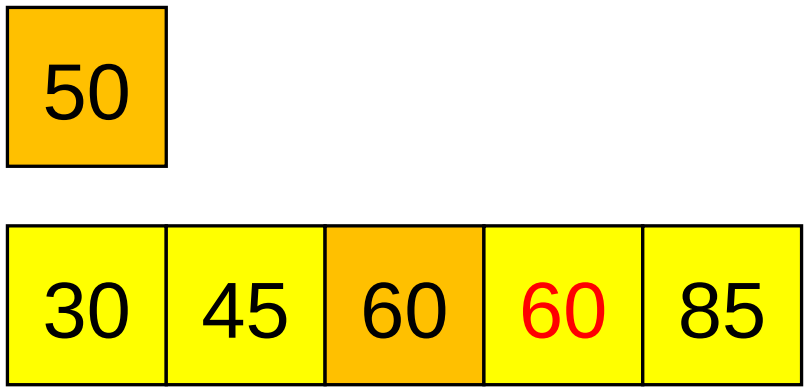
Copy `a[3]` to `a[4]`

The Insertion Sort Algorithm



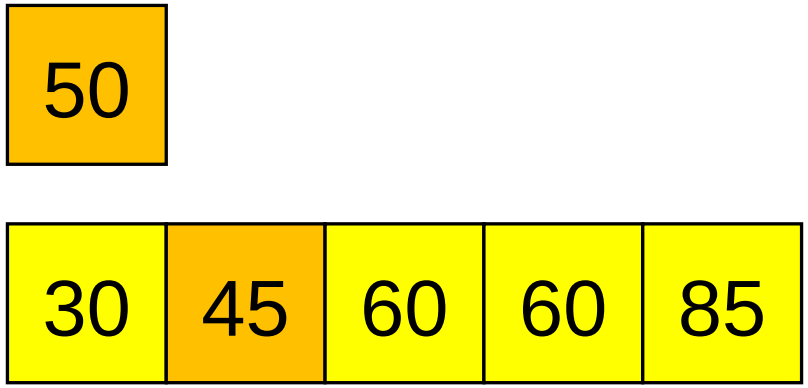
Compare `temp` to `a[2]`

The Insertion Sort Algorithm



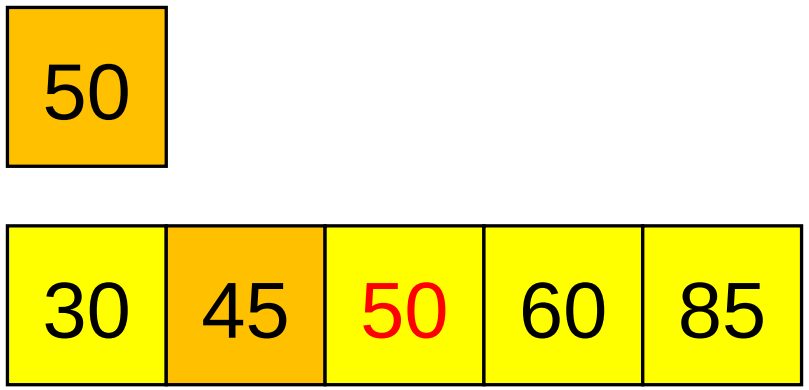
Move `a[2]` to `a[3]`

The Insertion Sort Algorithm



Compare `temp` to `a[1]`

The Insertion Sort Algorithm



Move `temp` to `a[2]`

Outline

Efficiency of the Insertion Sort

- Average case requires $\frac{1}{4}(n^2 + 3n - 4)$ comparisons.
- Best case requires $n - 1$ comparisons.
- Worst case requires $\frac{1}{2}(n^2 + n)$ comparisons.
- Average case requires $\frac{1}{4}(n^2 + 7n - 8)$ transfers.

Examples of an Insertion Sort

- Examples
  - `InsertionSort.cpp`
  - `InsertionSortTimer.cpp`

Outline

Assignment

Assignment

- Read Sections 9.3, 9.4.