Computer Programming (b) - E1124

(Spring 2021-2022)

Lecture 4

Sorting Algorithms

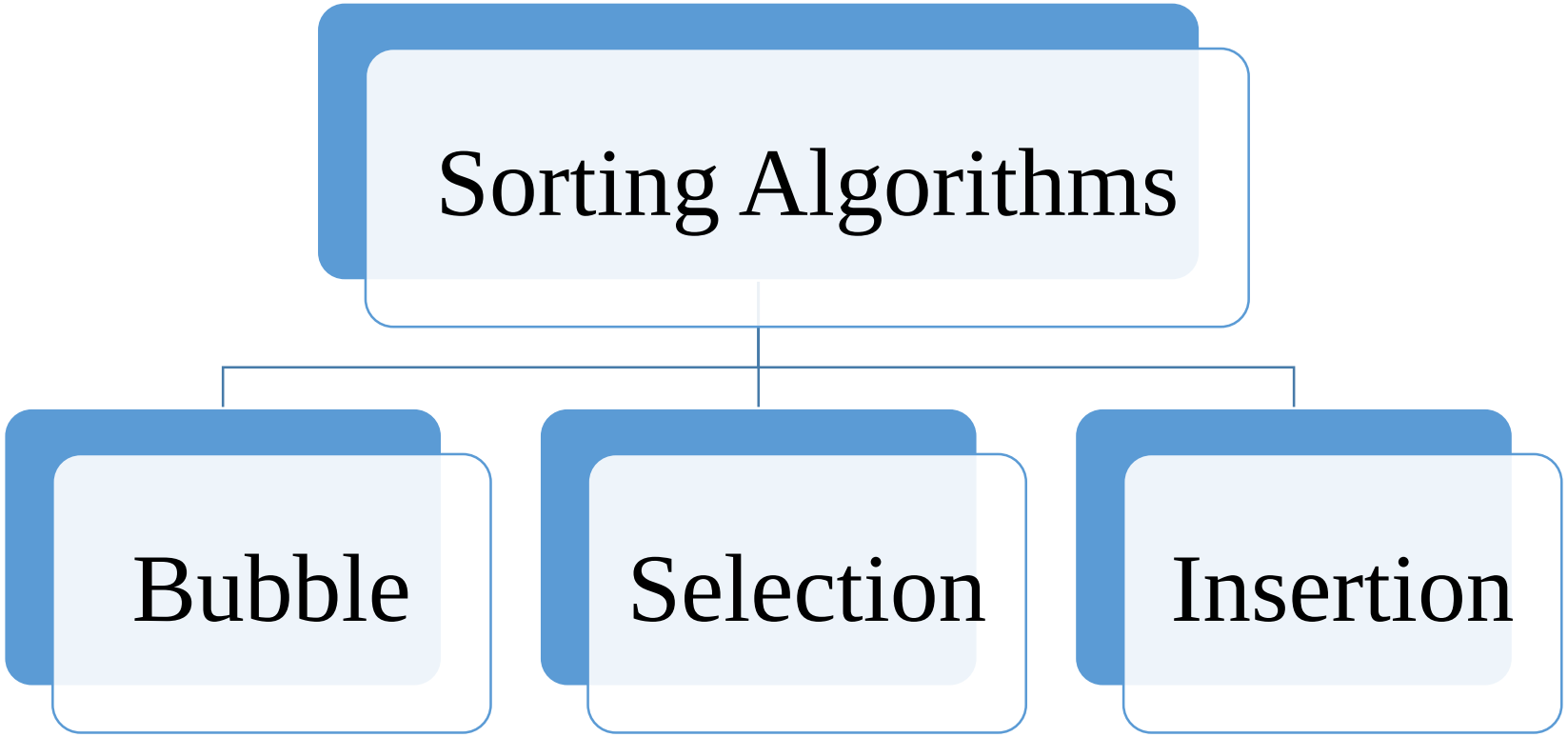
INSTRUCTOR

Dr / Ayman Soliman

Contents

- Sorting a List: Bubble Sort
- Sorting a List: Selection Sort
- Sorting a List: Insertion Sort

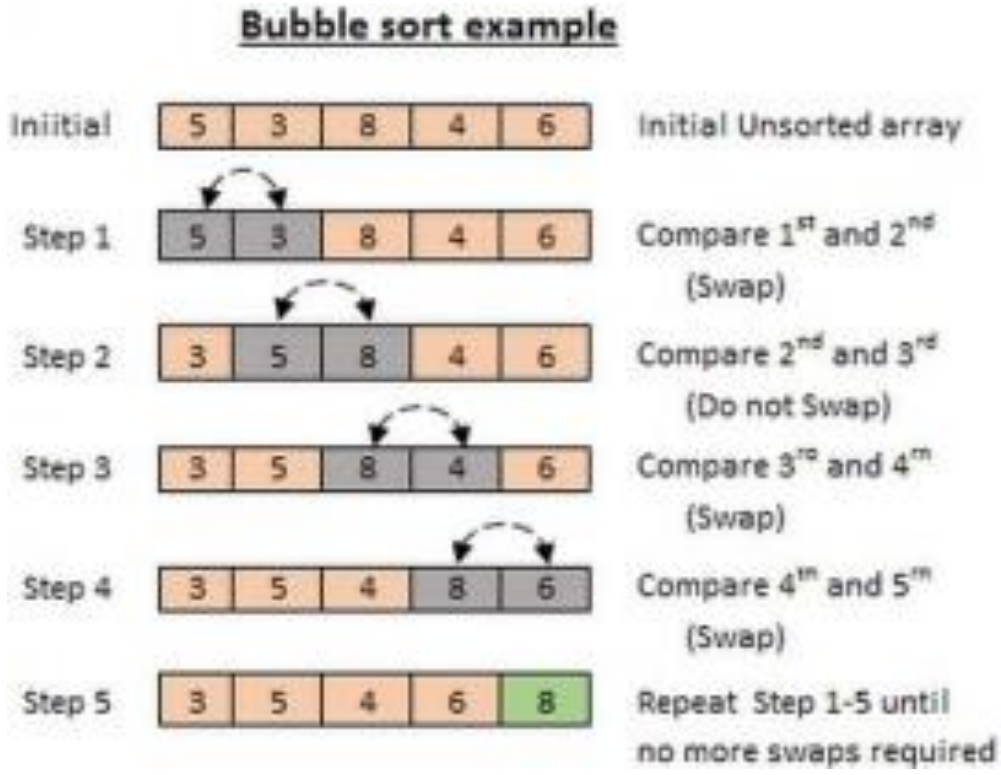
Sorting is the process of arranging all the elements in a particular order (Alphabetical, Ascending numeric Descending numeric)



Bubble Sort

➢ Sorting a List: Bubble Sort

➢ Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.



Sorting a List: Bubble Sort

- Sort an array from smallest to largest

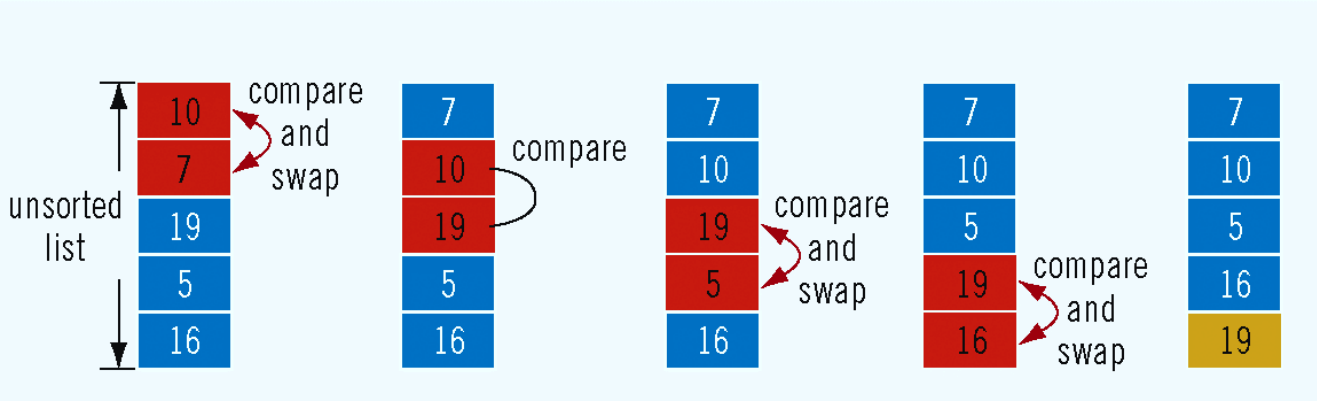
First Pass:

{ 6, 3, 2, 9, 7, 1, 5, 4, 8 }
{ 3, 6, 2, 9, 7, 1, 5, 4, 8 } current index =0
{ 3, 2, 6, 9, 7, 1, 5, 4, 8 } current index =1
{ 3, 2, 6, 9, 7, 1, 5, 4, 8 }
{ 3, 2, 6, 7, 9, 1, 5, 4, 8 }
{ 3, 2, 6, 7, 1, 9, 5, 4, 8 }
{ 3, 2, 6, 7, 1, 5, 9, 4, 8 }
{ 3, 2, 6, 7, 1, 5, 4, 9, 8 }
{ 3, 2, 6, 7, 1, 5, 4, 8, 9 } current index =7

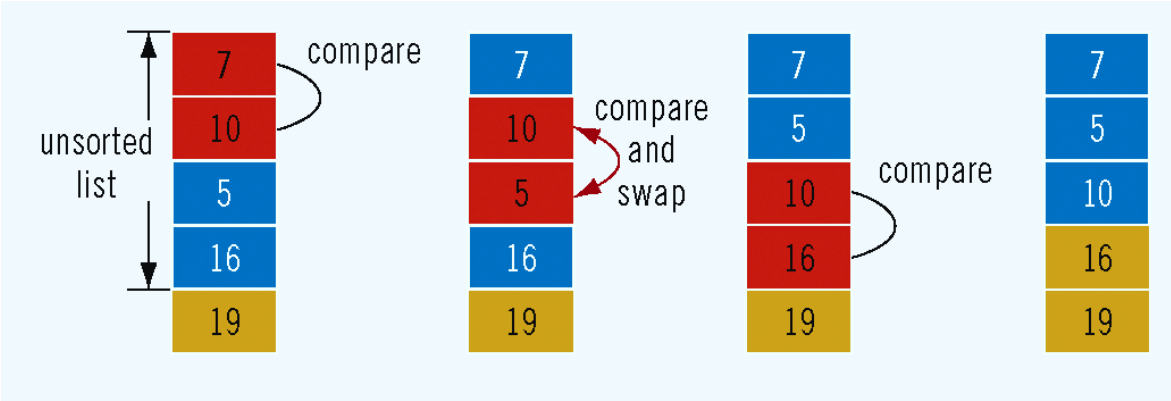
```
for (int currentIndex = 0; currentIndex < endOfArrayIndex - 1; ++currentIndex)
{
    // If the current element is larger than the element after it
    if (array[currentIndex] > array[currentIndex + 1])
    {
        // Swap them
        std::swap(array[currentIndex], array[currentIndex + 1]);
    }
}
```

➢ Example

List of five elements

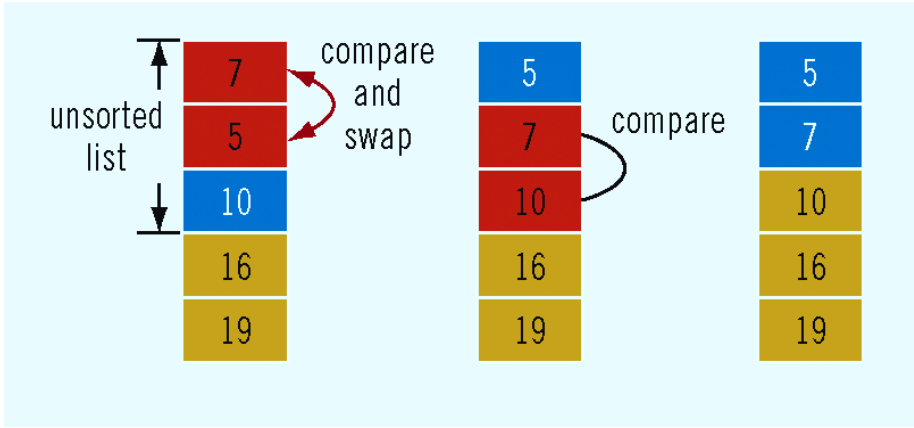


First iteration

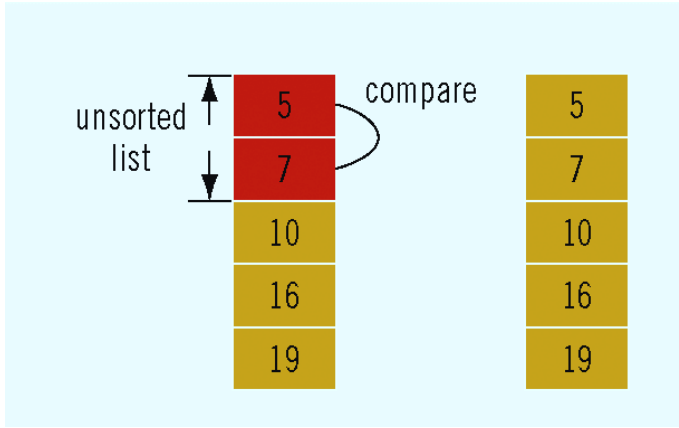


Second iteration

➢ Example (cont.)



Third iteration



Fourth iteration

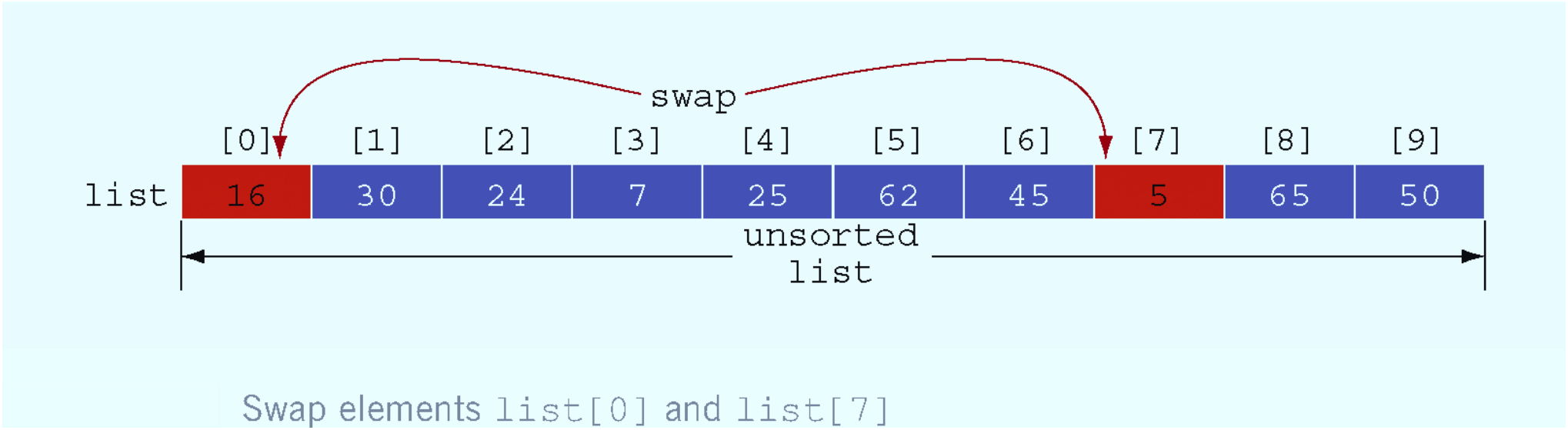
➢ Bubble Sort Code

```
void bubbleSort(int list[], int length)
{
    int temp;
    int iteration;
    int index;
    for (iteration = 1; iteration < length; iteration++)
    {
        for (index = 0; index < length - iteration; index++)
            if (list[index] > list[index + 1])
            {
                temp = list[index];
                list[index] = list[index + 1];
                list[index + 1] = temp;
            }
    }
}
```

Selection Sort

Sorting a List: Selection Sort

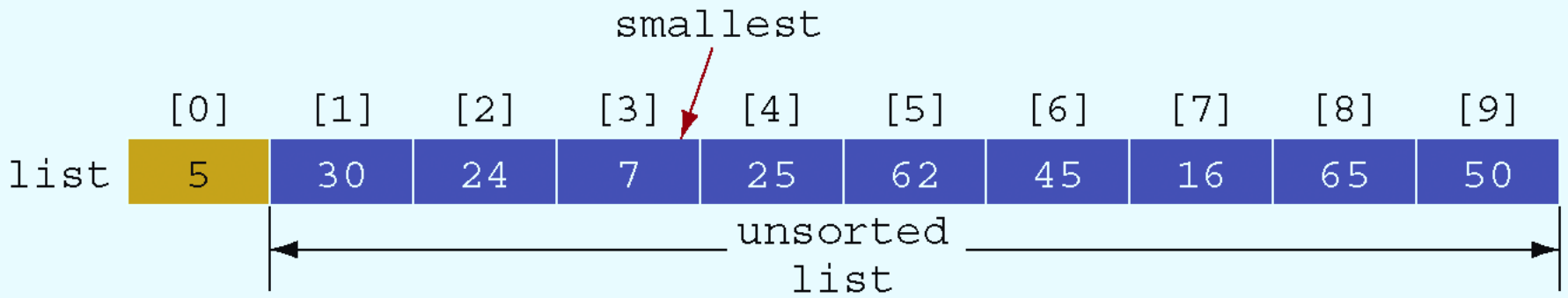
- Selection sort: rearrange list by selecting an element and moving it to its proper position
- Find the smallest (or largest) element and move it to the beginning (end) of the list



Swap elements `list[0]` and `list[7]`

➢ Sorting a List: Selection Sort (cont.)

- On successive passes, locate the smallest item in the list starting from the next element



Smallest element in unsorted portion of `list`

➢ Selection Sort Code

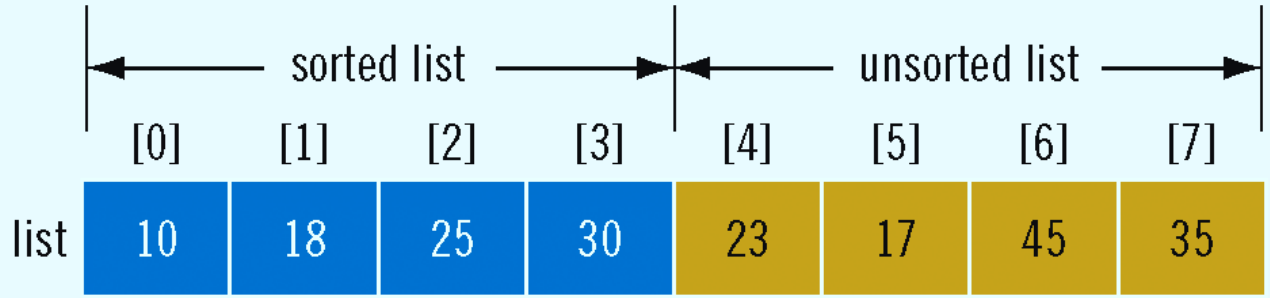
```
for (index = 0; index < length - 1; index++)
{
   a. Find the location, smallestIndex, of the smallest element in
      list[index]...list[length].
   b. Swap the smallest element with list[index]. That is, swap
      list[smallestIndex] with list[index].
}
```

```
void selectionSort(int list[], int length)
{
    int index;
    int smallestIndex;
    int minIndex;
    int temp;

    for (index = 0; index < length - 1; index++)
    {
        //Step a
        smallestIndex = index;

        for (minIndex = index + 1; minIndex < length; minIndex++)
            if (list[minIndex] < list[smallestIndex])
                smallestIndex = minIndex;

        //Step b
        temp = list[smallestIndex];
        list[smallestIndex] = list[index];
        list[index] = temp;
    }
}
```

Insertion Sort

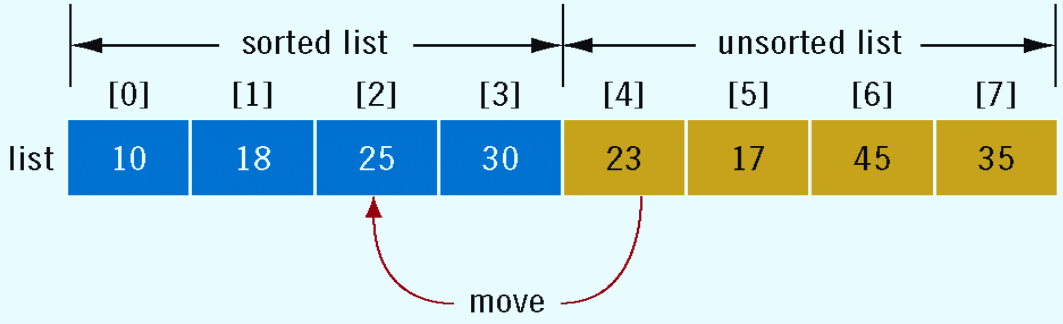
➢ Sorting a List: Insertion Sort

➢ The insertion sort algorithm sorts the list by moving each element to its proper place.

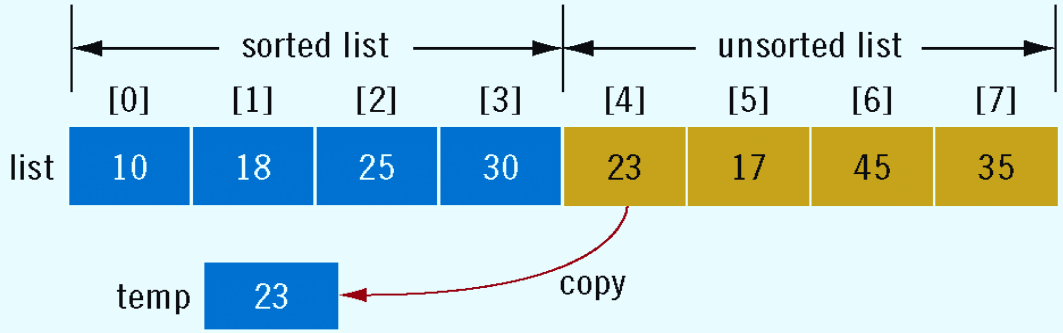


Sorted and unsorted portion of `list`

➢ Sorting a List: Insertion Sort (cont.)

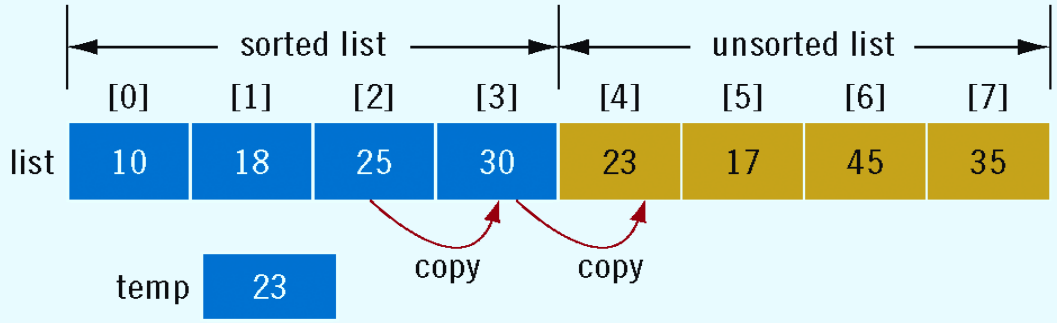


Move `list[4]` into `list[2]`

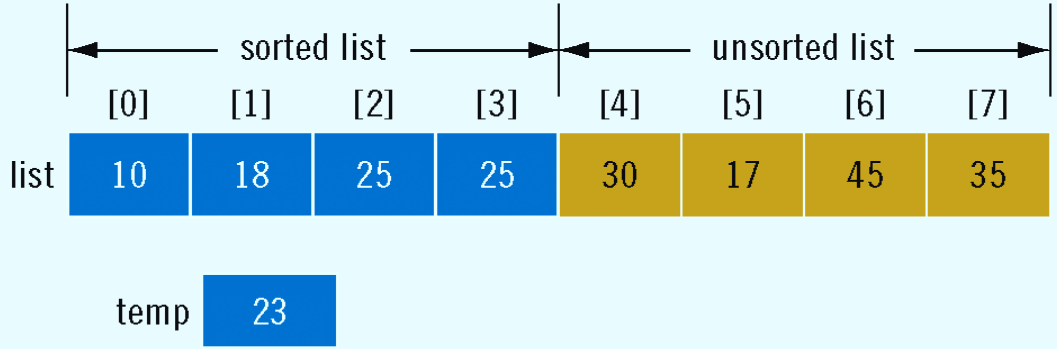


Copy `list[4]` into `temp`

➢ Sorting a List: Insertion Sort (cont.)

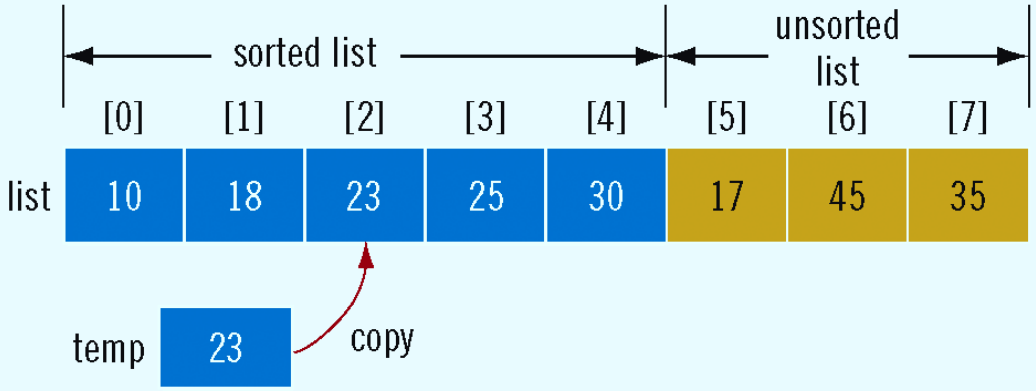


`list` before copying `list[3]` into `list[4]` and then `list[2]` into `list[3]`



`list` after copying `list[3]` into `list[4]` and then `list[2]` into `list[3]`

➢ Sorting a List: Insertion Sort (cont.)



list after copying temp into list[2]

➢ Insertion Sort Code

```
for (firstOutOfOrder = 1; firstOutOfOrder < listLength;
                          firstOutOfOrder++)
  if (list[firstOutOfOrder] is less than list[firstOutOfOrder - 1])
  {
      copy list[firstOutOfOrder] into temp

      initialize location to firstOutOfOrder

      do
      {
         a. copy list[location - 1] into list[location]
         b. decrement location by 1 to consider the next element
            in the sorted portion of the array
      }
      while (location > 0 && the element in the upper list at
                            location - 1 is greater than temp)
  }
copy temp into list[location]
```

➢ Insertion Sort Code (cont.)

```
void insertionSort(int list[], int listLength)
{
    int firstOutOfOrder, location;
    int temp;

    for (firstOutOfOrder = 1; firstOutOfOrder < listLength;
                              firstOutOfOrder++)
        if (list[firstOutOfOrder] < list[firstOutOfOrder - 1])
        {
            temp = list[firstOutOfOrder];
            location = firstOutOfOrder;

            do
            {
                list[location] = list[location - 1];
                location--;
            }
            while (location > 0 && list[location - 1] > temp);

            list[location] = temp;
        }
} //end insertionSort
```

➢ Comparison

	Best Time Complexity	Worst Time Complexity	Comments
Bubble sort	O(n^2)	O(n^2)	It can be optimized by stopping the algorithm if inner loop didn't cause any swap.
Selection sort	O(n^2)	O(n^2)	It never makes more than O(n) swaps and can be useful when memory write is a costly operation.
Insertion sort	O(n)	O(n^2)	It takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

➢ Major Sorting Algorithms

- Selection Sort
- Bubble Sort
- Recursive Bubble Sort
- Insertion Sort
- Recursive Insertion Sort
- Merge Sort
- Iterative Merge Sort
- Quick Sort
- Iterative Quick Sort
- Heap Sort
- Counting Sort
- Radix Sort
- Bucket Sort
- ShellSort
- TimSort
- Comb Sort
- Pigeonhole Sort
- Cycle Sort
- Cocktail Sort
- Strand Sort
- Bitonic Sort
- Pancake sorting
- Binary Insertion Sort
- BogoSort or Permutation Sort
- Gnome Sort
- Sleep Sort – The King of Laziness / Sorting while Sleeping
- Structure Sorting (By Multiple Rules) in C++
- Stooge Sort
- Tag Sort (To get both sorted and original)
- Tree Sort
- Cartesian Tree Sorting
- Odd-Even Sort / Brick Sort
- QuickSort on Singly Linked List
- QuickSort on Doubly Linked List
- 3-Way QuickSort (Dutch National Flag)
- Merge Sort for Linked Lists
- Merge Sort for Doubly Linked List
- 3-way Merge Sort

QUIZ

- Sort an array consists of 5 student names alphabetically in ascending order

- Ex: arr[]={Mahmoud, Ayman , Jana, Ziad, Hend}

- Output:
  Ayman
  Hend
  Jana
  Mahmoud
  Ziad

Thank you