Data Structures and Algorithms

Lecture 6: Sorting – Bubble, Insertion and Selection

Sorting Algorithm

A sorting algorithm is used to arrange elements of an array/list in a specific order.

Here, we are sorting the array in ascending order. There are various sorting algorithms that can be used to complete this operation. And, we can use any algorithm based on the requirement.

Bubble Sort

Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.

First Iteration (Compare and Swap)

Starting from the first index, compare the first and the second elements.

If the first element is greater than the second element, they are swapped.

Now, compare the second and the third elements. Swap them if they are not in order.

The above process goes on until the last element.

Remaining Iteration

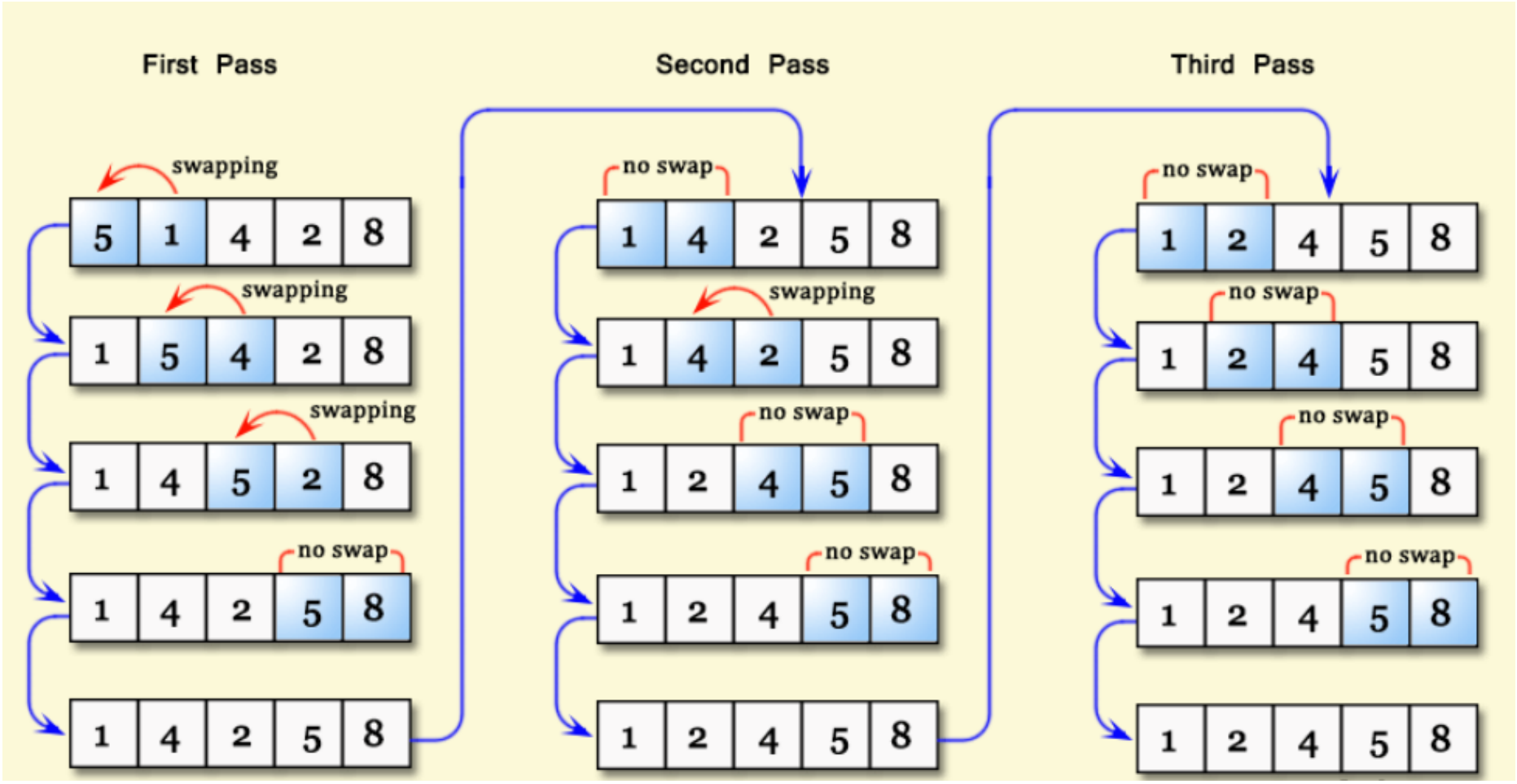
The same process goes on for the remaining iterations.

After each iteration, the largest element among the unsorted elements is placed at the end.

In each iteration, the comparison takes place up to the last unsorted element.

The array is sorted when all the unsorted elements are placed at their correct positions.

Bubble Sort


First Pass
Second Pass
Third Pass
swapping
swapping
swapping
no swap
no swap
no swap
no swap
no swap
no swap
no swap
no swap
5 1 4 2 8
1 5 4 2 8
1 4 5 2 8
1 4 2 5 8
1 4 2 5 8
1 4 2 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8
1 2 4 5 8

Bubble Sort - Algorithm

Algorithm:

```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
end bubbleSort
```

Bubble Sort - Complexity

Bubble Sort compares the adjacent element.

Cycle	Number of Comparisons
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
.....	.....
last	1

Hence, the number of comparisons is:

$(n-1) + (n-2) + (n-3) +.....+ 1 = n(n-1)/2$

nearly equals to n^2

Hence, Complexity: $O(n^2)$

Also, if we observe the code, bubble sort requires two loops. Hence, the complexity is $n*n = n^2$

Bubble Sort - Applications

Bubble sort is used if

- complexity does not matter
- short and simple code is preferred

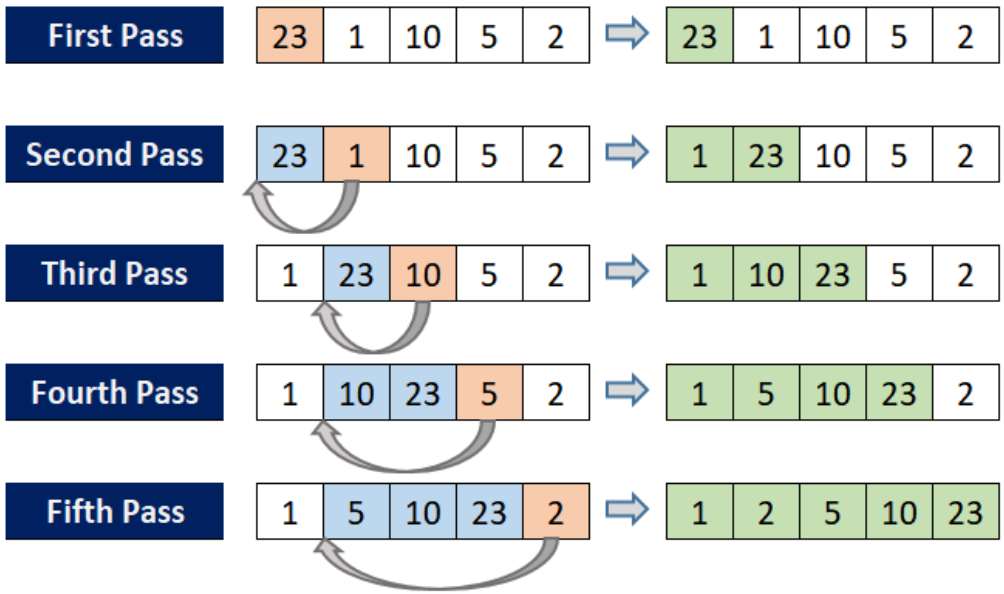
Insertion Sort

- Insertion sort is based on the idea of consuming one element from unsorted array and inserting it at the correct position in the sorted array.
- This will result into increasing the length of the sorted array by one and decreasing the length of unsorted array by one after each iteration.
- To understand the insertion sort, lets consider an unsorted array [23, 1, 10, 5, 2] and discuss each step taken to sort the array in ascending order. In every pass, one element is taken from the unsorted array and inserted it at the correct position in the sorted array.

Insertion Sort

First Pass: The whole array is unsorted array and (23) is taken to be inserted into the sorted array.

Second Pass: (1) is the taken from unsorted array to insert into sorted array. It is compared with all elements of sorted array and found that (23) is the only number which is greater than (1). (1) is inserted into the sorted array and position of (23) is shifted by one place in the array.

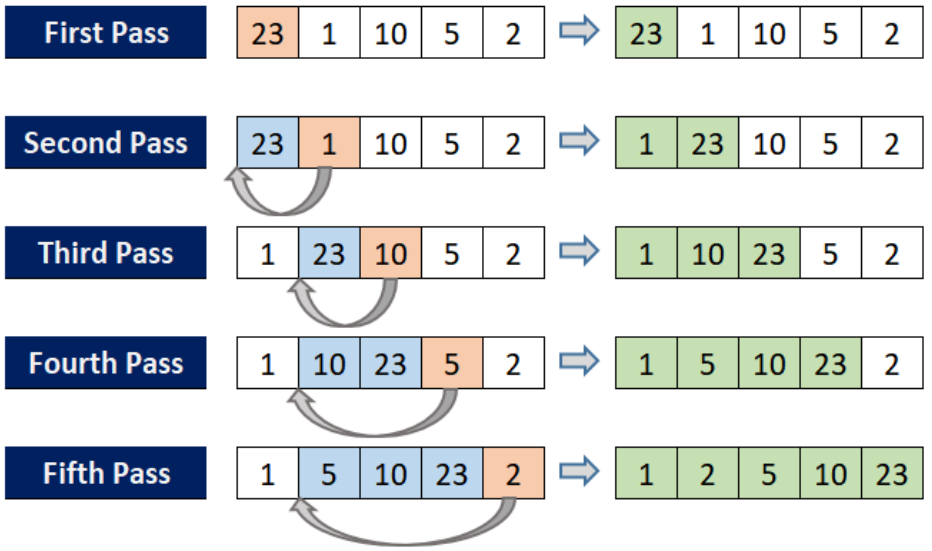


Insertion Sort

Third Pass: (10) is taken from the unsorted array and compared with all elements of sorted array. (23) is the only number in the sorted array which is greater than (10). (10) is inserted into the sorted array and position of (23) is shifted by one place in the array.

Fourth Pass: (5) is compared with all elements of sorted array and it is found that (10) and (23) are the numbers which need to be shifted by one place to insert (5) into the sorted array.

Fifth Pass: To insert (2) into the sorted array, (5), (10) and (23) are shifted by one place.



Insertion Sort - Algorithm

```
insertionSort(array)
        mark first element as sorted
        for each unsorted element X
          'extract' the element X
          for j <- lastSortedIndex down to 0
            if current element j > X
              move sorted element to the right by 1
          break loop and insert X here
end insertionSort
```

Insertion Sort - Complexity

Insertion sort takes maximum time if the array is in the reverse order and minimum time when the array is already sorted. In worst case scenario every element is compared with all other elements. Therefore, for N elements there will be N^2 comparison hence the time complexity of insertion sort is $\Theta(N^2)$.

Insertion Sort - Applications

The insertion sort is used when:

- the array is has a small number of elements
- there are only a few elements left to be sorted

Selection Sort

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

The selection sort algorithm is as follows:

Step 1: Set Min to location 0 in Step 1.

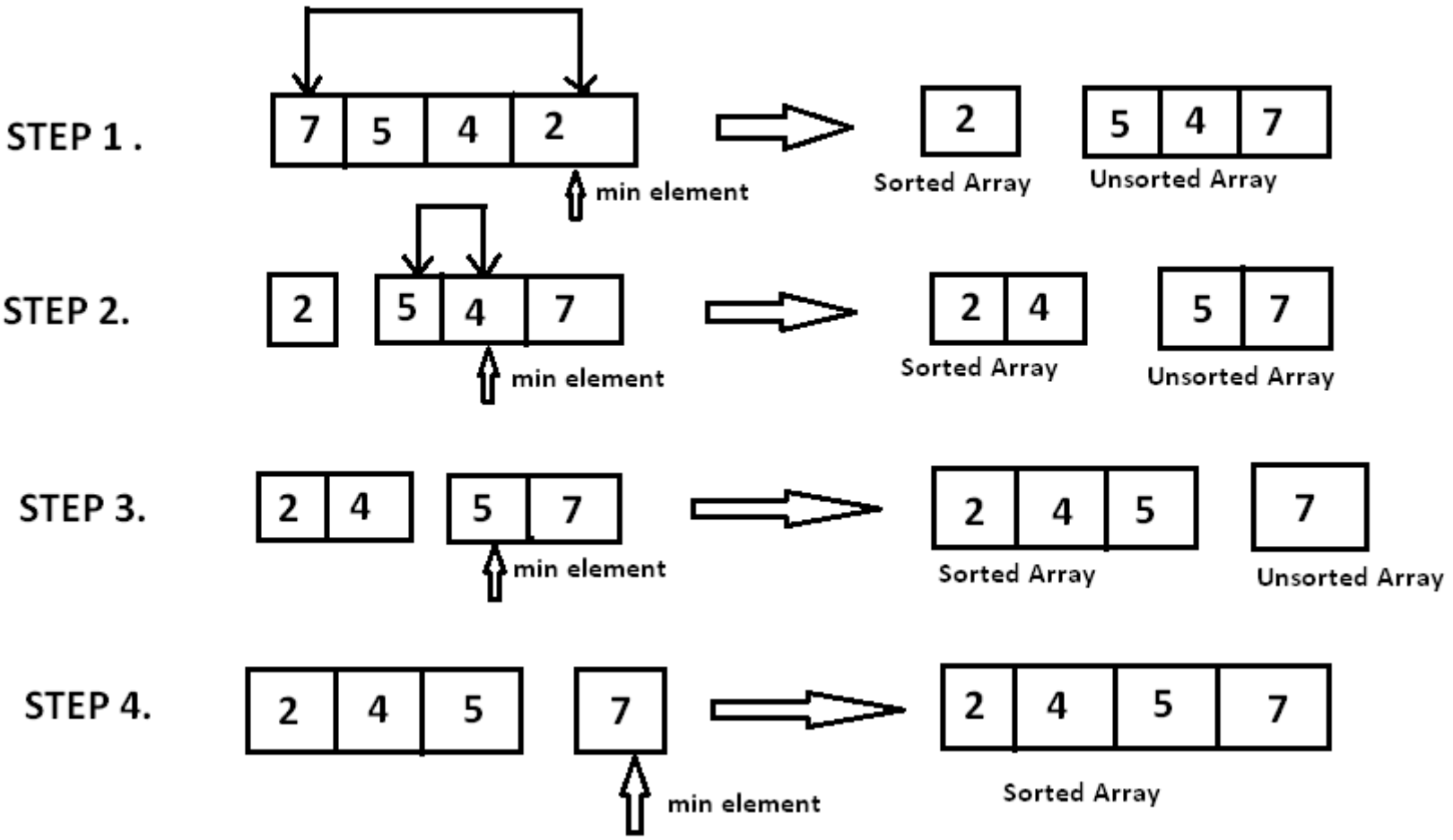
Step 2: Look for the smallest element on the list.

Step 3: Replace the value at location Min with a different value.

Step 4: Increase Min to point to the next element

Step 5: Continue until the list is sorted.

Selection Sort



Selection Sort - Algorithm

Algorithm:

```
selectionSort(array, size)
        repeat (size - 1) times
        set the first unsorted element as the minimum
        for each of the unsorted elements
          if element < currentMinimum
            set element as new minimum
        swap minimum with first unsorted position
end selectionSort
```

Selection Sort - Complexity

Bubble Sort compares the adjacent element.

Cycle	Number of Comparisons
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
.....	.....
last	1

Hence, the number of comparisons is:

$(n-1) + (n-2) + (n-3) + ... + 1 = n(n-1)/2$

nearly equals to n^2

Hence, Complexity: $O(n^2)$

Also, if we observe the code, insertion sort requires two loops. Hence, the complexity is $n*n = n^2$

Selection Sort - Applications

The selection sort is used when:

- a small list is to be sorted
- cost of swapping does not matter
- checking of all the elements is compulsory
- cost of writing to a memory matters like in flash memory (number of writes/swaps is O(n) as compared to O(n2) of bubble sort)

Recall:

- Sorting Algorithms
- Bubble sort – algorithm, complexity and applications
- Insertion sort – algorithm, complexity and applications
- Selection sort – algorithm, complexity and applications