Certificate - Software Development

Array Sorting

Array Sorting

- Here we are looking at algorithms which are used to arrange the elements of an array in ascending or descending order.
- There are two main types of Sort algorithms.

1. **Compare & Exchange Sorts**
   - Algorithms are Easier but Less Efficient.
2. **Divide and Conquer Sorts**
   - Algorithms are Complex but Very Efficient specially on large arrays.

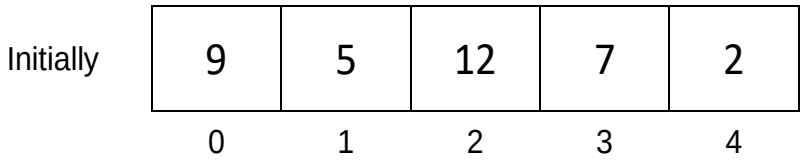
Compare & Exchange Sorts

- Mechanism for sorting is comparing elements and then exchanging them.
- Efficiency class is **O (N^2)**
- More suitable for small arrays.
- Examples:
    1. Bubble Sort
    2. Selection Sort
    3. Insertion Sort

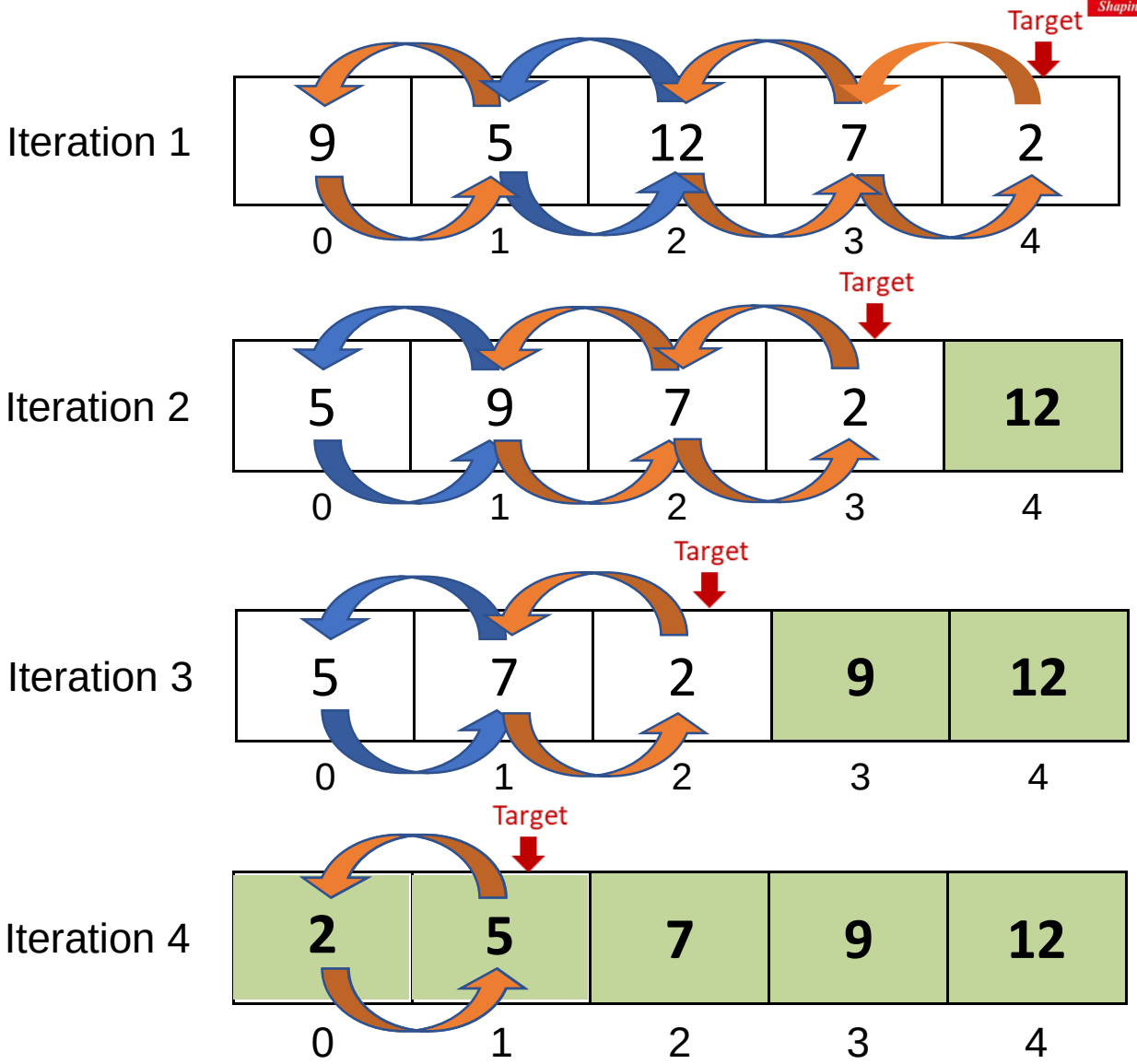
Divide and Conquer Sorts

- Mechanism for sorting is progressively dividing array into smaller segments until they are sorted then merge them
- Efficiency class is **O (N log N)**
- More suitable for large arrays.
- Examples:
    1. Shell Sort
    2. Merge Sort
    3. Quick Sort

Bubble Sort



- N – 1 Passes(Iterations)
- Target starts from Last Element
- Target comes down to 1
- Adjoining pairs are compared and swapped if they are out of order



Bubble Sort – Example 1

```
void BubbleSort(int arr[])
{       int target = SIZE-1;
        while(target > 0)
        {       int j;
                for(j=0; j < target; j++)
                        if(arr[j] > arr[j+1])
                                swap(arr, j, j+1);
                target --;
        }
}
swap(int arr[], int x, int y)
{       int hold;
        hold = arr[x];
        arr[x] = arr[y];
        arr[y] = hold;
}
```

Key Features

- ✓ Target starts from the Last Element
- ✓ Target is reduced by 1 after each iteration
- ✓ This has N-1 Iterations (Passes)
- ✓ Up to Target adjoining pairs are compared
- ✓ Adjoining pair is Swapped if out of order
- ✓ Swapping is done using a function

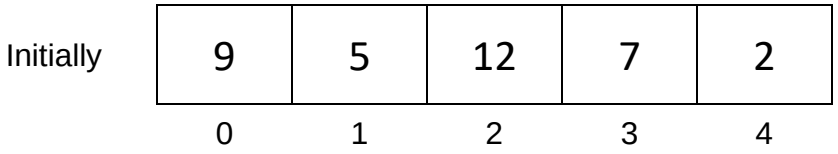
Bubble Sort – Example 2

```
void BubbleSort(int arr[])
{   int target = SIZE-1;
    while(1)
    {   int j, lastswap = -1;
        for(j=0; j < target; j++)
            if(arr[j] > arr[j+1])
            {   swap(&arr[j], &arr[j+1]);
                lastswap = j; }
            if (lastswap == -1)
                break;
            else
                target = lastswap;
    }
}
swap(int *x, int *y)
{   int hold = *x;   *x = *y;   *y = hold;   }
```

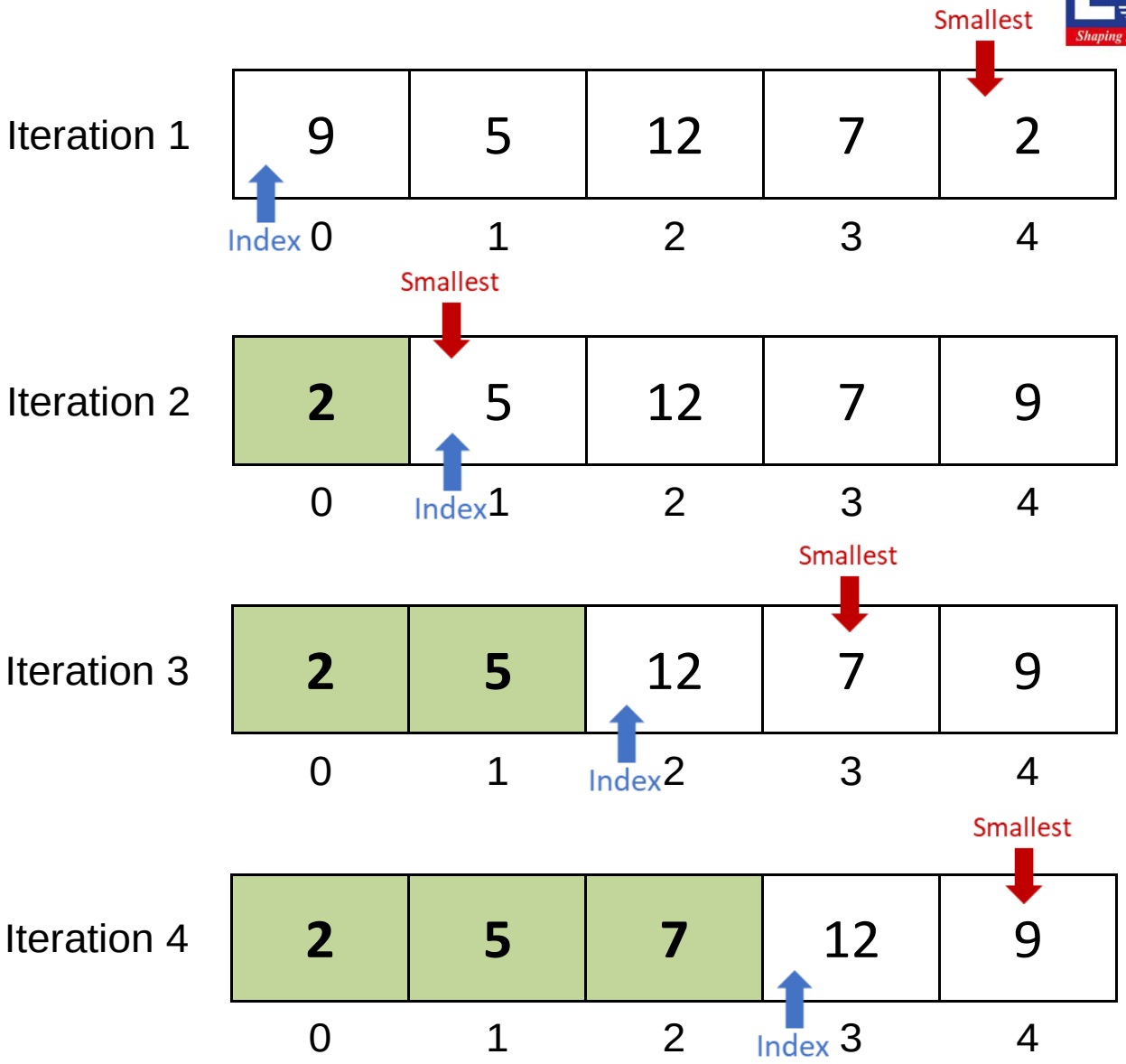
Key Features

- ✓ Target starts from the Last Element
- ✓ Target is set to **lastswap** for next iteration
- ✓ No of Iterations (Passes) are reduced
- ✓ Sorting stops when a pass is done without a swap
- ✓ Adjoining pair is Swapped if out of order
- ✓ Swap function uses pointers. Without passing the array, references to the 2 elements are given

Selection Sort



- This has N-1 passes
- Index starts from 0 and goes up to one before last element
- In each pass smallest is selected and swapped with index element
- Comp starts from index+1 and goes to the last



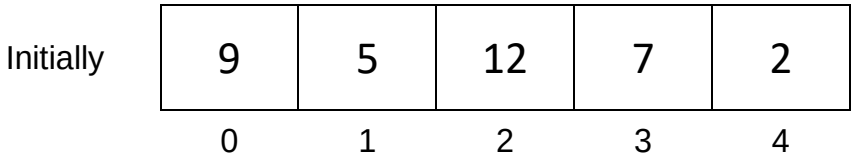
Selection Sort – Example

```
void SelectionSort(int arr[])
{    int index = 0,smallest,comp;
     while(index < SIZE-1)
     {
        smallest = index;
        comp = index+1;
        while(comp < SIZE)
        {
            if (arr[comp] < arr[smallest])
               smallest = comp;
            comp++;
        }
        swap(&arr[smallest], &arr[index]);
        index ++;
        }
}
```

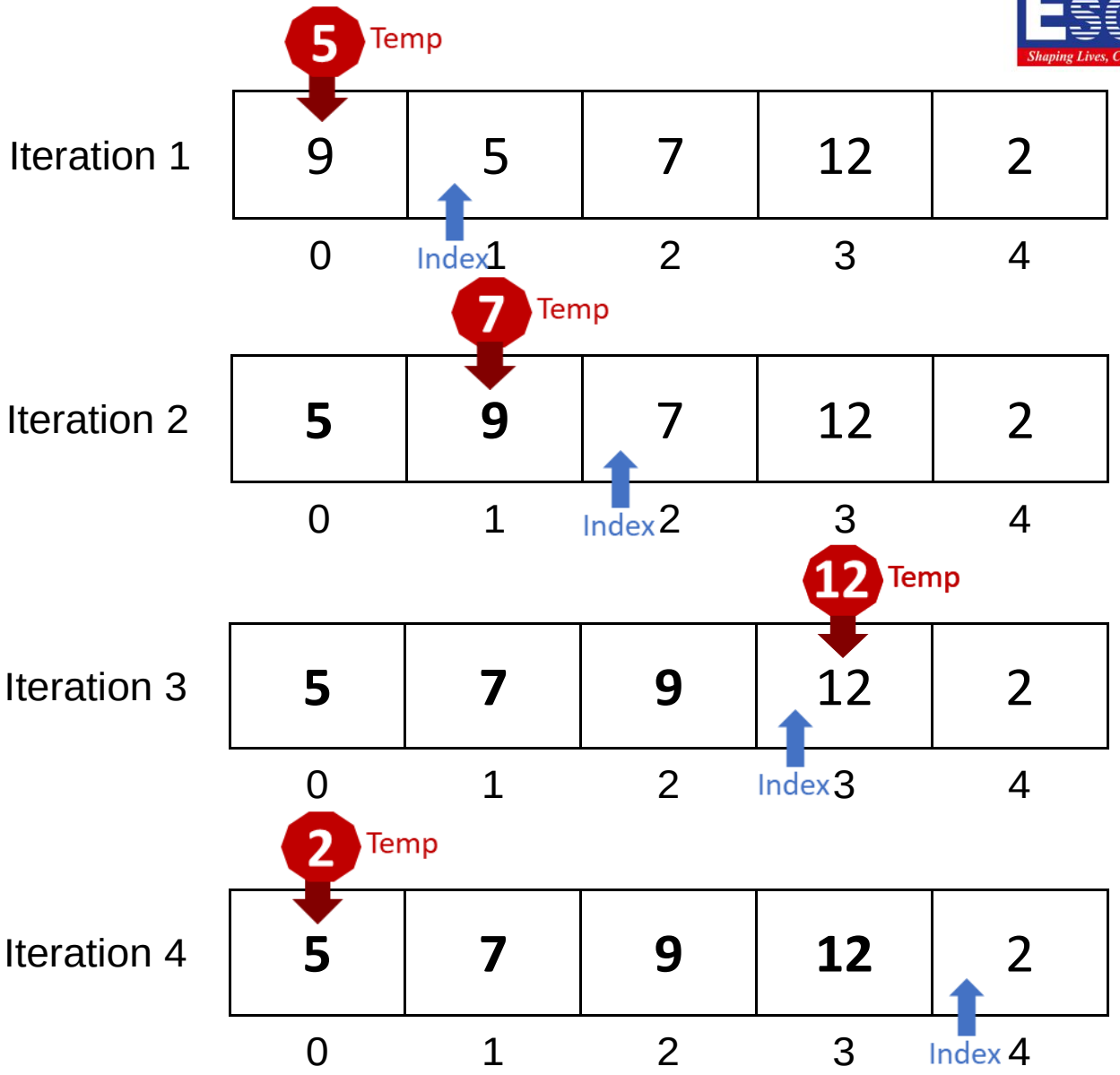
Key Features

- ✓ This always has **N-1** Iterations(Passes)
- ✓ **Index** starts from 0 and goes up to 1 before last
- ✓ **Comp** starts from Index+1 and goes to last
- ✓ At the beginning **Smallest** is set to index position
- ✓ When Comp is lower Smallest is set to that
- ✓ At the end of the pass elements at **Index** and **Smallest** are Swapped (Only 1 Swap per pass)

Insertion Sort



- This also has N-1 passes
- Index starts from 1 and goes up to last
- Element at index is copied to Temp and compared with previous elements
- Temp is inserted to correct place after shifting elements



Insertion Sort – Example

```
void InsertionSort(int arr[])
{    int index, prev, temp;
     index = 1;
     while(index < SIZE)
     {
        temp = arr[index];
        prev = index-1;
        while(prev>=0 && temp<arr[prev])
        {
           arr[prev+1] = arr[prev];
           prev--;
        }
        arr[prev+1] = temp;
        index++;
     }
}
```

Key Features

- ✓ This always has **N-1** Iterations(Passes)
- ✓ **Index** starts from 1 and goes up to last
- ✓ Element at Index is copied to **Temp**
- ✓ Temp is compared with previous elements
- ✓ They are **shifted** to the next slot if Temp is lower
- ✓ Process is repeated until an element lower than Temp is reached or array beginning is reached
- ✓ Temp is inserted to the very next element

How to test your Sorting Algorithms ?

Main Program for Sorting

```
# define SIZE 10
main()
{
   int arr1[]={34,87,12,8,25,90,5,15,2,28};
   int x;
   InsertionSort(arr1);
   for(x=0; x < SIZE; x++)
        printf("%d, ",arr1[x]);
   printf("\n");
}
```

Output

```
2, 5, 8, 12, 15, 25, 28, 34, 87, 90,
```

Efficiency Analysis of Sorting

Description	Bubble Sort	Selection Sort	Insertion Sort
Iterations	Min = 1, Max = **N-1**	Always **N-1**	Always **N-1**
Comparisons	Min = **N-1**	Min = **N (N-1) / 2**	Min = **N (N-1) / 2**
	Max = **N (N-1) / 2**	Max = **N (N-1) / 2**	Max = **N (N-1) / 2**
Interchanges (Swapping)	Min = **0**	Min = **0**	Min = **0**
	Max = **N (N-1) / 2**	**Max = N-1**	Max = **N (N-1) / 2** *(But it is Shifting)*
Big O – Efficiency Class	**O (N^2)**	**O (N^2)**	**O (N^2)**

Lesson Summary

- Sorting Arrays
- Compare & Exchange Sorts
- Divide & Conquer Sorts
- Bubble Sort
- Selection Sort
- Insertion Sort
- Efficiency Analysis of Sorting