Lecture 24.1

Topics

1. Non-Recursive Sort – Selection Sort
2. Non-Recursive Sort – Insertion Sort

---

1. Non-Recursive Sorting Algorithm – Selection Sort

Sorting is one of several important processes in software applications. There are just a few of the non-recursive sorting algorithms such as bubble sort, selection sort, insertion sort, etc. Let's consider the selection sort here.

- The selection sort is different from the bubble sort, which swaps the values based on the sorting criteria.
- Instead, the selection sort will swap the **indices** where the values can be found and met the sorting criteria.

1.1 Selection Sort

In the selection sort, the data are stored in an array and two values are selected and compared to each other to produce the index information. Note that the selection sort also divides the array into two halves (as explained below).

- Assuming that an ascending sort is desired, for the array of size **`N`**, there will be **`(N – 1)`** passes.
- In each pass, the values are processed (compared) so that the index of the smallest value will be determined. This index information is then used to swap the values appropriately.
- In each pass, the array is divided into two halves. One half has all sorted values and the other half has the unsorted values (the position of the unsorted values may not be the same as the original one).
    - After the first pass is done, there will be one sorted element in the sorted half and **`(N – 1)`** elements in the unsorted half.
    - After the second pass, there will be two elements in the sorted half and **`(N – 2)`** in the unsorted half; and so on.
    - In total, there will be **`(N – 1)`** passes to be performed.

1.2 Selection Sort -- Illustration

Let's look at the steps of the **selection sort** through an example.

Consider the following array,

16	30	24	7	25	62	45	5	65	50

Let's assume that an ascending sort is being applied to the array. The selection sort will

(a) Select the smallest element from the unsorted array (or portion of the array that is unsorted), and then

(b) Swap this smallest element to the beginning of the unsorted array (or to become the last element in the sorted portion of the array).

The first time around, the smallest element is found for the entire list. The second time around, the smallest element is located in the remaining portion of the array starting from the second element on. This process stops when there is one element left from the array.

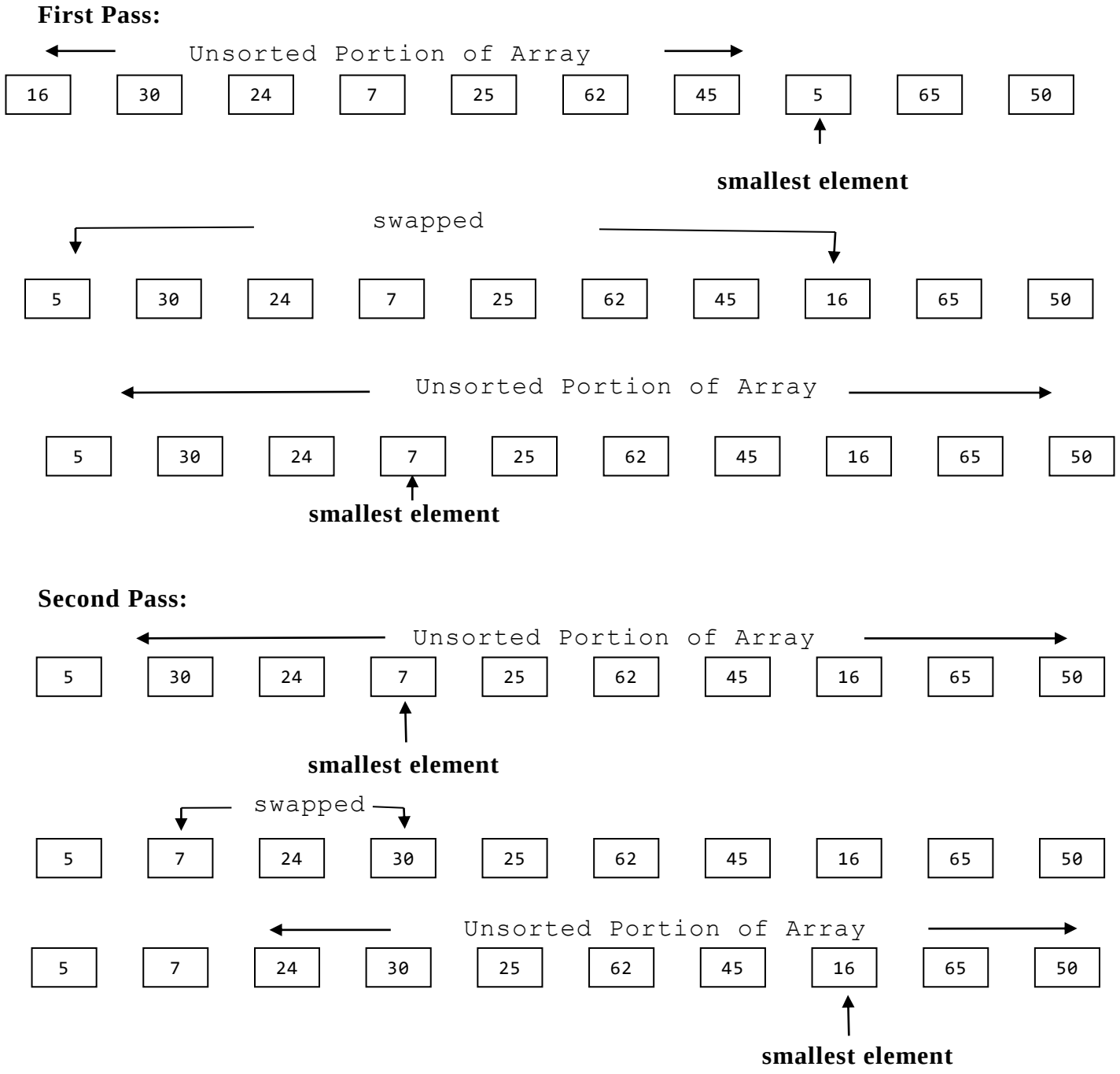


Figure 1 Stepping of the selection sort

Continue the above steps until all elements are placed in correct order. Note that the process stops **when the unsorted portion of the array is reduced to just one element.**

1.3 Selection Sort – Coding

The code is listed below; it is the copy from a textbook. Some functions are added to provide options or alternatives to the existing code.

Example 1

```
/**
 * Program Name: cis27L2411.c
 * Discussion:   Selection Sort
 */
/* ****************************************************************
**  This program is provided to the professors and students   **
**  using "Computer Science: A Structured Approach Using C,   **
**  Second Edition." All reprints in any form must cite this  **
**  copyright.                                                **
**                                                            **
**  Copyright(c) 2001 by Brooks/Cole                          **
**  A division of Thomson Learning                            **
****************************************************************
*/

/* Test driver for sorting.
Written by:
Date:
*/
#include <stdio.h>

#define MAX_ARY_SIZE 15

/* Prototype Declarations */
void selectionSort(int list[], int last);
void exchangeSmallest(int list[], int first, int last);

int main(void)
{
    /* Local Declarations */
    int i;
    int       ary[MAX_ARY_SIZE] = { 89, 72,  3, 15, 21,
        57, 61, 44, 19, 98,
        5, 77, 39, 59, 61 };
    /* Statements */
    printf("Unsorted array: ");
    for (i = 0; i < MAX_ARY_SIZE; i++)
        printf("%3d", ary[i]);

    selectionSort(ary, MAX_ARY_SIZE - 1);

    printf("\nSorted array:   ");
    for (i = 0; i < MAX_ARY_SIZE; i++)
        printf("%3d", ary[i]);
    printf("\n");
    return 0;
}      /* main */

    /* ================== selectionSort ====================
    Sorts by selecting smallest element in unsorted portion
    of array and exchanging it with element at the
    beginning of the unsorted list.
    Pre   list must contain at least one item
    last contains index to last element in list
```

```
        Post  list rearranged smallest to largest
        */
    void selectionSort(int list[],
        int last)
    {
        /* Local Definitions */
        int current;

        /* Statements */
        for (current = 0; current < last; current++)
            exchangeSmallest(list, current, last);

        return;
    } /* selectionSort */

        /* ================== exchangeSmallest ==================
        Given array of integers, place smallest element into
        position in array.
        Pre   list must contain at least one element
        current is beginning of array (not always 0)
        last element in array must be >= current
        Post  returns index of smallest element in array
        */
    void exchangeSmallest(int list[],
        int current,
        int last)
    {
        /* Local Definitions */
        int walker;
        int smallest;
        int tempData;

        /* Statements */
        smallest = current;
        for (walker = current + 1; walker <= last; walker++)
            if (list[walker] < list[smallest])
                smallest = walker;

        /* Smallest selected: exchange with current element */
        tempData = list[current];
        list[current] = list[smallest];
        list[smallest] = tempData;

        return;
    } /* exchangeSmallest */

/* Results:
Unsorted array:  89 72  3 15 21 57 61 44 19 98  5 77 39 59 61
Sorted array:     3  5 15 19 21 39 44 57 59 61 61 72 77 89 98
*/
```

OUTPUT

```
Unsorted array:  89 72  3 15 21 57 61 44 19 98  5 77 39 59 61
Sorted array:     3  5 15 19 21 39 44 57 59 61 61 72 77 89 98
```

2. Non-Recursive Sort – Insertion Sort

Insertion sort is an algorithm that will also work with data that are in two groups:

(1) Sorted sequence (**S**), and

(2) Unsorted sequence (**US**).

The sorting will be done in several passes just as the bubble sort or selection sort – There are **(N-1)** passes for a given sequence of **N** values.

For each pass,

- The first element **ue[0]** from the unsorted sequence **US** will be moved to the sorted sequence **S**.
- This **ue[0]** element will be inserted at the appropriate place in the **S** sequence so that all elements in **S** sequence will be sorted.

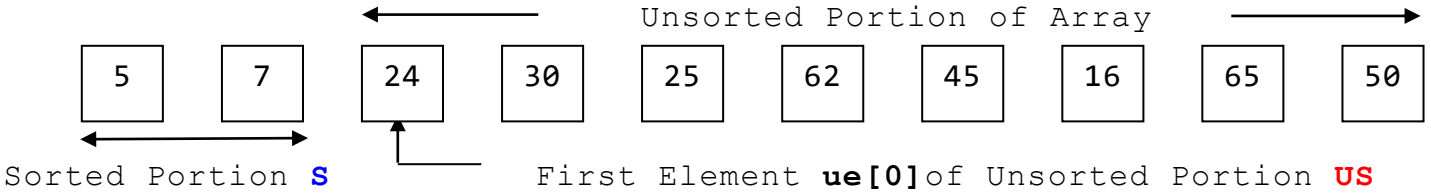
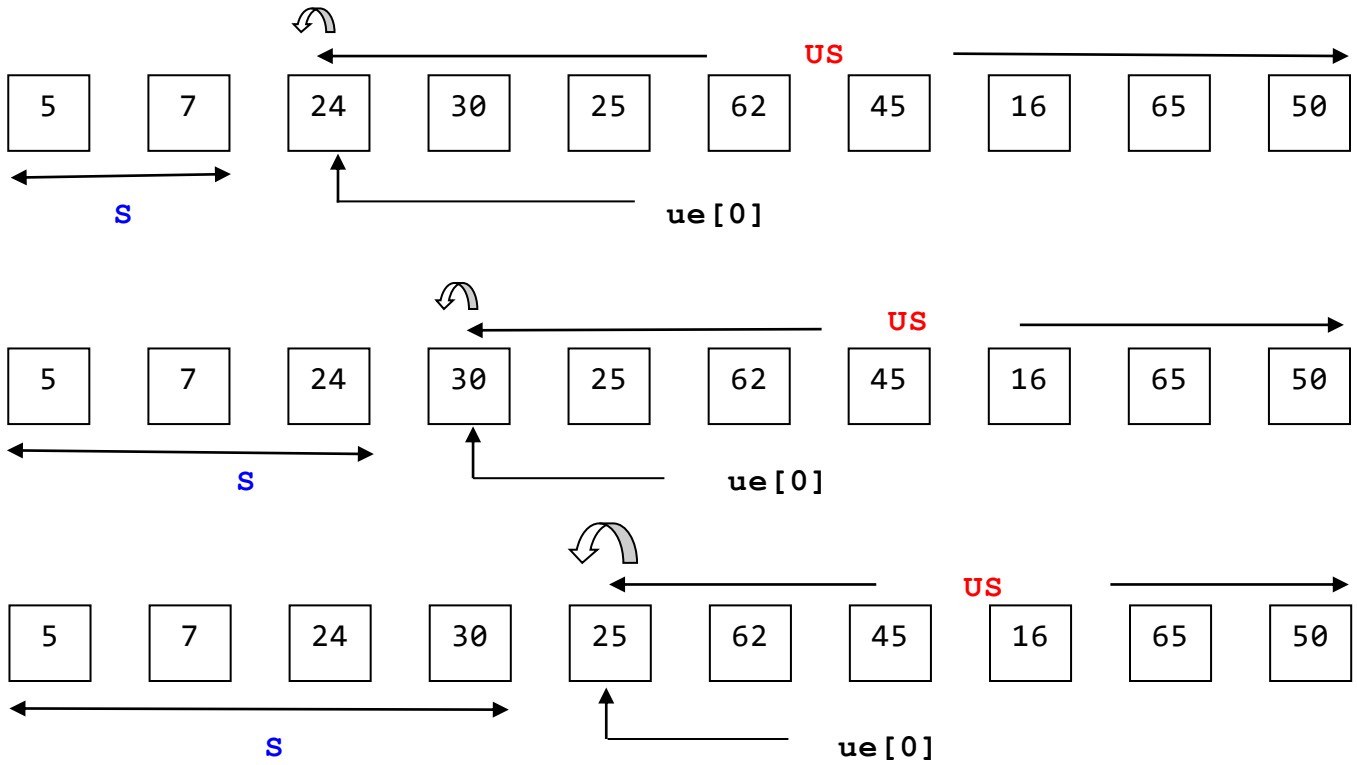


Figure 2 Assuming the **insertion sort** has was performed up to and including the first 2 elements

The subsequence passes are illustrated as follows,



Final Pass

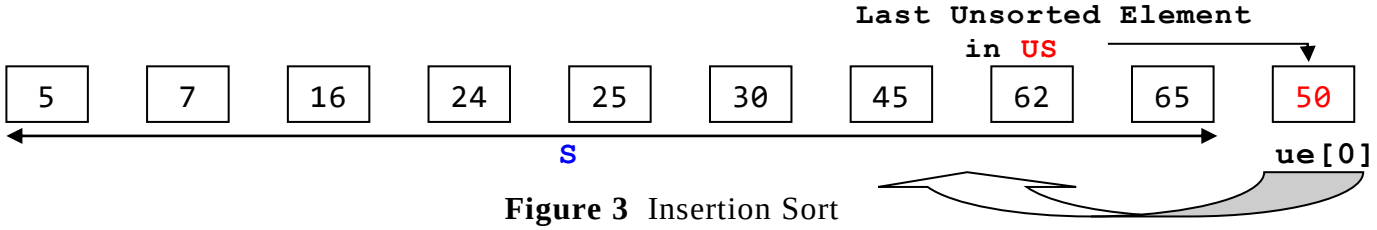


Figure 3 Insertion Sort

Example 2

```c
/* FILE #1 */
/**
 * Program Name: cis27L2412Driver.c
 * Discussion:   Insertion Sort
 */

#define _CRT_SECURE_NO_WARNINGS

// Include/Header File(s)
#include <stdio.h>
#include "cis27L2812.h"

// Application Driver
int main() {
    int i;
    int iSize;
    int iAry[MAX_ARY_SIZE] = {89, 72, 3, 15, 21,
        57, 61, 44, 19, 98,
        5, 77, 39, 59, 61};

    printf("Unsorted array: ");
    for (i = 0; i < MAX_ARY_SIZE; i++)
        printf("%3d", iAry[i]);

    iSize = MAX_ARY_SIZE;
    sortInsertion(iAry, iSize);

    printf("\nSorted array:   ");
    for (i = 0; i < MAX_ARY_SIZE; i++)
        printf("%3d", iAry[i]);

    printf("\n");

    return 0;
}

/* FILE #2 */
/**
 * Program Name: cis27L2812.h
 * Discussion:   Insertion Sort
 */

// Include/Header File(s)
#include <stdio.h>

#define MAX_ARY_SIZE 15

// Function prototypes

/**
 * Function Name:  sortInsertion()
 * Description:    Insertion sort
```

```
 * Pre:            Unsorted integer Array and its size
 * Post:           Sorted integer array
 */
void sortInsertion(int[], int);


/* FILE #3 */
/**
 * Program Name: cis27L2812.c
 * Discussion:   Insertion Sort
 */

// Include/Header File(s)
#include "cis27L2812.h"

void sortInsertion(int iAry[], int iSize) {
    int i, j;
    int iSorted;
    int iTemp;

    iSize = MAX_ARY_SIZE;
    for (i = 1; i < iSize; i++) {
        iSorted = 0;
        iTemp = iAry[i];

        for (j = i - 1; j >= 0 && !iSorted;) {
            if (iTemp < iAry[j]) {
                iAry[j + 1] = iAry[j];
                j--;
            } else {
                iSorted = 1;
            }
        }

        iAry[j + 1] = iTemp;
    }
}
```

OUTPUT

```
Unsorted array:  89 72  3 15 21 57 61 44 19 98  5 77 39 59 61
Sorted array:     3  5 15 19 21 39 44 57 59 61 61 72 77 89 98
```