Insertion Sort

Assignment 5

Due date: Sunday, 1 Dec (11:59 pm)

1 Overview

In this assignment, students will develop and implement a program that sorts an array using the **insertion** sort algorithm.

The primary objective of this assignment is for students gain experience in using arrays.

2 The Program

The program performs the same task as the selection sort program that was covered in class. Specifically, it generates an array of 50 random integers in the range $[0 \ldots 99]$ and sorts the array into ascending order; the array is printed both before and after sorting, 10 integers per line.

The only significant difference in the two
programs is the algorithm used to sort the array.

Since the two programs are so similar, students are encouraged to reuse as much of the selection sort program as they wish. For their convenience, the source code of the program can downloaded from the following URL.

`http://cs.nyu.edu/~campbell/Sort.c`

This writeup contains a *brief* description of the insertion sort algorithm; students are encouraged to follow the links to sorting demonstrations listed on the Downloads section of the course web site for a more complete explanation and illustration of the algorithm.

3 Program Structure

In addition to the `main` function, as well as any other functions reused from the selection sort program, the program must define the following two functions:

3.1 The `insertionSort` Function

The `insertionSort` function has the following prototype:

```
    void insertionSort(int a[], int n);
```

It sorts the array `a` with effective size `n`.

There are several similarities between insertion sort and selection sort. Both algorithms:

- divide the array into sorted and unsorted slices;
- make a number of passes through the array; and
- call upon an auxiliary function to do much of the work.

This time, the sorted slice consists of elements with the smaller index numbers (the elements on the "left" of the array). In each pass, the first element of the unsorted slice (*not* necessarily the largest nor the smallest) is inserted into the sorted slice by calling the `insert` function. Once again, the sorted slice will grow (and the unsorted slice shrink) by one element each pass.

3.2 The `insert` Function

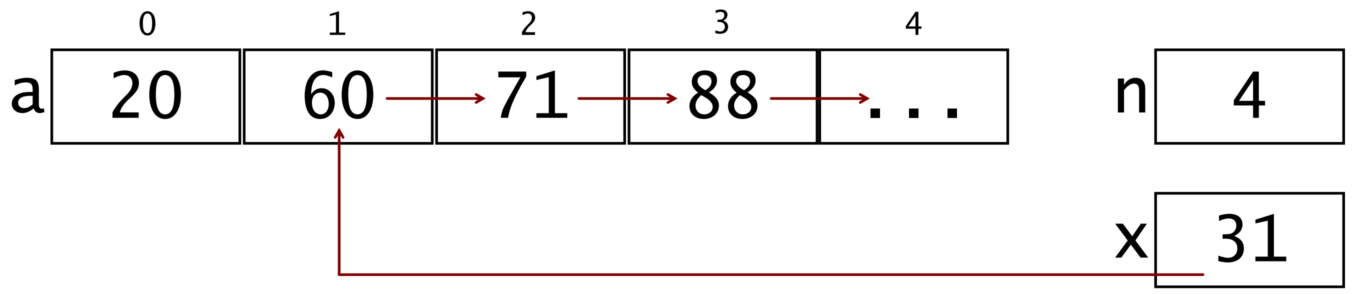
The `insert` function has the following prototype:

```
    void insert(int a[], int n, int x);
```

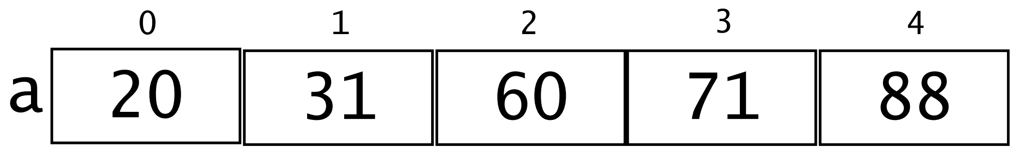
Upon function entry, the slice consisting of the first `n` elements of array `a` are assumed to be in sorted order. Array elements are scanned from *right* to *left*. Elements of the slice larger than the integer parameter `x` are moved one position to the right, and `x` is inserted into the array in a location to the right of the greatest element less than or equal to `x`. Upon function exit, the first `n` plus one elements of `a` must be in sorted order.

The diagram on the next page illustrates the effect of calling the `insert` function.

Parameter values upon function entry:



The array a upon function exit:



4 Sample Output

```
Elements of the unsorted array

  88  71  60  20  31  69  71  34  12  89
  60  70   8  25  64  46  28  76  71   8
  58  34  88  93   2  85  86   3  62  89
  73   0  73  79  93  92   7  60  39  76
  86  55  89  64   4  29  39  45  75   9

Elements of the sorted array

   0   2   3   4   7   8   8   9  12  20
  25  28  29  31  34  34  39  39  45  46
  55  58  60  60  60  62  64  64  69  70
  71  71  71  73  73  75  76  76  79  85
  86  86  88  88  89  89  89  92  93  93
```

5 Testing the Program

This program can be tested adequately by executing it several times and verifying that the array is indeed sorted each time.