Lab 10 Report: Insertion Sort

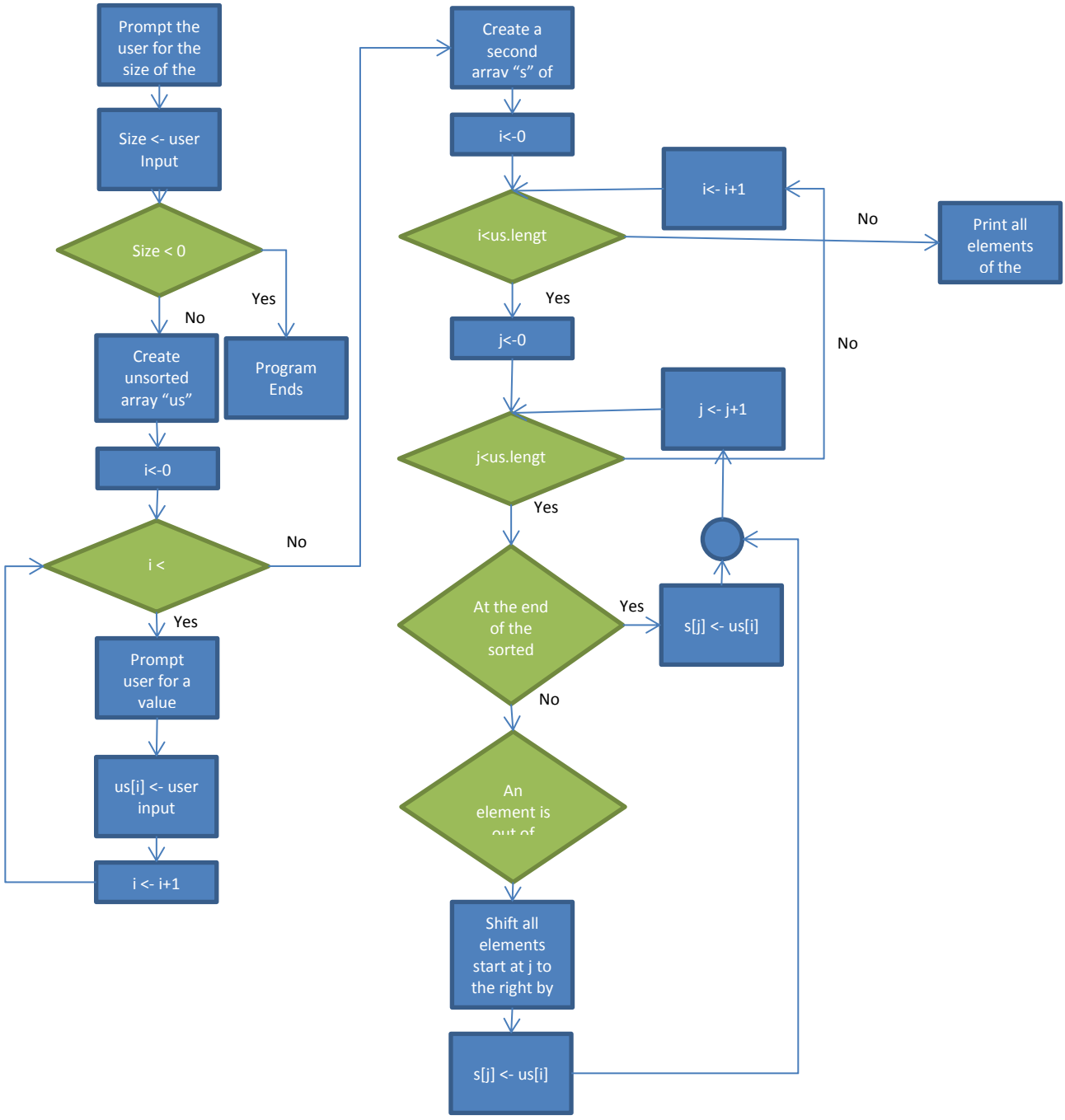
Problem

We had to write a program in which a user populated an array of integers and then it was sorted using insertion sort. Finally, the program printed out the sorted array to the console.

Proposed Solution

1. Prompt the user for the size of the array
2. If the size is a negative value then quit the program
3. Create an array of integers of size given in step 1
4. For each element in the array
    a. Prompt the user to input a value
    b. Store that value as that element of the array
5. Create a second array the same size as the first
6. For each element in the first array
    a. For each element in the second array
        i. If we are at the end of the second array then insert that element
        ii. Otherwise if we find a value in the second array that is smaller than the examined value in the first
            1. Shift the values in the second array right
            2. Insert the value of the first array into the second
7. For each element in the second array print the values thus in printed order

Flow Chart


Prompt the user for the size of the
Size <- user Input
Size < 0
Yes
Program Ends
No
Create unsorted array "us"
i<-0
i <
No
Yes
Prompt user for a value
us[i] <- user input
i <- i+1
Create a second arrav "s" of
i<-0
i<us.lengt
i<- i+1
No
Print all elements of the
Yes
j<-0
j<us.lengt
j <- j+1
No
Yes
At the end of the sorted
Yes
s[j] <- us[i]
No
An element is
Shift all elements start at j to the right by
s[j] <- us[i]

Tests and Results

Test Case	Test Input	Result
A reversed order array	7 6 5 4 3 2 1	1 2 3 4 5 6 7
An in-order array	1 2 3 4 5 6 7	1 2 3 4 5 6 7
Random order array	2 4 3 1 6 5 7	1 2 3 4 5 6 7
A larger random order array	10 7 5 6 4 2 3 1 9 8	1 2 3 4 5 6 7 8 9 10
Negative size was entered	-1 for the size	The program quits

Problems Encountered

Using the incorrect index for unsorted array and the sorted array was a major problem. It was difficult to keep track of one to the other until I went back and renamed the arrays and the variables using a more descriptive but much longer identifier.

Index out of bounds exceptions came up as I accidentally was using <= the array's length for loop's Boolean expression instead of <. I've made a note that the last valid index in an array is the length-1.

Another index out of bound exception arose when I forgot to check to make sure the entered size of the array was non-negative. That was fixed by putting an if-statement that halted the program when that occurred.

Conclusions and Discussion

In this lab we sorted an array using the insertion sort algorithm. The way it worked was creating two arrays and then inserting the elements of the first array in the correct order in the second. Shifting required finding where the value belonged and shifting values over.

While this algorithm works, I think a better solution for sorting is sticking to bubble sort. First, bubble sort is much simpler to code as there isn't a need for shifting values. One simply swaps values until the correct location is found. Second bubble sort only requires one array. There is a version of insertion sort that uses one array, but its implementation seems to be a little trickier. I see the benefit of using insertion sort in some cases, but for small cases like this I would prefer to use bubble sort.

Additional Questions

1. Could insertion sort be implemented using only one array?

   Yes I believe it can. For instance if it were to examine a particular index, and then examine each element before to determine if it needs to be inserted elsewhere or remain where it is.

2. Given this array demonstrate each step of insertion sort as described in the lab. Use two arrays.

Index	0	1	2	3	4
Value	6	5	2	1	3

Index	0	1	2	3	4
Unsorted	6	5	2	1	3
Sorted	-	-	-	-	-

Index	0	1	2	3	4
Unsorted	6	5	2	1	3
Sorted	6	-	-	-	-

Index	0	1	2	3	4
Unsorted	6	5	2	1	3
Sorted	5	6	-	-	-

Index	0	1	2	3	4
Unsorted	6	5	2	1	3
Sorted	2	5	6	-	-

Index	0	1	2	3	4
Unsorted	6	5	2	1	3
Sorted	1	2	5	6	-

Index	0	1	2	3	4
Unsorted	6	5	2	1	3
Sorted	1	2	3	5	6