Bubble Sort Program in C

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-51

In the last lecture, we learnt what bubble sort is and how it is used to sort a linear collection of elements. Towards the end, we drew some conclusions regarding bubble sorting. Before we move on to the programming part, let's review some important notes concerning bubble sort.

1. Time Complexity of the bubble sort algorithm is O(n2).
2. It is a stable algorithm, because it preserves the order of equal elements.
3. It is not a recursive algorithm.
4. Bubble sort is not adaptive by default, but can be made adaptive by modifying the program. I'll show this part too.

Writing the program for implementing bubble sort is as easy as pie. I have attached the source code below. Follow it as we proceed.

Understanding the code snippet below:

1. The first step is to define an array of elements, such as integers or characters. I am taking an array of integers.

2. Define an integer variable for storing the size/length of the array.

3. Before we proceed to write the function for Bubble Sort, we would first make a function for displaying the contents of the array.

4. Create a void function *printArray,* and pass the address of the array and its length as its parameters. It doesn't take much to use a for loop to print the array elements. So, we'll skip that.

```
void printArray(int* A, int n){
    for (int i = 0; i < n; i++)
    {
        printf("%d ", A[i]);
    }
    printf("\n");
}
```

Code Snippet 1: Creating the *printArray* function

5. Create another void function *bubbleSort* and pass the address of the array and its length as its parameters. Now, create a *for* loop which would track the number of passes. If you recall, to sort an array of length 6, we made a total of 5 passes, which is obviously, 6-1. So, for length n, we would make (n-1) passes. So, make this loop run from 0 to (n-1). Inside this loop, make another *for* loop to track the index we are making a comparison at.

Can you decode the limit of this loop? It is obvious that we start from 0, but to which index? In the last lecture, we saw that with each pass, we reduced the size of the unsorted array by 1. In the first pass, we had the size of the unsorted array, 6, hence we made 5 comparisons. And for every subsequent pass, we made 4, 3, 2, and 1 comparison. Let i be the variable to store the pass we are at. Then the number of comparisons for ith pass would be (*n-i),* where n is the length of the array. Since we started from i=0 in the program, it would be *(n-i-1)* number of comparisons.

Inside this nested *for* loop, check if the jth element of the array is greater than the (j+1)th element. Here, j is the counter variable of the second *for* loop. So, if the jth element of the array is greater than the (j+1)th element, then swap their positions, since we want these to be sorted. Swapping needs you to define a temporary integer variable *temp*. Use it to swap the jth and the (j+1)th element.

And the array would itself get sorted. All we had to do was this.

```
void bubbleSort(int *A, int n){
    int temp;
    int isSorted = 0;
    for (int i = 0; i < n-1; i++) // For number of pass
    {
        printf("Working on pass number %d\n", i+1);
        for (int j = 0; j <n-1-i ; j++) // For comparison in each pass
        {
            if(A[j]>A[j+1]){
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
    }
}
```

Code Snippet 2: Creating the *bubbleSort* function

Modifying *bubbleSort* to make it adaptive

6. What would an adaptive bubble sort do? Once it detects that our array has already been sorted, it will not perform any more comparisons. So, just a single pass should do the job.

Therefore, the catch here is that the array is already sorted if we didn't have to perform any swapping during any of the passes. This is where we will stop making any more passes.

Create another void function *bubbleSortAdaptive,* and pass the address of the array and its length as its parameters. Create the same two loops, one nested in the other. First one runs from 0 to *n-1*, and another from 0 to *n-i-1*. We will make an integer variable *isSorted* which would hold 1 if our array is sorted and 0 otherwise. Make *isSorted* equal to 1 prior to starting any comparison in each pass. If any of our comparisons demands swapping of elements, we switch *isSorted* to 0.

At the end of each pass, check if the *isSorted* changed to 0. If it did, our array was not yet sorted; otherwise, end the comparison there itself, since our array was already sorted.

And this makes our bubble sort algorithm adaptive.

```
void bubbleSortAdaptive(int *A, int n){
    int temp;
    int isSorted = 0;
    for (int i = 0; i < n-1; i++) // For number of pass
    {
        printf("Working on pass number %d\n", i+1);
        isSorted = 1;
        for (int j = 0; j <n-1-i ; j++) // For comparison in each pass
        {
            if(A[j]>A[j+1]){
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
                isSorted = 0;
            }
        }
        if(isSorted){
            return;
        }
    }
}
```

Code Snippet 3: Creating the *bubbleSortAdaptive* function

Here is the whole source code:

```c
#include<stdio.h>

void printArray(int* A, int n){
    for (int i = 0; i < n; i++)
    {
        printf("%d ", A[i]);
    }
    printf("\n");
}
void bubbleSort(int *A, int n){
    int temp;
    int isSorted = 0;
    for (int i = 0; i < n-1; i++) // For number of pass
    {
        printf("Working on pass number %d\n", i+1);
        for (int j = 0; j <n-1-i ; j++) // For comparison in each pass
        {
            if(A[j]>A[j+1]){
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
    }
}

void bubbleSortAdaptive(int *A, int n){
    int temp;
    int isSorted = 0;
    for (int i = 0; i < n-1; i++) // For number of pass
    {
        printf("Working on pass number %d\n", i+1);
        isSorted = 1;
        for (int j = 0; j <n-1-i ; j++) // For comparison in each pass
        {
            if(A[j]>A[j+1]){
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
                isSorted = 0;
            }
        }
        if(isSorted){
            return;
        }
    }
}

int main(){
    // int A[] = {12, 54, 65, 7, 23, 9};
    int A[] = {1, 2, 5, 6, 12, 54, 625, 7, 23, 9, 987};
    // int A[] = {1, 2, 3, 4, 5, 6};
    int n = 11;
    printArray(A, n); // Printing the array before sorting
    bubbleSort(A, n); // Function to sort the array
    printArray(A, n); // Printing the array before sorting
    return 0;
}
```

Code Snippet 4: Program to implement the Bubble Sort Algorithm

Let us now check if our functions work well. Consider an array A of length 11.

```
    int A[] = {1, 2, 5, 6, 12, 54, 625, 7, 23, 9, 987};
    int n = 11;
    printArray(A, n);
    bubbleSort(A, n);
    printArray(A, n);
```

Code Snippet 5: Using the *bubbleSort* function

And the output we received was:

```
1 2 5 6 12 54 625 7 23 9 987
Working on pass number 1
Working on pass number 2
Working on pass number 3
Working on pass number 4
Working on pass number 5
Working on pass number 6
Working on pass number 7
Working on pass number 8
Working on pass number 9
Working on pass number 10
1 2 5 6 7 9 12 23 54 625 987
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above program

So, our array got sorted, and it made 10 passes as you can see. Let us now put the same array in the adaptive bubble sort function, and see if it still makes 10 comparisons.

```
    bubbleSortAdaptive(A, n);
    printArray(A, n);
```

Code Snippet 6: Using the *bubbleSortAdaptive* function

In fact, it only took one pass to detect it was already sorted.

```
Working on pass number 1
1 2 5 6 7 9 12 23 54 625 987
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 2: Output of the above program

So, this was all about the bubble sort algorithm. We started from the very basics and finished everything it could have. We are good to move to another sorting algorithm. Put in as much practice as you can.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube

channel to access it. See you all in the next tutorial where we'll learn another sorting algorithm called the **Insertion Sort**. Till then keep coding.