Show Course Contents

Overview Q&A Downloads Announcements

Insertion Sort in C Language (With Explanation)

In the last lecture, we learned what insertion sort is and how it is used to sort an array of elements by using the method of inserting an element in a sorted array. Finally, we enlisted the key characteristics of the algorithm. Before we move on to the programming part, let's review these characteristics of the insertion sort algorithm.

1. Time Complexity of the insertion sort algorithm is $O(n^2)$ in the worst case and $O(n)$ in the best case.
2. It is a stable algorithm since it preserves the order of equal elements.
3. It is not a recursive algorithm.
4. Insertion sort is adaptive by default and no extra effort is needed to make it adaptive. The time complexity itself gets reduced from $O(n^2)$ to $O(n)$ when the algorithm finds an array already sorted.

Having finished revising the insertion sort algorithm, let’s now move to the programming part. 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. I am defining 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 Insertion Sort, we would first make a function for displaying the contents of the array.
4. Since we did that already in the Bubble Sort lecture, we would skip that here. Rather just copy the function *printArray* from the previous programming lecture.

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

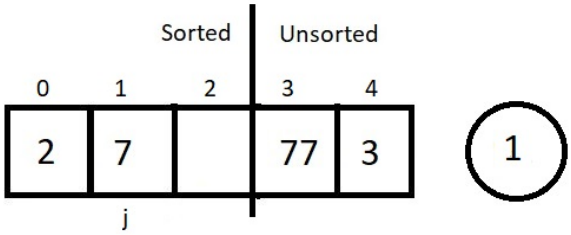
Code Snippet 1: Creating the *printArray* function

5. Create a void function *insertionSort* 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 5 using insertion sort algorithm, we made a total of 4 passes, which is obviously, 5-1. So, for an array of length n, we would make (n-1) passes. But this time the loop starts from the 1^{st} index, and not from the 0^{th} since the first element is sorted whatsoever. So, make this loop run from 1 to (n-1).
Inside this loop, collect the element at the index *i* in an integer variable *key.* This *key* is the element we would insert in the sorted subarray.

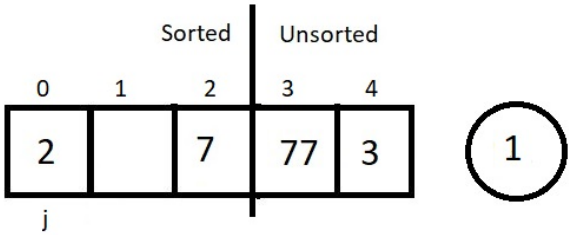
Create another index variable j, which would be used to iterate through the sorted subarray, and to find a perfect position for the *key*. The index variable *j* holds the value *i-1.*
Make a *while* loop run until either we finish through the sorted subarray and reach the last position, or else we find an index fit for the *key.* And until we come out of the loop, keep shifting the elements to their right and reduce *j* by 1. And once we come out, insert the *key* at the current value of *j+1.* And this would go on for *n-1*

passes.

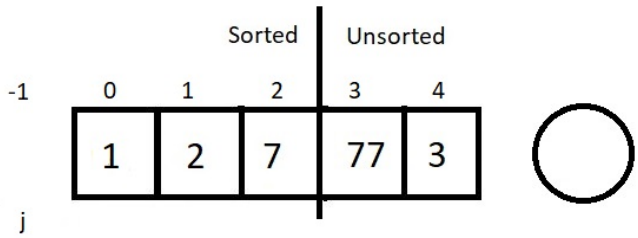
To understand the functioning of the above while loop, follow the illustrations below:



Here, i=2, and the element we have at index i is 1. No, this element needs to be given a position in the sorted subarray. So, *j = i - 1* which is 1. We run a while loop up until j becomes 0 or we find a position for the *key 1.* So, when j=1, we check if the element at the j^{th} index is smaller than the key or not. Since it's not, we shift it to the right and reduce j by 1.



And now j=0, and we have element 2 at the j^{th} index, and it is bigger than the key, hence, we shift even this. And then reducing *j* makes it equal to -1. So, we stop there itself. And insert the key at *j+1* which is at 0.



And at the end, we would receive a sorted array. All we had to do was this.

```
void insertionSort(int *A, int n){
    int key, j;
    // Loop for passes
    for (int i = 1; i <= n-1; i++)
    {
        key = A[i];
        j = i-1;
        // Loop for each pass
```

```
        while(j>=0 && A[j] > key){
            A[j+1] = A[j];
            j--;
        }
        A[j+1] = key;
    }
}
```

Code Snippet 2: Creating the *insertionSort* function

Here is the whole source code:

```
#include<stdio.h>

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

void insertionSort(int *A, int n){
    int key, j;
    // Loop for passes
    for (int i = 1; i <= n-1; i++)
    {
        key = A[i];
        j = i-1;
        // Loop for each pass
        while(j>=0 && A[j] > key){
            A[j+1] = A[j];
            j--;
        }
        A[j+1] = key;
```

```
    }
}

int main(){
   // -1   0    1   2   3   4   5
   //      12,| 54, 65, 07, 23, 09 --> i=1, key=54, j=0
   //      12,| 54, 65, 07, 23, 09 --> 1st pass done (i=1)!

   //      12, 54,| 65, 07, 23, 09 --> i=2, key=65, j=1
   //      12, 54,| 65, 07, 23, 09 --> 2nd pass done (i=2)!

   //      12, 54, 65,| 07, 23, 09 --> i=3, key=7, j=2
   //      12, 54, 65,| 65, 23, 09 --> i=3, key=7, j=1
   //      12, 54, 54,| 65, 23, 09 --> i=3, key=7, j=0
   //      12, 12, 54,| 65, 23, 09 --> i=3, key=7, j=-1
   //      07, 12, 54,| 65, 23, 09 --> i=3, key=7, j=-1--> 3rd pass

   // Fast forwarding and 4th and 5th pass will give:
   //      07, 12, 54, 65,| 23, 09 --> i=4, key=23, j=3
   //      07, 12, 23, 54,| 65, 09 --> After the 4th pass

   //      07, 12, 23, 54, 65,| 09 --> i=5, key=09, j=4
   //      07, 09, 12, 23, 54, 65| --> After the 5th pass

    int A[] = {12, 54, 65, 7, 23, 9};
    int n = 6;
    printArray(A, n);
    insertionSort(A, n);
    printArray(A, n);
    return 0;
}
```

Code Snippet 3: Program to implement the Insertion Sort Algorithm

Let us now check if our functions work well. Consider an array A of length 6.

```
int A[] = {12, 54, 65, 7, 23, 9};
int n = 6;
printArray(A, n);
insertionSort(A, n);
printArray(A, n);
```

Code Snippet 4: Using the *insertionSort* function

And the output we received was:

```
12 54 65 7 23 9
7 9 12 23 54 65
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above program

So, our array got sorted. Follow the dry run of the sorting process in the source commented. I have shown each of the 5 passes, and all the comparisons, and the shifting we did.

And, this was all about the insertion sort algorithm. We have finished writing the program for insertion sort, and have analyzed it as well. Once again, we are ready to move on to another sorting algorithm. Keep practicing everything we have learned so far by taking some random array of your own and sorting them using these algorithms. Do use the dry run method to gain confidence.
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 **Selection Sort**. Till then keep coding.

Previous Next

CodeWithHarry