MergeSort Source Code in C (Helpful Explanation)

Show Course Contents

Overview Q&A Downloads Announcements

MergeSort Source Code in C (Helpful Explanation)

In the last tutorial, we deeply covered the concepts behind the merge sort algorithm. We saw its implementation as well via some pseudocodes. We implemented the merge sort algorithm to sort a few arrays. I hope you did learn everything till here. If you couldn't, I recommend first going through the last lecture before proceeding to the programming part. Today, we'll solely look at the programming part of the merge sort algorithm in C.

Before we move on to the programming part, let's revise a few things

1. Whenever you are asked to sort an array using the merge sort algorithm, first divide the array until the size of each subarray becomes 1.
2. Now, since an array or subarray of size 1 is considered already sorted, we call our merge function, which merges these subarrays into bigger sorted subarrays.
3. And finally, you end up with your array fully sorted. Voila!

I will use an example from the last lecture to illustrate points 1 and 2.

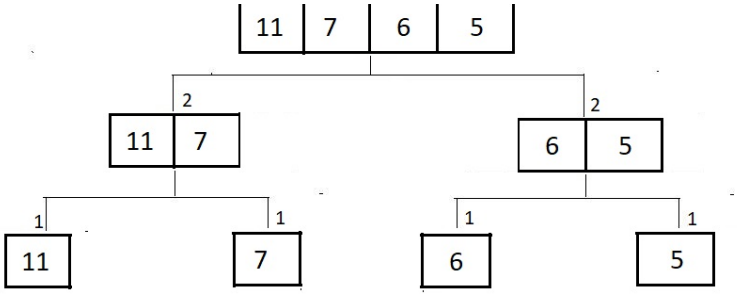

Figure: Point-1

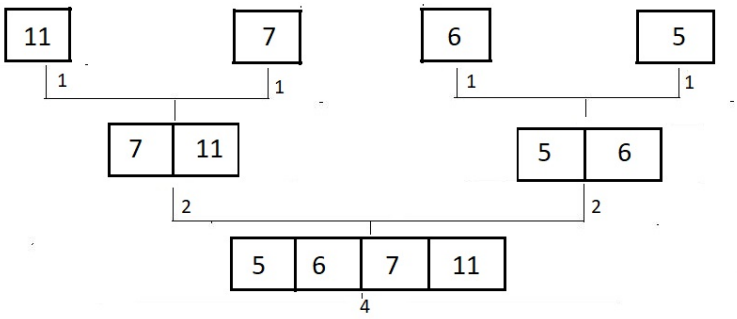

Figure: Point-2

That being all that we did yesterday, let us now move on to the programming part. I have attached the source code below. Follow it as we proceed.

Understanding the code snippet below:

1. Before we proceed with the functions *merge* and *mergeSort*, let's copy the *printArray* part in our current programs. Copying would save us some time. Having a print function helps a lot seeing the contents of the array before and after the sorting. Anyways, I have attached the snippet for *printArray* as well.

```
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

2. The next step is to define an array of elements. I'd rather copy that from our previous lecture. Then define an integer variable for storing the size/length of the array.

Creating the merge function:

3. This is just the merge function, whose only job is to merge two sorted arrays into a bigger sorted array. I'd recommend keeping the pseudo code with you for better understanding. Create a void function *merge* that takes the array and the integer indices *low, mid, and high* as its parameters. Create integer variables i,j, and k for iterating through the array A and an auxiliary array B. Create this integer array B of size, but for now, we would choose some larger size, say 100. Initialize *i* with *low, j* with *mid+1,* and *k* with *low* Here, i marks the current element of the first subarray of array A, and j marks the first element of the second subarray. And, k is the iterator for array B to insert the smaller of elements at indices i and j.
Run a while loop until either *i or j* or both reaches the threshold of their corresponding subarray. Inside the loop, see if the element at index i is smaller than the one at index j. If it is, insert element at index i in index k of array B i.e., B[k] = A[i] and increment both i and k by 1, else, insert element at index j in index k of array B i.e. B[k] = A[j] and increment both j and k by 1.
The above ends when either *i or j* or both reach its corresponding subarray's end. Now, run two separate while loops for inserting the remaining elements, if left, in both the subarrays. And this would finish filling all the elements in sorted order in array B. The only thing left is just to copy the sorted array back again to array A. And we are done.

```
void merge(int A[], int mid, int low, int high)
{
    int i, j, k, B[100];
    i = low;
    j = mid + 1;
```

```
    k = low;

    while (i <= mid && j <= high)
    {
        if (A[i] < A[j])
        {
            B[k] = A[i];
            i++;
            k++;
        }
        else
        {
            B[k] = A[j];
            j++;
            k++;
        }
    }
    while (i <= mid)
    {
        B[k] = A[i];
        k++;
        i++;
    }
    while (j <= high)
    {
        B[k] = A[j];
        k++;
        j++;
    }
    for (int i = low; i <= high; i++)
    {
        A[i] = B[i];
    }
}
```

Code Snippet 2: Creating the *merge* function

Creating the mergeSort function:

4. Create a void function *mergeSort* and pass the address of the array and the index variables *low and high* as its parameters. Here, the lower index would be 0 for the first time, and the higher index would be *length -1* for the first time.
We would recursively call this function only if *low* is less than *high; that* is, there are at least two elements in the subarray. Otherwise, we break off from the loop. Create an integer variable *mid* for holding the index of the mid element, which would be. Now recursively call the *mergeSort* function twice but with parameters changed to (low, *mid-1)* for the left subarray and (*mid+1*, high) for the right subarray. Applying mergeSort sorts the left half and the right half separately. This is where we would merge them back in the array. Call the merge function and pass the array, its index variables low, mid, and high. And this would return a sorted array.

```
void mergeSort(int A[], int low, int high){
    int mid;
    if(low<high){
        mid = (low + high) /2;
        mergeSort(A, low, mid);
        mergeSort(A, mid+1, high);
        merge(A, mid, low, high);
    }
}
```

Code Snippet 3: Creating the *mergeSort* 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 merge(int A[], int mid, int low, int high)
{
    int i, j, k, B[100];
    i = low;
    j = mid + 1;
    k = low;

    while (i <= mid && j <= high)
    {
        if (A[i] < A[j])
        {
            B[k] = A[i];
            i++;
            k++;
        }
        else
        {
            B[k] = A[j];
            j++;
            k++;
        }
    }
    while (i <= mid)
    {
        B[k] = A[i];
        k++;
        i++;
    }
```

```
    while (j <= high)
    {
        B[k] = A[j];
        k++;
        j++;
    }
    for (int i = low; i <= high; i++)
    {
        A[i] = B[i];
    }

}

void mergeSort(int A[], int low, int high){
    int mid;
    if(low<high){
        mid = (low + high) /2;
        mergeSort(A, low, mid);
        mergeSort(A, mid+1, high);
        merge(A, mid, low, high);
    }
}

int main()
{
    // int A[] = {9, 14, 4, 8, 7, 5, 6};
    int A[] = {9, 1, 4, 14, 4, 15, 6};
    int n = 7;
    printArray(A, n);
    mergeSort(A, 0, 6);
    printArray(A, n);
    return 0;
}
```

Code Snippet 4: Program to implement the Merge Sort Algorithm

Let us now check if our functions work well. Consider an array A of length 7.

```
int A[] = {9, 1, 4, 14, 4, 15, 6};
int n = 7;
printArray(A, n);
mergeSort(A, 0, 6);
printArray(A, n);
```

Code Snippet 5: Using the *mergeSort* function

And the output we received was:

```
9 1 4 14 4 15 6
1 4 4 6 9 14 15
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above program

So, our array got sorted. Practice sorting your own arrays using the merge sort method, and you'll feel confident about it. It's always possible to watch a segment again if you missed something. Make sure you understand everything.

And, this was all about the merge sort algorithm. The MergeSort function was all good, but the core segment was understanding the process of merging. It would be beneficial if you practiced these algorithms yourself. We have now completed a total of 5 sorting algorithms. Let me know if you could remember them all and apply them in sorting an array of your own.
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 our last of the sorting algorithm series called the **Count Sort Algorithm**. Till then, keep coding.

Previous

Next

CodeWithHarry