Computer Science >

Arrays and Sorting: Merge Sort (with C Program source code)

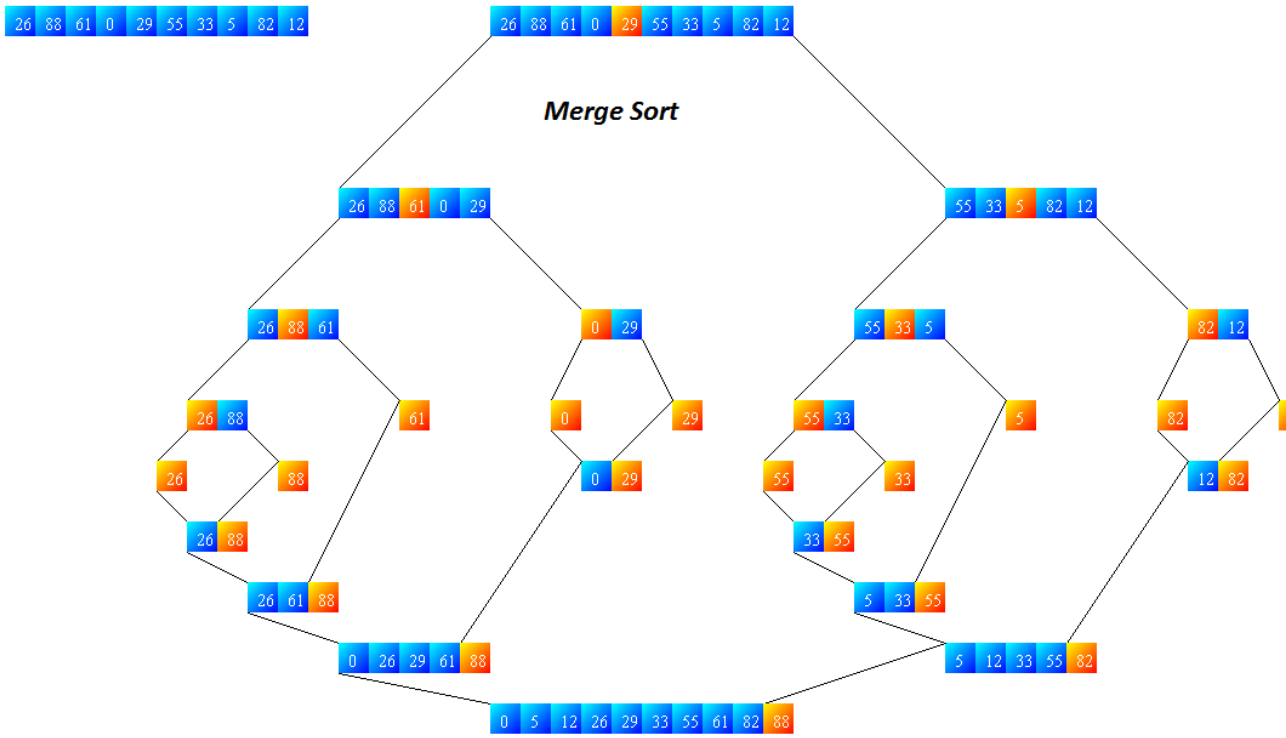
Quick Links to Various Sections on the Main Page

The Learning Point
Computer Science
Algorithms: An introduction to Dynamic Programming
CS - Data Structures and Algorithms
CS - Programming Interviews
CS - Miscellaneous C Programs
CS - Intro to DB Queries
Electrical Science and Engineering
Electrical Sci - DC Circuits
Electrical Sci - Digital Electronics
Mathematics
Mathematics - Linear Algebra
Mathematics - Geometry
Mathematics - Single Variable Calculus
Mathematics - Game Theory
Mathematics - Operations Research
Physics
Physics - Basic Mechanics
Physics - Electrostatics
Sitemap
Recent site activity

These are Quick Links to sections of the Main Page

2

To go through the C program / source-code, scroll down to the end of this page



Merge Sort

Merge sort is based on Divide and conquer method. It takes the list to be sorted and divide it in half to create two unsorted lists. The two unsorted lists are then sorted and merged to get a sorted list. The two unsorted lists are sorted by continually calling the merge-sort algorithm; we eventually get a list of size 1 which is already sorted. The two lists of size 1 are then merged.

Algorithm:

This is a divide and conquer algorithm. This works as follows –

1. Divide the input which we have to sort into two parts in the middle. Call it the left part and right part.
Example: Say the input is -10 32 45 -78 91 1 0 -16 then the left part will be -10 32 45 -
78 and the right part will be 91 1 0 6.
2. Sort each of them separately. Note that here sort does not mean to sort it using some other method. We use the same function recursively.
3. Then merge the two sorted parts.

Input the total number of elements that are there in an array (number_of_elements). Input the array (array[number_of_elements]). Then call the function MergeSort() to sort the input array. MergeSort() function sorts the array in the range [left,right] i.e. from index left to index right inclusive. Merge() function merges the two sorted parts. Sorted parts will be from [left, mid] and [mid+1, right]. After merging output the sorted array.

MergeSort() function:

It takes the array, left-most and right-most index of the array to be sorted as arguments. Middle index (mid) of the array is calculated as (left + right)/2. Check if (left<right) cause we have to sort only when left<right because when left=right it is anyhow sorted. Sort the left part by calling MergeSort() function again over the left part MergeSort(array,left,mid) and the right part by recursive call of MergeSort function as MergeSort(array,mid + 1, right). Lastly merge the two arrays using the Merge function.

Merge() function: (Explained in greater detail within the tutorial document)

It takes the array, left-most , middle and right-most index of the array to be merged as arguments.
Finally copy back the sorted array to the original array.

Properties:

Best case – When the array is already sorted O(nlogn).
Worst case – When the array is sorted in reverse order O(nlogn).
Average case – O(nlogn).
Extra space is required, so space complexity is O(n) for arrays and O(logn) for linked lists.

Merge Sort Visualization :

Here's a Java Applet Visualization which might help you get a clearer idea of what exactly happens in merge-sort.

Merge Sort Algorithm - Java Applet Visualization

Complete Tutorial :

Merge Sort

Merge sort is based on Divide and conquer method. It takes the list to be sorted and divide it in half to create two unsorted lists. The two unsorted lists are then sorted and merged to get a sorted list. The two unsorted lists are sorted by continually calling the merge-sort algorithm; we eventually get a list of size 1 which is already sorted. The two lists of size 1 are then merged.

Algorithm:

This is a divide and conquer algorithm. This works as follows –

1. Divide the input which we have to sort into two parts in the middle. Call it the left part and right part.
   Example: Say the input is -10 32 45 -78 91 1 0 -16 then the left part will be -10 32 45 -78 and the right part will be 91 1 0 6.
2. Sort each of them separately. Note that here sort does not mean to sort it using some other method. We use the same function recursively.
3. Then merge the two sorted parts.

Input the total number of elements that are there in an array (number_of_elements). Input the array (array[number_of_elements]). Then call the function MergeSort() to sort the input array. MergeSort() function sorts the array in the range [left,right] i.e. from index left to index right inclusive. Merge() function merges the two sorted parts. Sorted parts will be from [left, mid] and [mid+1, right]. After merging output the sorted array.

MergeSort() function:

It takes the array, left-most and right-most index of the array to be sorted as arguments.

Merge Sort - C Program Source Code

```
#include<stdio.h>
/*This is called Forward declaration of function */
void Merge(int * , int , int , int );
/* Logic: This is divide and conquer algorithm. This works as follows.
         (1) Divide the input which we have to sort into two parts in the middle. Call it the left part
              and right part.
             Example: Say the input is  -10 32 45 -78 91 1 0 -16 then the left part will be
             -10 32 45 -78 and the right part will be  91 1 0 6.
         (2) Sort Each of them seperately. Note that here sort does not mean to sort it using some other
              method. We already wrote fucntion to sort it. Use the same.
         (3) Then merge the two sorted parts.
*/
/*This function Sorts the array in the range [left,right].That is from index left to index right inclusive
 */
void MergeSort(int *array, int left, int right)
{
        int mid = (left+right)/2;
        /* We have to sort only when left<right because when left=right it is anyhow sorted*/
        if(left<right)
        {
```

```
                /* Sort the left part */
                MergeSort(array,left,mid);
                /* Sort the right part */
                MergeSort(array,mid+1,right);
                /* Merge the two sorted parts */
                Merge(array,left,mid,right);
        }
}
/* Merge functions merges the two sorted parts. Sorted parts will be from [left, mid] and [mid+1, right].
 */
void Merge(int *array, int left, int mid, int right)
{
        /*We need a Temporary array to store the new sorted part*/
        int tempArray[right-left+1];
        int pos=0,lpos = left,rpos = mid + 1;
        while(lpos <= mid && rpos <= right)
        {
                if(array[lpos] < array[rpos])
                {
                        tempArray[pos++] = array[lpos++];
                }
                else
                {
                        tempArray[pos++] = array[rpos++];
                }
        }
        while(lpos <= mid)  tempArray[pos++] = array[lpos++];
        while(rpos <= right)tempArray[pos++] = array[rpos++];
        int iter;
        /* Copy back the sorted array to the original array */
        for(iter = 0;iter < pos; iter++)
        {
                array[iter+left] = tempArray[iter];
        }
        return;
}
int main()
{
        int number_of_elements;
        scanf("%d",&number_of_elements);
        int array[number_of_elements];
        int iter;
        for(iter = 0;iter < number_of_elements;iter++)
        {
                scanf("%d",&array[iter]);
        }
        /* Calling this functions sorts the array */
        MergeSort(array,0,number_of_elements-1);
        for(iter = 0;iter < number_of_elements;iter++)
        {
                printf("%d ",array[iter]);
        }
        printf("\n");
        return 0;
}
```

Related Tutorials :

Bubble Sort	One of the most elementary sorting algorithms to implement - and also very inefficient. Runs in quadratic time. A good starting point to understand sorting in general, before moving on to more advanced	Insertion Sort	Another quadratic time sorting algorithm - an example of dynamic programming. An explanation and step through of how the algorithm works, as well as the source code for a C program which performs insertion sort.	Selection Sort	Another quadratic time sorting algorithm - an example of a greedy algorithm. An explanation and step through of how the algorithm works, as well as the source code for a C program which performs selection sort.	Shell Sort	An inefficient but interesting algorithm, the complexity of which is not exactly known.	Merge Sort	An example of a Divide and Conquer algorithm. Works in O(n log n) time. The memory complexity for this is a bit of a disadvantage.	Quick Sort	In the average case, this works in O(n log n) time. No additional memory overhead - so this is better than merge sort in this regard. A partition element is selected, the array is restructured such that all elements greater or less than	Heap Sort	Efficient sorting algorithm which runs in O(n log n) time. Uses the Heap data structure.	Binary Search Algorithm	Commonly used algorithm used to find the position of an element in a sorted array. Runs in O(log n) time.

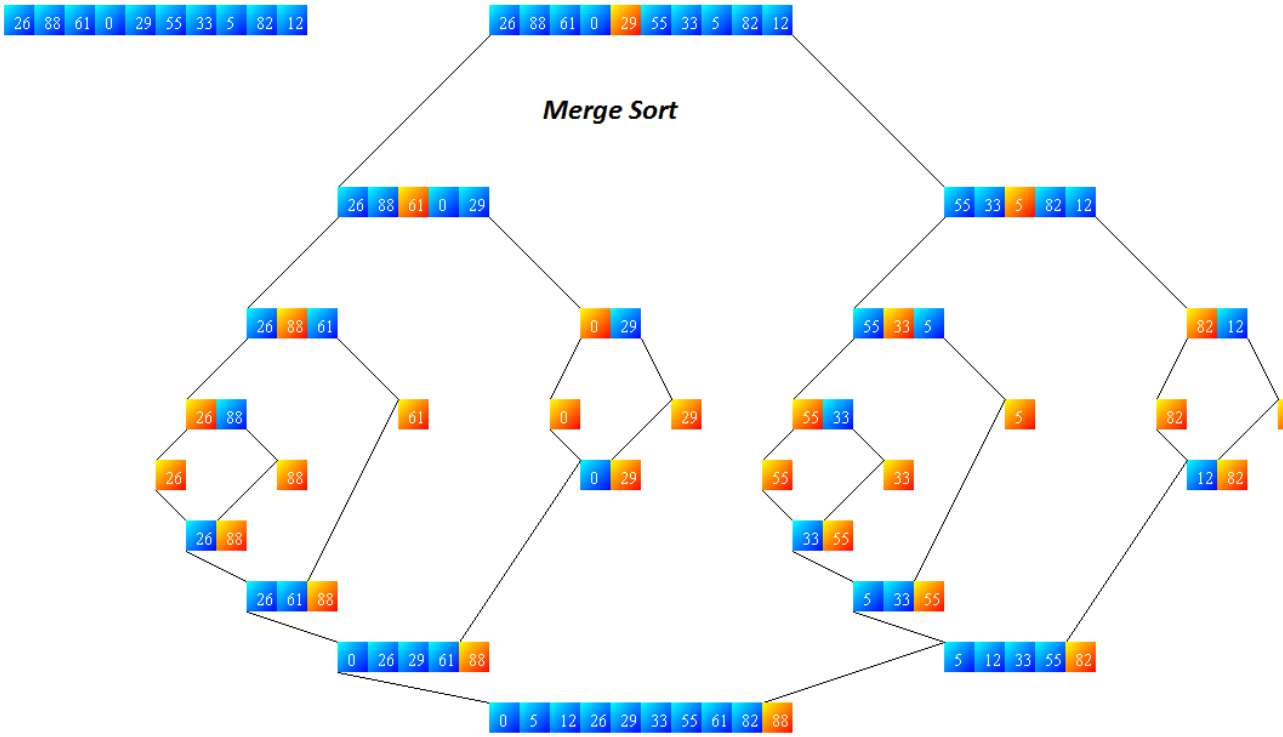
	techniques and algorithms. A general idea of how the algorithm works and a the code for a C program.										the partition are on opposite sides of the partition. These two parts of the array are then sorted recursively.				

Merge Sort Visualization :

Here's a Java Applet Visualization which might help you get a clearer idea of what exactly happens in merge-sort.

Merge Sort Algorithm - Java Applet Visualization

Click here or on the Image Below to check out a Merge Sort Visualization (in the form of a Java Applet)



Some Important Data Structures and Algorithms, at a glance:

Arrays : Popular Sorting and Searching Algorithms			
Bubble Sort	**Insertion Sort**	**Selection Sort**	**Shell Sort**

Merge Sort	Quick Sort	Heap Sort	Binary Search Algorithm
Basic Data Structures and Operations on them			
Stacks	Queues	Single Linked List	Double Linked List
Circular Linked List	1.		

Tree Data Structures			
Binary Search Trees	Heaps	Height Balanced Trees	
Graphs and Graph Algorithms			
Depth First Search	Breadth First Search	Minimum Spanning Trees: Kruskal Algorithm	Minumum Spanning Trees: Prim's Algorithm
Dijkstra Algorithm for Shortest Paths	Floyd Warshall Algorithm for Shortest Paths	Bellman Ford Algorithm	
Popular Algorithms in Dynamic Programming			
Dynamic Programming	Integer Knapsack problem	Matrix Chain Multiplication	Longest Common Subsequence
Greedy Algorithms			
Elementary cases : Fractional Knapsack Problem, Task Scheduling	Data Compression using Huffman Trees		

6 comments

Bob Owuor · Top Commenter · Stone Mountain, Georgia
good look
Reply · Like · October 22 at 8:28pm

Elia Atika Abu Bakar · Politeknik Tuanku Syed Sirajuddin (PTSS)
in c++ program?
Reply · Like · October 21 at 2:11am

Sakibul Mowla · Student at Department of Computer Science and
thanks much!
it's very helpful!
Reply · Like · October 3 at 12:17am

Wiliza Fermano · SKSU isulan campus
thaks
Reply · Like · October 14 at 1:44am

Anis Syafiqah · Programming at Politeknik Tuanku Sye
output dia cmnana?kena masukkan sendiri data?
Reply · 1 · Like · October 31 at 8:13am

View 2 more

Facebook social plugin