Merit-Quality-Excellence

Sukkur IBA University Khairpur Campus

Data Structures

LAB No: 06

Instructor: Marina Gul

Objective of Lab No. 06:
After performing lab 06, students will be able to:

- Sorting algorithm

Iterative Merge Sort:

In iterative merge sort, we do bottom up approach ie, start from 2 element sized array (we know that 1 element sized array is already sorted). Also the key point is that since we don't know how to divide the array exactly as in top down approach, where the 2 element sized array may be of size sequence 2,1,2,2,1…we in bottom up approach assume the array was divided exactly by powers of 2 (n/2,n/4….etc) for an array size of powers of 2, ex: n=2,4,8,16.
So for other input sizes such as 5, 7, 11 we will have remaining sublist that didn't go into the power of 2 width at each level as we keep on merging and go upwards, this unmerged sublist which is of size that is not exact power of 2, will remain isolated till the final merge.
To merge this unmerged list at final merge we need to force the mid to be at the start of unmerged list so that it is a candidate for merge.

The unmerged sublist element count that will be isolated until final merge call can be found out using the remainder (n % width). The final merge (when we have uneven lists) can be identified by (width>n/2).

Since width grows by powers of 2 when width == n/2 then it means the input was already of size in powers of 2, else if width < n/2 then we haven't reached final merge yet, so when width > n/2 we must be having pending unmerged uneven list hence we reset mid only in such case.

Read more

Exercise

1. Implement selection sort and insertion sort.

2. MergeSort using Iterative Approach: Implement the **IterativeMergeSort** a version of mergesort that does not use recursion. Your class should not contain any (static or non-static) functions except for the function mergeSort itself.

3. Prove that any comparison-based algorithm to sort four elements requires at least five comparisons.

Implement in the function **SortingFour()** a function for sorting four elements, which requires at most five comparisons.

4. **(Solve in NlogN):** We are given an array that contains N numbers. We want to determine if there are two numbers whose sum equals a given number K. For instance, if the input is 8, 4, 1, and 6, and K is 10, then the answer is yes (4 plus 6 is 10). A number n may appear more than once in the input array; in that case and only in that case the sum may have the form n + n.

Implement a function **TwoSum()** to solve this problem in **O(N log N)** time.

Hint: Sort the items first!