**Tutorial 07**

**Advanced Sort – Merge Sort**

1. Explain using an example of sorting algorithm, the divide and conquer strategy used in algorithm design.

**Solution:**

Divide and conquer is a strategy (also commonly described as an algorithmic paradigm) that is based on recursion and works by:

* Recursively breaking down a problem into two or more sub-problems, until these become simple enough to be solved directly.
* The solutions to the sub-problems are then combined to give a solution to the original problem.

For example, in the Merge Sort algorithm, an input list S with n elements is sorted in the following steps:

* **Divide**: partition S into two sublists S1 and S2 of about n/2 elements each
* **Recur**: recursively sort S1 and S2
* **Conquer**: merge S1 and S2 into a sorted list

Additional Notes:

(Reference: <https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms> )

In some other texts, divide and conquer is also explained to be consisting of three parts:

* **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
* **Conquer** the subproblems by solving them recursively. If they are small enough, solve the subproblems as base cases.
* **Combine** the solutions to the subproblems into the solution for the original problem.

1. Using Merge Sort binary tree, trace the execution of merge sort with the following list of numbers:
   1. 4, 7, 1, 8, 3, 2, 6, 5 (in ascending order)

Graphical user interface

Description automatically generated with medium confidence

* 1. 5, 2, 7, 8, 1, 4, 6, 3 (in descending order)

Diagram

Description automatically generated

1. In computer science, an in-place algorithm can be described as:

“an algorithm that does not need an extra space and produces an output in the same memory that contains the data by transforming the input ‘in-place’. However, a small constant extra space used for variables is allowed”

Are the sorting algorithms we have discussed so far, i.e. bubble sort, selection sort, insertion sort and merge sort, in-place algorithm?

**Solution:**

In-place sorting algorithms:

* Bubble Sort
* Selection Sort
* Insertion Sort

The implementation discussed during the lecture for Merge Sort is not an in-place algorithm. For example,

* List slicing creates a new list; and
* A new list is also created when we call the mergedSortedLists() function to merge two sorted sublists.

# Split the list and perform the recursive step

leftHalf = mergeSort(theList[:mid])

rightHalf = mergeSort(theList[mid:])

# Merge the two sorted sublists

newList = mergeSortedLists(leftHalf, rightHalf)

**return** newList

However, an in-place implementation of Merge Sort is possible e.g. by using index markers to specify a sublist of elements to create virtual sublists within the original physical list(e.g. in Binary Search, we use low & high index markers):

In addition, the mergeSortedLists() method will need to be modified as well to support in-place merging of the virtual sublists.

1. Given a sequence 𝑆 of 𝑛 values, each equal to 0 or 1 e.g. [1, 0, 0, 1, 1, 1, 0]. Describe an in-place method for sorting 𝑆.

[**NOTE**: Do not use the standard sorting algorithms e.g. bubble sort, selection sort, insertion sort etc. to answer this question.]

IT2153/IT2352/IT2553/IT2653/IT2852

Data Structures & Algorithms



**Solution:**

Imagine that we color the 0’s blue and the 1’s red. Start with a marker at the beginning of the list and one at the end of the list. While the first marker is at a blue element, continue incrementing its index. Likewise, when the second marker is at a red element and the second a blue element, swap the elements. Continue moving the markers and swapping until they meet. At this point, the sequence is ordered.

[**NOTE**: This approach is similar to the solution for Practical 06- Recursion, Question 3, where you were asked to re-arrange a sequence of integer values so that all the even values appear before all the odd values.]

If there are three colours in the sequence, we can order it by doing the above algorithm twice. In the first run, we will move one color to the front, swapping back elements of the other two colors. Then we can start at the end of the first run and swap the elements of the other two colors in exactly the same way as before. Only this time the first marker will begin where it stopped at the end of the first run.

***-- End of Tutorial --***

AY2020/21 S1 Page 1