Merge Sort (Recursive)

Merge Sort is based on divide and conquer technique. This algorithm divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(a, lb, mid, ub) is key process that assumes that a[lb..mid] and a[mid+1..ub] are sorted and merges the two sorted sub-arrays into one. Recursive Merge sort is also known as top down Merge sort.

Pseudocode:

```
MergeSort (a[],lb,ub)
Begin
        If(lb<ub)
        Begin
                mid=(lb+ub)/2 // find middle index of the array to divide it into two halves
                MergeSort (a,lb,mid) // Call Merge Sort recursively for first half
                MergeSort (a,mid+1,ub) // Call Merge Sort recursively for second half
                merge(a,lb,mid,ub) // Merge the two halves
        EndIf
End
```

Figure 1 shows the complete merge sort process for an example array {9, 7, 8, 3, 2, 1}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.

Whenever we divide a number into half in every step, it can be represented using a logarithmic function, which is $\log_2 n$ and the number of steps can be represented by $\text{floor}(\log_2 n) + 1$(at most) Also, we perform a single step operation to find out the middle of any subarray, i.e. O(1). And to merge the sub-arrays, made by dividing the original array of n elements, a running time of O(n) will be required. Hence the total time for MergeSort function will become $n(\log_2 n + 1)$, which gives us a time complexity of $O(n\log_2 n)$.

Merge sort is a stable sort. Since, the operation of merging is not in place, it is not an in place sort. Space complexity is O(n) because it needs O(n) extra space.

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

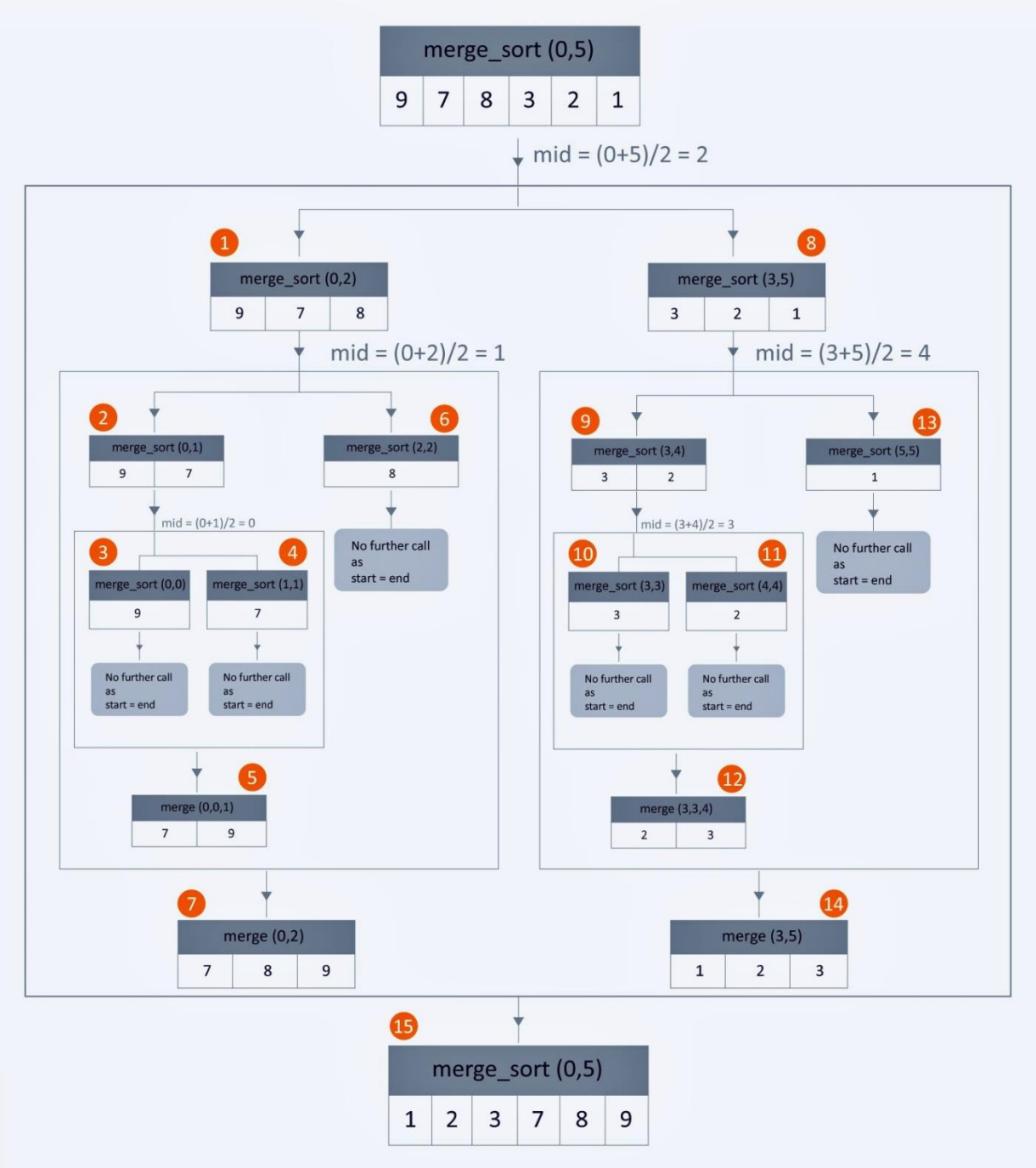

Figure 1: Steps of Merge Sort

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

Quick Sort (Recursive)

Quick sort is based on the divide-and-conquer approach based on the idea of choosing one element as a pivot element and partitioning the array around it such that: Left side of pivot contains all the elements that are less than the pivot element Right side contains all elements greater than the pivot.

There are many different versions of Quick sort that pick pivot in different ways.

- Always pick first element as pivot
- Always pick last element as pivot
- Pick a random element as pivot

The key process in quickSort is partition(). Target of partition() is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

Pseudocode:

```
QuickSort (a[],lb,ub)
Begin
        If(lb<ub)
        Begin
                p=partition(a,lb,ub) // p is partitioning index, a[p] will be at proper place after partition
                QuickSort (a,lb,p-1) // sorts the left partition of the pivot element recursively
                QuickSort (a,p+1,ub) // sorts the right partition of the pivot element  recursively
        EndIf
End
```

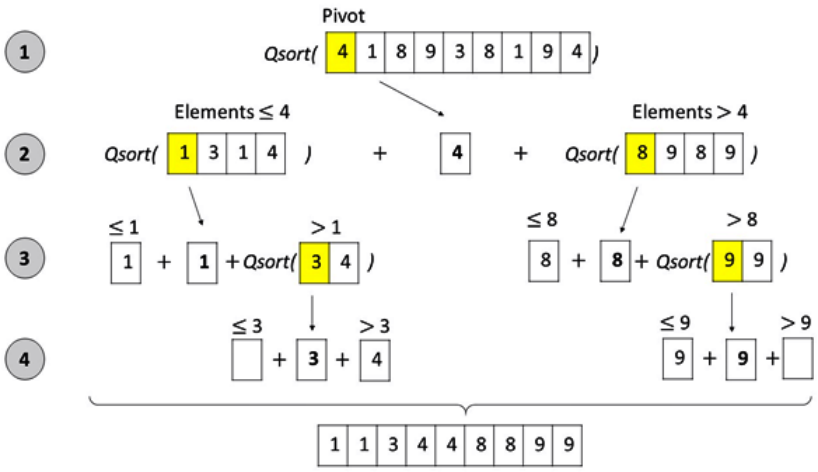

Figure 2: Steps of Quick Sort

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata

Quick sort is also known as partition exchange sort. Worst case time complexity of Quick sort is $O(n^2)$. Best case and average case time complexity of Quick sort is $O(n\log_2 n)$. Quick sort is not a stable sort. It qualifies as an in-place sorting algorithm as it uses extra stack space only for storing recursive function calls but not for manipulating the input. Quick sort with in place and unstable partitioning uses only constant additional space before making any recursive call. Quick sort must store a constant amount of information for each nested recursive call. Quick Sort is tail recursive, therefore tail call optimizations is done and it keeps the stack depth bounded by $O(\log_2 n)$. Hence, space complexity of in place, unstable Quick Sort (with tail recursion) is $O(\log_2 n)$. However, without tail recursion, in the worst case quicksort could make O(n) nested recursive calls and need O(n) auxiliary stack space.

Reference books and website links:

1. Data Structures Through C In Depth by S.K.Srivastava and Deepali Srivastava
2. Data Structures Using C by Reema Thareja
3. https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/tutorial/
4. https://www.studytonight.com/data-structures/merge-sort
5. https://www.geeksforgeeks.org/merge-sort/
6. https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/tutorial/
7. https://www.geeksforgeeks.org/quick-sort/
8. https://en.wikipedia.org/wiki/Quicksort#Space_complexity

Abhishek Dey
Assistant Professor
Department of Computer Science
Bethune College, Kolkata