Merge Sort

Arnab Ganguly, Assistant Professor
Department of Computer Science, University of Wisconsin – Whitewater
Data Structures (CS 223)

Simulation Weblink

https://visualgo.net/bn/sorting

1 Introduction

Merge-sort, like binary search, falls into the category of **divide-and-conquer** algorithms. The idea is that we will break a problem into smaller sub-problems, solve each smaller sub-problem, and use the answers to these sub-problems to build our answer for the larger problem.

We see that this is closely related to recursion, which uses a similar analogy – break a problem down to a base-case, whose solution is trivially known. Then build the solution back up from the base-case. Hence, it is no surprise that arguably the simplest and cleanest way to implement Merge-sort is via Recursion.

However, before getting into Merge-sort directly, we will digress a bit and look into another useful problem - *merging two sorted arrays*

2 Merging Two Sorted Arrays

The problem is simple to state – we have two sorted arrays A and B, containing $lenA$ and $lenB$ elements. Our task is to merge them and create a longer sorted array C having $lenC = lenA + lenB$ elements. Let us look at some examples.

If $A = [7]$ and $B = [5]$, then we know that $C = [5, 7]$. This is simply obtained by comparing 7 and 5 and placing the smaller of the two values first and then the larger one.

If $A = [7, 10]$ and $B = [5, 11, 16]$, then we know that $C = [5, 7, 10, 11, 16]$. Here, we first compare 7 and 5, and place 5 into C. Now compare 7 and 11, and place 7 into C. Now compare 10 and 11, and place 10 into C. We have reached the end of array A. Now, starting from 11, put the remaining elements of B into C.

Therefore, we see that the idea is – maintain three variables a, b, and c, for the respective arrays. Compare $A[a]$ with $B[b]$ and place smaller of the two into $C[c]$. Increment the variable (either a or b) for which the smaller value was copied into C and then increment c. Repeat this process as long as both a and b are smaller than the lengths of A and B respectively. Once the end of either A or B has been reached, copy the remaining elements of the other array into C.

See Algorithm on the next page on how to merge two sorted arrays.

The important thing to notice is the complexity. Note that in every iteration of the first while loop, we are advancing either a or b. At the end of this process, we will reach either the end of array A or B. Following this, we will advance the variable for the array, whose end we have not

Algorithm 1 Merging two sorted arrays

```
function MERGE(int A[ ], int B[ ], int lenA, int lenB)
    int $lenC = lenA + lenB$
    create an integer array $C$ of length $lenC$
    int $a = 0, b = 0, c = 0$;
    while ($a < lenA$ && $b < lenB$) do
        if ($A[a] < B[b]$) then
            copy $A[a]$ to $C[c]$
            increment $a$;
        else
            copy $B[b]$ to $C[c]$
            increment $b$;
        increment $c$;
    while ($a < lenA$) do
        copy $A[a]$ to $C[c]$
        increment both $a$ and $c$
    while ($b < lenB$) do
        copy $B[b]$ to $C[c]$
        increment both $b$ and $c$
    return $C$
```

reached. Since we never decrement a or b, and both of them start from 0, they are incremented a total of $lenA + lenB$ times. Hence the complexity is $O(lenA + lenB)$.

3 Recursive Algorithm for Merge Sort

Given our solution to the problem of merging two sorted arrays, merge sort is now rather straightforward. Simply break the array into roughly two equal parts. Recursively break again until we get single element arrays. Now each single element array is by default sorted. So use the algorithm for merging two sorted arrays and build the final solution.

Let us look at an example on the next page. We start of with $array = [8, 17, 1, 10, -8, 7, 3, -4]$. We break it into two equal parts based on $mid = (left+right)/2$. Initially, $left = 0$ and $right = 7$, so $mid = 3$. We consider the two parts separately – $array[left, mid]$ and $array[mid + 1, right]$. Now again divide these parts into two equal halves. Repeat the process recursively, until we reach single element arrays. At this point, the array only contains (logically) one element, which implies it is sorted. Starting from the single element arrays, we apply the algorithm for merging two sorted arrays, and keep on repeating the process until we get the sorted array back.

Thus, $array = [8, 17, 1, 10, -8, 7, 3, -4]$ is divided into two equal halves: $[8, 17, 1, 10]$ and $[-8, 7, 3, -4]$. The sorted arrays for these two halves are $[1, 8, 10, 17]$ and $[-8, -4, 3, 7]$ respectively. If we now merge these sorted arrays, we will get $[-8, -4, 1, 3, 7, 8, 10, 17]$.

Thus, merge-sort on whole array can be viewed as: merge-sort on the first half, merge-sort on the second half, and then merging the resultant sorted arrays. We will only break into two halves when we have more than one element; thus, the base case is $left \geq right$, where we stop.

3.1 Complexity

If we start with an array of length n, we keep on breaking it into two roughly equal parts, so we can break until $\log n$ levels. In each level, we will merge sorted arrays, whose total length is n. Hence, the complexity is $O(n \log n)$.

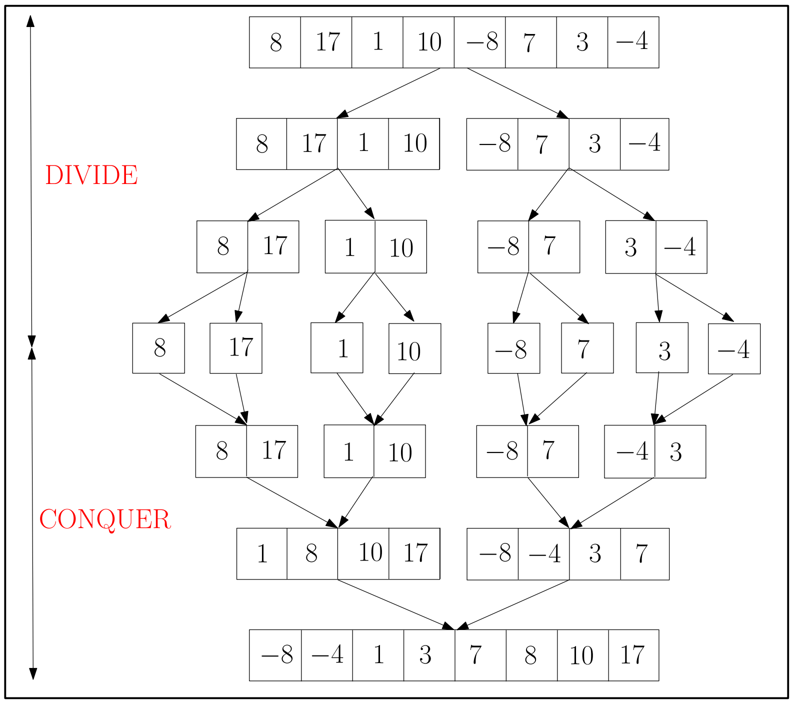

Figure 1: Illustration of the Merge-sort Algorithm

Algorithm 2 Merge-sort Algorithm

```
function MERGESORT(int array[ ], int left, int right)
    if ($left < right$) then
        int $mid = (left + right)/2$;
        MERGESORT(array, left, mid);
        MERGESORT(array, mid+1, right);
        create an integer array $mergedArray[]$ of length $right - left + 1$
        int $i = left, j = mid + 1, k = 0$;
        while ($i \leq mid$ && $j \leq right$) do
            if ($array[i] < array[j]$) then
                copy $array[i]$ to $mergedArray[k]$
                increment $i$;
            else
                copy $array[j]$ to $mergedArray[k]$
                increment $j$;
            increment $k$;
        while ($i \leq mid$) do
            copy $array[i]$ to $mergedArray[k]$
            increment both $i$ and $k$
        while ($j \leq right$) do
            copy $array[j]$ to $mergedArray[k]$
            increment both $j$ and $k$
        use a for-loop/while loop to copy element-by-element from $mergedArray[]$ into
    $array[left, right]$, i.e.,   $array[left] = mergedArray[0]$,
                                  $array[left + 1] = mergedArray[1]$,
                                  $array[left + 2] = mergedArray[2]$,
                                  $\vdots$
                                  $array[right] = mergedArray[right - left]$
```