Practice 13

Mayra Cristina Berrones Reyes

April 24, 2020

1 Introduction

If we can take any problem, divide it into smaller sub problems, solve these smaller sub problems and combine their solutions to find the solution for the original problem, then it becomes easier to solve the whole problem. This concept is known as **Divide and conquer**. This concept involves three steps [Gimel'farb, 2016]:

1. **Divide** the problem into multiple small problems.
2. **Conquer** the sub problems by solving them.
3. **Combine** the solutions of the subproblems to find the solution to the actual problem.

In Figure 1 we illustrate a diagram showing these three steps.

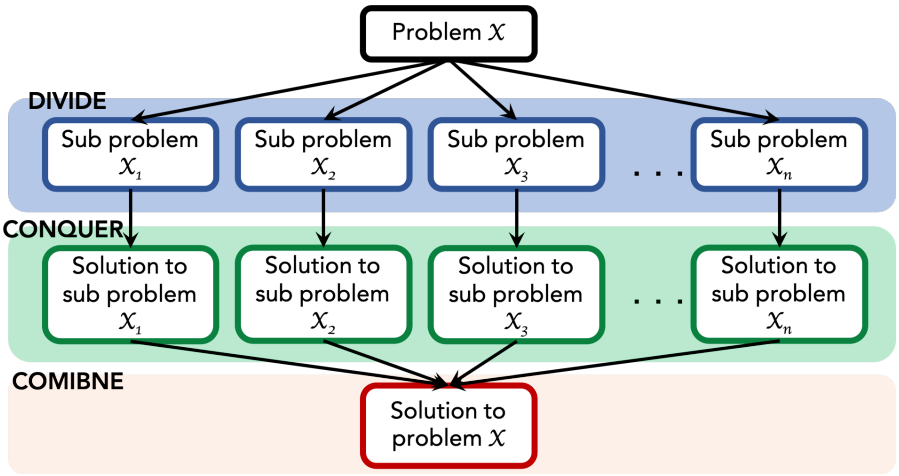

Figure 1: Diagram of divide and conquer steps.

2 Merge sort algorithm

Merge sort uses the divide and conquer rules, breaking a problem into smaller sub problems, the problem in this case being of sorting a given array of numbers. The steps for the merge sort algorithm are as follows:

1. We are going to take a variable x, and store the starting index of our array in it. Then we do the same but for the last index of our array and store it in a variable y.
2. With the formula $(x + y)/2$ we find the middle of the array and store the index as z.
3. Break the array into two sub arrays, from the index of x to z and form the index $z + 1$ to y.
4. We take each of the sub arrays and repeat steps 1 to 3 until the sub arrays are in single elements.
5. We start to merge the sub arrays.

In Figure 2 and 3 we have an example of the steps enumerated above.

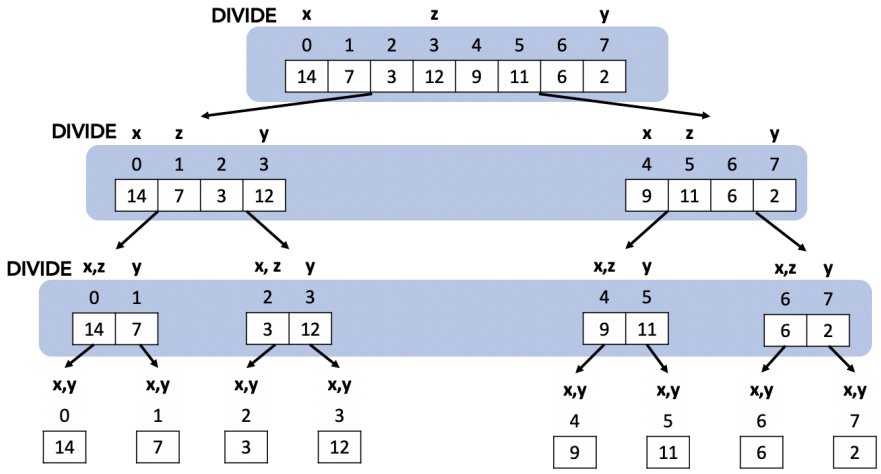

Figure 2: Example of the divide part on the merge sort algorithm.

2.1 Complexity analysis of Merge sort

The merge sort algorithm is very fast, and it is also a stable sort, which means that the equal elements in the array are ordered in the same order in the sorted list. The time complexity of the merge sort in all the 3 possible cases (Best,

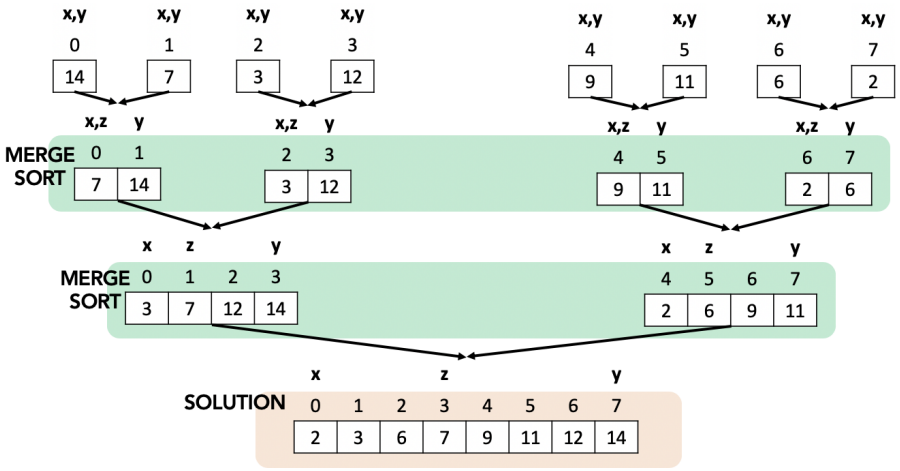

Figure 3: Example of the merge and sort part of the merge sort algorithm.

worst and average cases) is always $\mathcal{O}(n * \log\ n)$. This is because merge sort always divides the array in two halves and takes linear time to merge two halves. It also needs the same amount of additional space as the unsorted array, so this is not recomended for searches in large unsorted arrays. This is however, the best technique for sorting linked lists [Ahlawat, 2020].

Now we explain why the complexity of this algorithm is $\mathcal{O}(n * \log\ n)$ [G, 2016].

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \tag{1}$$

where:

- $T(n) =$ represents the runtime of the algorithm.
- n is the number of elements in the array.
- $2T\left(\frac{n}{2}\right)$ represents solving the sub problems through recursion.
- cn represents that merging them together will take constant time c of n.

In this case, if the value of n in Equation 1 is 1, then $T(n) = c$ which means that the runtime of the algorithm is constant time. With that we can put it as Equation 2.

$$\begin{cases} T(1) & = c, \\ T(n) & = 2T\left(\frac{n}{2}\right) + cn \end{cases} \tag{2}$$

Now, with the iterative application, we can get Equation 3

$$\begin{aligned}
T(n) &\leq 2T\left(\frac{n}{2}\right) + cn \\
T\left(\frac{n}{2}\right) &= 2T\left(\frac{n}{4}\right) + \frac{cn}{2} \\
T(n) &= 2\left(2T\left(\frac{n}{4}\right) + \frac{cn}{2}\right) + cn \\
T(n) &= 4T\left(\frac{n}{4}\right) + cn + cn \\
T(n) &= 4T\left(\frac{n}{4}\right) + 2cn \\
T\left(\frac{n}{4}\right) &= 2T\left(\frac{n}{8}\right) + \frac{cn}{4} \\
T(n) &= 4\left(2T\left(\frac{n}{8}\right) + \frac{cn}{4}\right) + 2cn \\
T(n) &= 8T\left(\frac{n}{8}\right) + 2cn + cn \\
T(n) &= 8T\left(\frac{n}{8}\right) + 3cn
\end{aligned} \tag{3}$$

Equation 3 represents the progression of the iterative application, and in the end, we begin to see a pattern, form which we can form Equation 4.

$$T(n) = 2^k * T\left(\frac{n}{2^k}\right) + k * cn \tag{4}$$

From Equation 4 we need to get to the $\mathcal{O}(n * \log n)$ time complexity, and for this, we need to establish a few things. So we have that

$$n = 2^k$$

so we can say that

$$\frac{n}{2^k} = \frac{2^k}{2^k} = 1$$

so finally we have

$$T\left(\frac{n}{2^k}\right) = T(1).$$

Now to get the log we have the formula

$$\log_{base} result = exponent$$

taking into consideration the $n = 2^k$ we then have

$$\log_2 n = k$$

Now we take Equation 4 and replace the elements

$$\begin{aligned} T(n) &= 2^k * T\left(\frac{n}{2^k}\right) + k * cn \\ T(n) &= 2^{\log_2 n} * T(1) + \log n * cn \\ T(n) &= 2^k * c + \log n * cn \\ T(n) &= n * c + \log n * cn \\ T(n) &= n + n(\log n) \end{aligned} \tag{5}$$

In Equation 5 we can see the few considerations above to follow the changes on the equation. In the end, we need to compare and weigh which side of that sum has a greater impact on the equation. We can conclude that $n(\log n)$ is a slightly bigger number than n so, then we have the complexity as $\mathcal{O}(n * \log n)$

3 Conclusions

In this case, I finally understood every equation, because I took it step by step, and whenever I was dubious about parts of the equation, I took the example from Figure 2 and 3 to see if everything matched. An example of this is when I tried to understand the changes made in Equation 5 when it incorporated the log function, I did not understand why the $n = 2^k$, but following the example, if I substitute n for the 8 elements in the example, and find the value of k with the $\log_2 n$ I got

$n = 8$
$\log_2 n = \log_2 8 = 3$

so $k = 3$ and finally, with $n = 2^k$ is the same as $8 = 2^3 = 8$. And then all the rest makes sense.

References

[Ahlawat, 2020] Ahlawat, A. (2020). Merge sort algorithm. Avialable: "https://www.studytonight.com/data-structures/merge-sort".

[G, 2016] G, C. (2016). Course of computer science algorithms. Avialable: "https://www.youtube.com/channel/UCPjbFOR74Ns0jTorAd3Fgdg".

[Gimel'farb, 2016] Gimel'farb, G. (2016). Algorithm mergesort complexity. Avialable: "https://www.cs.auckland.ac.nz/compsci220s1c/lectures/2016S1C/CS220-Lecture09.pdf".