Divide and conquer

Divide and conquer is an algorithm design paradigm, in which we divide a problem into smaller subproblems (often two subproblems) of the same type and then solve each subproblem independently.

The division is applied recursively until subproblems become simple enough for us to solve directly using a base case.

Finally, the solutions of all sub-problems are combined to get the solution for the original problem.

The idea of divide and conquer

Henry is a tomato sauce lover. So, this winter, he has decided to prepare five gallons of sauce for the upcoming year. To do so, he needs to blend $35kg$ of tomatoes.

However, Henry has a blender machine that can blend a maximum of $2kg$ of tomatoes at a time. How can he blend all $35kg$ of tomatoes? The solution is obvious: Henry has to take a portion of $2kg$ (or less) at a time and blend them.

This way, the task will require a minimum of 18 batches of blending. Finally, Henry should combine those blended tomatoes to obtain the final result: $35kg$ of blended tomatoes.

The divide and conquer algorithm paradigm works in the same way.

First, we have to divide our original problem into smaller similar subproblems. We take $2kg$ from $35kg$ of tomatoes in our example, whereas our problem remains the same — to blend them, i.e., the subproblem is similar to the original problem.
Next, we solve our subproblems: blend each smaller batch of tomatoes. Finally, we combine each subsolution by mixing all blended tomatoes.

Now imagine, what if Henry had four blender machines? He could have then blended

$$4*2=8kg$$

of tomatoes at a time.

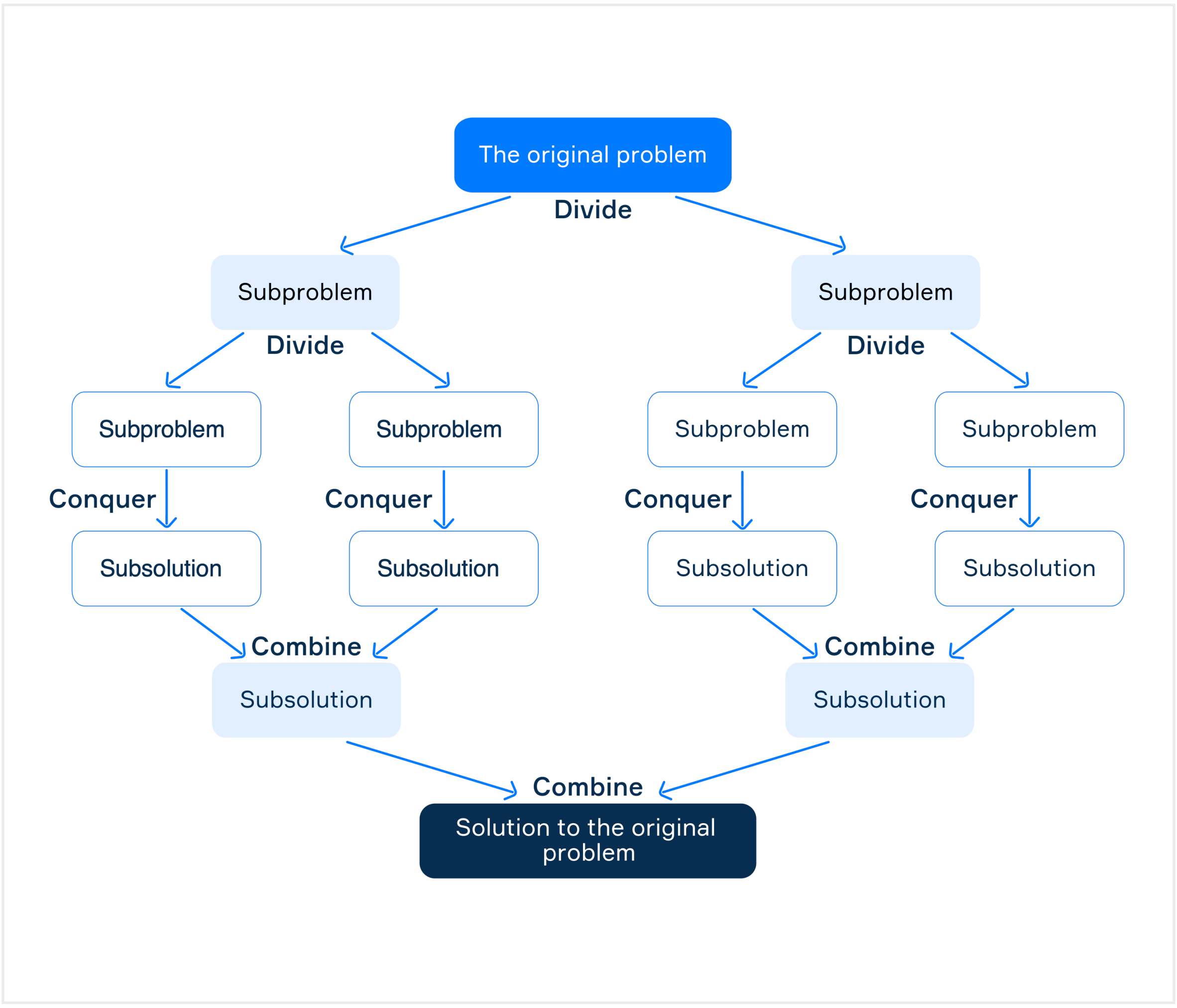
This idea is known as parallel computing in programming, which can be exploited using the divide and conquer algorithm paradigm.

The steps of a divide-and-conquer-based algorithm

A typical algorithm based on the divide and conquer paradigm consists of three steps:

1. Divide: split a problem into smaller sub-problems of the same type. Each sub-problem should represent a part of the original problem.
2. Conquer: recursively solve the sub-problems. If they are simple enough, solve them directly using base case conditions.
3. Combine: unite the solutions of the sub-problems to get the solution for the original problem.

The following picture shows the steps and their results:



The steps of a divide and conquer algorithm

In the above example, we first divide the original problem into two subproblems. Each subproblem is then divided into new subproblems until they are small enough for us to solve directly. After solving the smallest subproblems, we obtain subsolutions. We combine them to get subsolutions for more complex subproblems. The process continues until we obtain the solution for the original problem. As you can see, the process is recursive by nature.

A simple example: the sum of elements in an array

Let's consider how we can use the divide and conquer paradigm to calculate the sum of elements in an array
A={1,4,2,8,3,1,6}
. Note that there's a simpler and more efficient way to solve this problem; here, we apply the paradigm to get a better understanding of how it works. The procedure is the following:

1. We first define base case solutions. In our case, they are:
   - The sum of zero elements is 0, i.e., the sum of {} is 0.
   - The sum of a one-element array is the element itself. For example, the sum of {5} is 5.
2. Next, we divide our original problem into smaller and similar subproblems. In our example, we first separate {
   {1,4,2,8,3,1,6}
    into
   {1,4,2,8}
    and
   {3,1,6}
   . Now, let's take a look at our original problem and the subproblems:
   - Original problem: Find the sum of {

     {1,4,2,8,3,1,6}
     .
   - Subproblem 1: Find the sum of {

     {1,4,2,8}
     .
   - Subproblem 2: Find the sum of {

     {3,1,6}
     .
3. Take notice that the subproblems are the same as the original problem: finding the sum of an array. The only difference is that the array size is smaller.
4. Now, let us see how we can solve the subproblems. As the subproblems are similar to the original problem, we can consider the subproblems as an initial problem and repeat step 2
   . At some point, the subproblems will be small enough to be solved by the base solutions, and we will get subsolutions by following step 1
5. At this point, let us assume that we have solved subproblem 1 and subproblem 2

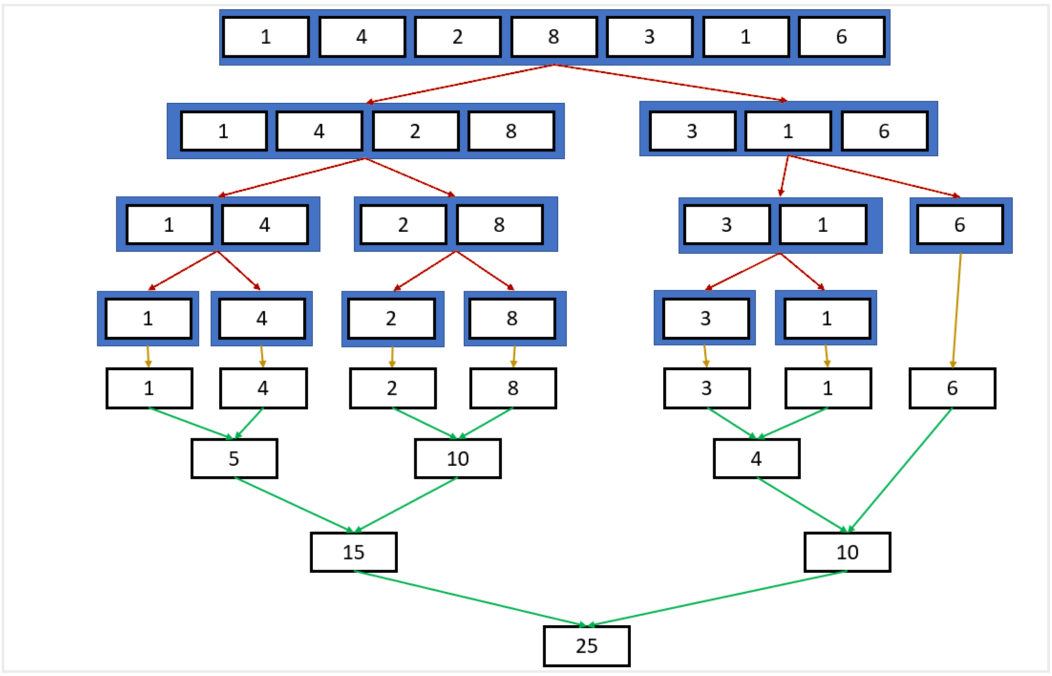
   - Subsolution 1

: The sum is 15
- Subsolution 2

  : The sum is 10

6. How can we combine these two solutions to find the original solution? For our example, we have to add subsolution 1
 and subsolution 2
. This is the rule of combination we have to follow for combining any two subsolutions. Eventually, we will reach the final solution by combining all subsolutions.

Here is an illustration of the steps mentioned above. In the picture, the blue-bordered arrays indicate problems and subproblems; the black-bordered arrays indicate subsolutions and solutions. The red arrows indicate dividing steps, the yellow arrows indicate conquering steps, and the green arrows indicate combining steps.



Here is the pseudocode of our program:

```
function calc_sum(array, left, right):
   // the sum of zero elements is 0
   if left == right then:
      return 0

   // the sum of one-element sub-array is the element
   if left == right-1 then:
      return array[left]

   // the index of the middle element to divide the array into two sub-arrays
   middle = (left + right) div 2
```

```
    // the sum of elements in the left subarray
    left_sum = calc_sum(array, left, middle)

    // the sum of elements in the right subarray
    right_sum = calc_sum(array, middle, right)

    // the sum of elements in the array
    return left_sum + right_sum
```

Complexity Analysis

We cannot calculate the divide and conquer algorithm's time complexity precisely, since it is an abstract algorithm designing technique. However, we can discuss its performance in terms of space and time.

Although the recursive nature of the algorithm costs space and time, the similar and smaller subproblems can be processed independently and simultaneously, which allows us to take advantage of multi-thread processors and parallel computing.

Moreover, solving problems using some defined base cases can be done within the cache memory. These features of the divide and conquer paradigm make it space-efficient and quite fast to perform.

Conclusion

The divide and conquer algorithm often provides us with a fairly simple solution to a complex problem, since it divides the problem into smaller subproblems. After that, we can combine the solutions of the smaller subproblems to obtain the final solution. There are many widely-used and efficient algorithms based on the divide and conquer paradigm: merge sort, quick sort, and others. If you decide to dive deep into algorithms, you will certainly come across this paradigm in the following topics.