Divide and Conquer Paradigm

Melissa Manley

California State University Northridge

Author's address: 18111 Nordhoff Street, Northridge, CA, 91330

The Divide and Conquer Paradigm. First off, a paradigm is a method of designing algorithms, a general approach to construct an efficient solution to a problem. The Divide and Conquer Paradigm is an algorithm design paradigm which uses this simple process: It Divides the problem into smaller sub-parts until these sub-parts become simple enough to be solved, and then the sub parts are solved recursively, and then the solutions to these sub-parts can be combined to give a solution to the original problem. In other words, this paradigm follows three basic steps; Step 1: Break down the problem into smaller sub-problems, Step 2: Solve these separate sub-problems, and Step 3: Combine the solutions of the sub-problems.

Sometimes, this Divide and Conquer name is given to algorithms that only reduce problems into one sub-problem. However, some authors only want to use this name when dealing with problems that will generate at least two or more sub-problems, and the name 'Decrease and Conquer' has been suggested when discussing the single-sub-problem group. The advantages of using the divide and conquer paradigm is that it allows us to solve difficult problems, it helps discover efficient algorithms, and they make efficient use of memory caches. However, with these algorithms also comes certain implementation issues, which include such things as slow recursion, Choosing the base cases, and sharing repeated sub-problems.

The time it takes for the divide and conquer algorithm to run is influenced by three things: (1) The number of sub-parts into which a problem is split, (2) The ratio of the size of the beginning problem to the size of the sub-problem, and (3) The number of steps required to divide and then combine the sub parts. So now that we have all of the basic concepts down about the process of divide and conquer, I will explain a few different examples of algorithms that are based on the Divide and Conquer paradigm, such as the Binary search, Merge Sort and Quick Sort algorithms, and the Cooley-Tukey fast Fourier transform algorithm.

Categories and Subject Descriptors: D.2.7 [**Software Engineering**]: Distribution andMaintenance—*documentation*

General Terms: Documentation, Languages

Additional Key Words and Phrases: Divide and Conquer

1. INTRODUCTION

Have you ever taken time out of your day to stop and think about how your toughest tasks can be simplified so they are easier to solve? Mail is just one of the many examples of a mathematical concept encountered each day. For instance, everyone gets mail delivered to their house daily, but many do not stop to think about the process behind the sorting of mail. The process of sorting the letters by region, and then sorting them even more by sub-regions until they are in small enough bags to be hand delivered to certain blocks by different mailmen is an everyday example of a process called dividing and conquering.

In mathematics and computer science, the concept of Divide and Conquer is an algorithm-design paradigm. By Definition, it is the process of dividing a complex problem into two or more sub-problems of the same type until the sub-problems are easy enough to be solved directly and recursively, and then these solutions are combined to give a solution to the initial problem. So, to clarify, the main steps of the divide and conquer paradigm are as follows:

(1) Divide the problem into two (or more) sub-problems

(2) Conquer these sub problems recursively

(3) Combine the solutions to the sub problems in order to get a solution to the original problem.

This technique of dividing and conquering is the foundation of efficient algorithms for many different problems, including sorting problems and the discrete Fourier transform. But when exactly was the divide and conquer paradigm first used?

2. HISTORY

This paradigm had certain instances of use dating back throughout history, with one of the first to use it being Gauss. So in a way, this idea of divide and conquer can sometimes be accredited to Gauss. He is one of the earliest mathematicians who, with his description of the now-known Cooley-Tukey FFT algorithm in 1805, even though he did not thoroughly analyze it (and these FFT algorithms became popular only once they had been rediscovered at a later date), had used the idea of divide and conquer using several sub-problems. Another mathematician who contributed early examples of this algorithm was Anatolii Karatsuba, who in 1960 proposed an algorithm based on the basic core ideas of the paradigm, which "could multiply two n-digit numbers in $O(n^{log_2 3})$ operations" [Wikipedia]. The ideas presented by Gauss were similar to those found in the proposal by Anatolii Karatsuba, and shows that there was no one mathematician who founded this process, but rather it was from many mathematicians over hundreds of years using early examples of the process that make up the history of this paradigm. The history of this process, in a way, can be thought of as still continuing to be made today because of the advantages that these early examples had demonstrated [Wikipedia].

3. ADVANTAGES

The first, and probably most recognizable benefit of the divide and conquer paradigm is the fact that it allows us to solve difficult and often impossible looking problems, such as the Tower of Hanoi, which is a mathematical game or puzzle. Being given a difficult problem can often be discouraging if there is no idea how to go about solving it. However, with the divide and conquer method, it reduces the degree of difficulty since it divides the problem into sub problems that are easily solvable, and usually runs faster than other algorithms would. Another advantage to this paradigm is that it often plays a part in finding other efficient algorithms, and in fact it was the central role in finding the quick sort and merge sort algorithms. It also uses memory caches effectively. The reason for this is the fact that when the sub problems become simple enough, they can be solved within a cache, without having to access the slower main memory, which saves time and makes the algorithm more efficient. And in some cases, it can even produce more precise outcomes in computations with rounded arithmetic than iterative methods would. Packaged with all of these advantages, however, are some weaknesses in the process [Wikipedia].

4. DISADVANTAGES

One of the most common issues with this sort of algorithm is the fact that the recursion is slow, which in some cases outweighs any advantages of this divide and conquer process. Another concern with it is the fact that sometimes it can become more complicated than a basic iterative approach, especially in cases with a large n. In other words, if someone wanted to add a large amount of numbers together, if they just create a simple loop to add them together, it would turn out to be a much simpler approach than it would be to divide the numbers up into two groups, add these groups recursively, and then add the sums of the two groups together. Another downfall is that sometimes once the problem is broken down into sub problems, the same sub problem can occur many times. In cases like these, it can often be easier to identify and save the solution to the repeated sub problem, which is commonly referred to as memoization. And the last recognizable implementation issue is that these algorithms can be carried out by a non-recursive program that will store the different sub problems in things called explicit stacks, which gives more freedom in deciding just which order the sub problems should be solved. These implementation issues do not make this process a bad decision when it comes to solving difficult problems, but rather this paradigm is the basis of many frequently used algorithms [Wikipedia].

5. COMMON TYPES OF DIVIDE AND CONQUER ALGORITHMS

There are many different specific types of Divide and Conquer algorithms. Some of these algorithms only use one sub problem, and can sometimes be referred to as 'Decrease and Conquer', however they are still just as important as the examples of this divide and conquer process that contain two sub problems. The Binary Search, the Merge Sort Algorithm, the Quick sort algorithm, Matrix Multiplication, and the fast Fourier transform are all examples of this Divide and Conquer paradigm that divide the problems in order to solve them. So, lets take a closer look at some of these algorithms that use this divide and conquer method.

6. BINARY SEARCH

What if an algorithm has to select between two very different alternatives at each step? Well there are plenty of algorithms that have to do this, and one of the most essential is called the Binary Search. Sometimes referred to as the "Ultimate divide-and-conquer algorithm", Binary Search uses a simple process to decide on using either one half of a data set, or the other [Dasgupta, 2006]. For example, if trying to find a key k in a set of keys containing data z[0,1,...,n-1], then this Search would compare k with $z[n/2]$, and then recurse on either the first half or the second half of the data, where which half is recursed is determined by the result of the comparison of k [Dasgupta, 2006]. This seems a bit on the confusing side just being explained in words, so lets take a look at an example of this simplest form of the divide and conquer paradigm:

Suppose one has a directory containing a set of names and a telephone number associated with each name. The directory is sorted by alphabetical order of names, and contains n entries which are stored in two arrays:

names(1...n); numbers(1...n)

Given a name and the value of n the problem is to find the number associated with the name. We assume that any given input name actually does occur in the directory, and then the divide and conquer algorithm is based on the following observations:

Given a name, X, occurs in the middle place of the name arrays, or X occurs in the first half of the name arrays(U), or X occurs in the second half of the names array(L). This means that U and L are only true if:

X come before or after the name stored in the middle place [Dunne].

These Observations then lead to the following algorithm:

function search (X : name; start, finish : integer)

```
return integer is middle : integer;
begin
middle := (start+finish)/2;
if names(middle)=x then
return numbers(middle);
elsif $X$ <names(middle) then
return search(X,start,middle-1);
else – $X$ >names(middle)
return search(X,middle+1,finish);
end if;
end search;
```

[Dunne]. This example is a great one to help to understand not only just how binary search works, but also how to set up a general type of algorithm for these types of problems. Binary Search is one of the most common types of the divide and conquer method, but another just as simple example are the sorting algorithms.

7. SORTING ALGORITHMS BASED ON D AND C

The Merge Sort and Quick Sort algorithms use the same procedures as the Binary search, since they are also based on the divide and conquer paradigm. What Merge Sort does is it will split the data or list that is given into two halves, then it will recursively sort each half, and then merge the two sub problems or sub lists once they have been sorted. This algorithm tends to be shorter, and in a general form, would look something like this:

```
Function mergesort ($a[1...n]$)
```

```
Input: An array of numbers $a[1...n]$
Output: A sorted version of this array
If $n > 1$:
    return merge(mergesort($a[1...[n/2]]$),
        mergesort ($a[[n/2] + 1...n]$))
else:
    return $a$
```

[Dasgupta, 2006]. The Merge Sort algorithms are a simple idea of the paradigm since you divide the list, and in this way conquer that list by sorting.

The Quick Sort Algorithm is the same thing as the merge sort algorithm, except it is a random sorting algorithm. That means that in this case, you divide the data by picking a random element, call it s, and make this data into three groups:

(1) A-elements less than s

(2) B-elements equal to s

(3) C-elements greater than s

Once these three subgroups are created, then the next step is to recursively sort A and C, and then to conquer the problem by combing A, B, and C back together [Amato, 2008]. So what about more complicated examples where there is more than one sub problem?

8. FAST FOURIER TRANSFORM

Well in the case of the fast Fourier transform, you are dealing with polynomials, which already makes it a more complicated example. In order to solve this sort of problem by the divide and conquer method, the first step would be to divide the polynomial into the even and odd coefficients, which would result in something along the lines of this:

$$A(x) = A_e(x^2) + xA_0(x^2)$$

In this equation, the first value $A_e(x^2)$ is the polynomial with even coefficients and the second value $xA_0(x^2)$ is the polynomial with odd coefficients. Then, to evaluate A(x) at n terms, it is easy to just evaluate $A_e(x)$ and $A_0(x)$ and n/2 terms. Then the next step, like all others in the divide and conquer process, would be to recursively solve this. You only run into problems with this type of problem when the recursion goes past the top level, because then the negative values need to be taken into consideration, which add complex numbers into the solutions and makes it seem like one big complicated mess. This being one of the more complex examples of divide and conquer, it makes it more evident that this paradigm can even make these very difficult problems seem way simpler. The divide and conquer paradigm makes a lot of problems easier to solve, but does the running time of these algorithms defeat the purpose, or actually make using the algorithm worth it? [Dasgupta, 2006].

9. RUNNING TIMES

For these Divide and Conquer algorithms, the running time is influenced by three standard things:

(1) The number of sub-instances the problem is split into.

(2) The ratio of the original problem size to sub-problem size.

(3) The number of steps required to divide and conquer, expressed as a function of the input size, n.

[Dunne]. In this case, if you have a divide and conquer algorithm that produces α sub problems, and each sub problem is of size n/β, then Tp(n) represents the number of steps taken by the algorithm on anything of size n. Then, in general, when you have α and β stand for constants, then the Time function

$$T(n) = \alpha T(n/\beta) + O(n^{\gamma})$$

has the solution:

$$O(n^{\gamma}) \text{ if } \alpha < \beta^{\gamma}$$
$$T(n) = O(n^{\gamma} logn) \text{ if } \alpha = \beta^{\gamma}$$
$$O(n^{log^{-}\beta\alpha)} \text{ if } \alpha > \beta^{\gamma}$$

[Dunne]. And in most cases of the divide and conquer paradigm, it is most often more time efficient than other paradigms that have the same types of algorithms. For example, lets compare some of the different types of sorting algorithms.

Algorithm	Time	Notes
Selection-sort	$O(n^2)$	slow
Insertion-sort	$O(n^2)$	slow
Heap-sort	$O(nlogn)$	fast
Quick-sort	$O(nlogn)$	fastest
Merge-Sort	$O(nlogn)$	fast

[Amato, 2008]. This table compares five different sorting algorithms: the selection sort, insertion sort, heap sort, quick sort and merge sort. Their running times, as seen in the middle column of the table above, along with the notes contained in the last column of the above table, can be used together to help conclude that the divide and conquer algorithms, like the merge sort and quick sort, can be proved as much more time efficient algorithms, and faster than other methods turn out to be.

10. CONCLUSION

In conclusion, the divide and conquer paradigm has many beneficial qualities that make it a very important algorithm-design paradigm. It involves only three simple steps to make complex problems easier, and those steps are to:

(1) Divide the complex problem into two or more sub-problems

(2) Recursively solve the sub problems

(3) Conquer by combining the solutions to the sub problems to get a solution to the original problem

It enables us to solve difficult problems, helps to find other efficient algorithms, and is sometimes the better choice than an iterative approach would be. One of the only downfalls to this paradigm is the fact that recursion is slow, and this fact alone can almost negate all the other advantages of this process. Merge Sort and Quick Sort are both sorting algorithms that are based on this process of the divide and conquer paradigm. Other examples that are based on this process are matrix multiplication, pair wise summation, and the fast Fourier transform. Once a closer look is taken at the running times of these different divide and conquer algorithms, and the paradigm in general, you will see that this paradigm's running time will beat out many of the other algorithm running times proving to be the most time efficient and the most practical choice.

This basic idea of dividing and conquering can be seen in everyday life. From people going to the market with a list of things to get, and then dividing that list into certain items for each person to get, then putting them into the same cart and checking out is a very simple example of the concept of the paradigm. Another example most of us will run into is a list of errands. Well a big list of errands that are scattered all over town may seem like a daunting task, but if the person was to divide their list by where each place is located, and then sort each location's places for where to go first and second and so on, then that person could conquer their errand list without feeling overworked. In reality it just makes everything in life simpler if you try and apply the process of divide and conquer. So in other words, wouldn't it only seem logical if you were to use the divide and conquer algorithm in order to make all of the complex problems in mathematics that much simpler and easier to solve? Overall, the point is simple. The divide and conquer algorithm design paradigm is a definite advantage to the world of mathematics and in this way, is a very good concept to know about and understand.

REFERENCES

Amato, Nancy. Sorting. slide 25 and 40 (2008)
http://parasol.cs.tamu.edu/~amato/Courses/221/lectures/Ch10.Sorting.pdf

Dasgupta, S.,and Papadimitriou, C.H., and Vazirani, U.V. 2006. Algorithms.pg 60,71-72. http://www.cs.berkeley.edu/~vazirani/algorithms/chap2.pdf

Dunne, P.E. Divide-and-conquer.
http://www.csc.liv.ac.uk/~ped/teachadmin/algor/d_and_c.html

Divide and conquer algorithm. Wikipedia.
http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm