Designing Algorithms

It is important to be able to analyze an algorithm to determine its performance and thereby assess the efficiency of the algorithm on a given problem. The other side of the problem is designing the algorithm in the first place. In keeping with the focus of analysis, we will explore techniques for designing algorithms that maximize efficiency, and we will back that design up with the analysis to prove the expected bounds are met. Throughout the remainder of the course, we will consider several different algorithm designs as case studies to provide examples of good algorithms. We will also consider various design methods and apply those methods in designing our own algorithms.

In this unit, we will have considered one method for algorithm design already—the incremental method. In the incremental method, we tackle a problem by considering each of the data items one at a time and process that data item until finished. We found, for the sorting procedure, this incremental approach was not efficient; however, we may find that such incremental methods are satisfactory, depending on the requirements of the problem being solved. Here, we focus on an additional design method as an example of one that frequently yields efficient solutions. That method is referred to as the divide-and-conquer method.

Fundamentally, the divide-and-conquer method is a recursive method. In other words, as a method, we construct an algorithm, that solves a problem by applying itself to smaller and smaller problems until it reaches a point where solution is trivial. The standard form of a recursive algorithm is as follows:

Algorithm 1 Recursive Functions

```
RecursiveFunction(argList)
  if problem is small enough then
    solution ← SimpleSolution(argList)
  else
    reduce problem size through division, processing, etc.
    solution ← RecursiveFunction(argList for subproblem)
  return solution
```

Given this template, the divide-and-conquer design method constructs such recursive algorithms by following three steps:

1. **Divide**: Split the problem into two or more smaller subproblems.
2. **Conquer**: Solve each of the subproblems recursively (or directly if the problem is simple enough).
3. **Combine**: Take the solutions to the subproblems and combine them into a solution for the current problem.