# Decrease and Conquer

As divide-and-conquer approach is already discussed, which include following steps:
**Divide** the problem into a number of subproblems that are smaller instances of the same problem.

**Conquer** the sub problems by solving them recursively. If the subproblem sizes are small enough, however, just solve the sub problems in a straightforward manner.

**Combine** the solutions to the sub problems into the solution for the original problem.
Similarly, the approach decrease-and-conquer works, it also include following steps:

**Decrease** or reduce problem instance to smaller instance of the same problem and extend solution.

**Conquer** the problem by solving smaller instance of the problem.

**Extend** solution of smaller instance to obtain solution to original problem .
Basic idea of the decrease-and-conquer technique is based on exploiting the relationship between a solution to a given instance of a problem and a solution to its smaller instance. This approach is also known as incremental or inductive approach.

**"Divide-and-Conquer" vs "Decrease-and-Conquer":**

As per Wikipedia, some authors consider that the name "divide and conquer" should be used only when each problem may generate two or more subproblems. The name decrease and conquer has been proposed instead for the single-subproblem class. According to this definition, Merge Sort and Quick Sort comes under divide and conquer (because there are 2 sub-problems) and Binary Search comes under decrease and conquer (because there is one sub-problem).

**Implementations of Decrease and Conquer :**

This approach can be either implemented as top-down or bottom-up.

**Top-down approach :** It always leads to the recursive implementation of the problem.
**Bottom-up approach :** It is usually implemented in iterative way, starting with a solution to the smallest instance of the problem.

**Variations of Decrease and Conquer :**

There are three major variations of decrease-and-conquer:

1. Decrease by a constant
2. Decrease by a constant factor
3. Variable size decrease

**Decrease by a Constant** : In this variation, the size of an instance is reduced by the same constant on each iteration of the algorithm. Typically, this constant is equal to one , although other constant size reductions do happen occasionally.

Below are example problems :

- Insertion sort
- Graph search algorithms: DFS, BFS
- Topological sorting
- Algorithms for generating permutations, subsets

**Decrease by a Constant factor**: This technique suggests reducing a problem instance by the same constant factor on each iteration of the algorithm. In most applications, this constant factor is equal to two. A reduction by a factor other than two is especially rare.
Decrease by a constant factor algorithms are very efficient especially when the factor is greater than 2 as in the fake-coin problem. Below are example problems :

- Binary search
- Fake-coin problems
- Russian peasant multiplication

**Variable-Size-Decrease** : In this variation, the size-reduction pattern varies from one iteration of an algorithm to another.
As, in problem of finding gcd of two number though the value of the second argument is always smaller on the right-handside than on the left-hand side, it decreases neither by a constant nor by a constant factor.
Below are example problems :

- Computing median and selection problem.
- Interpolation Search
- Euclid's algorithm

There may be a case that problem can be solved by decrease-by-constant as well as decrease-by-factor variations, but the implementations can be either recursive or iterative. The iterative implementations may require more coding effort, however they avoid the overload that accompanies recursion.

**Reference :**
Anany Levitin
Decrease and conquer