Divide And Conquer

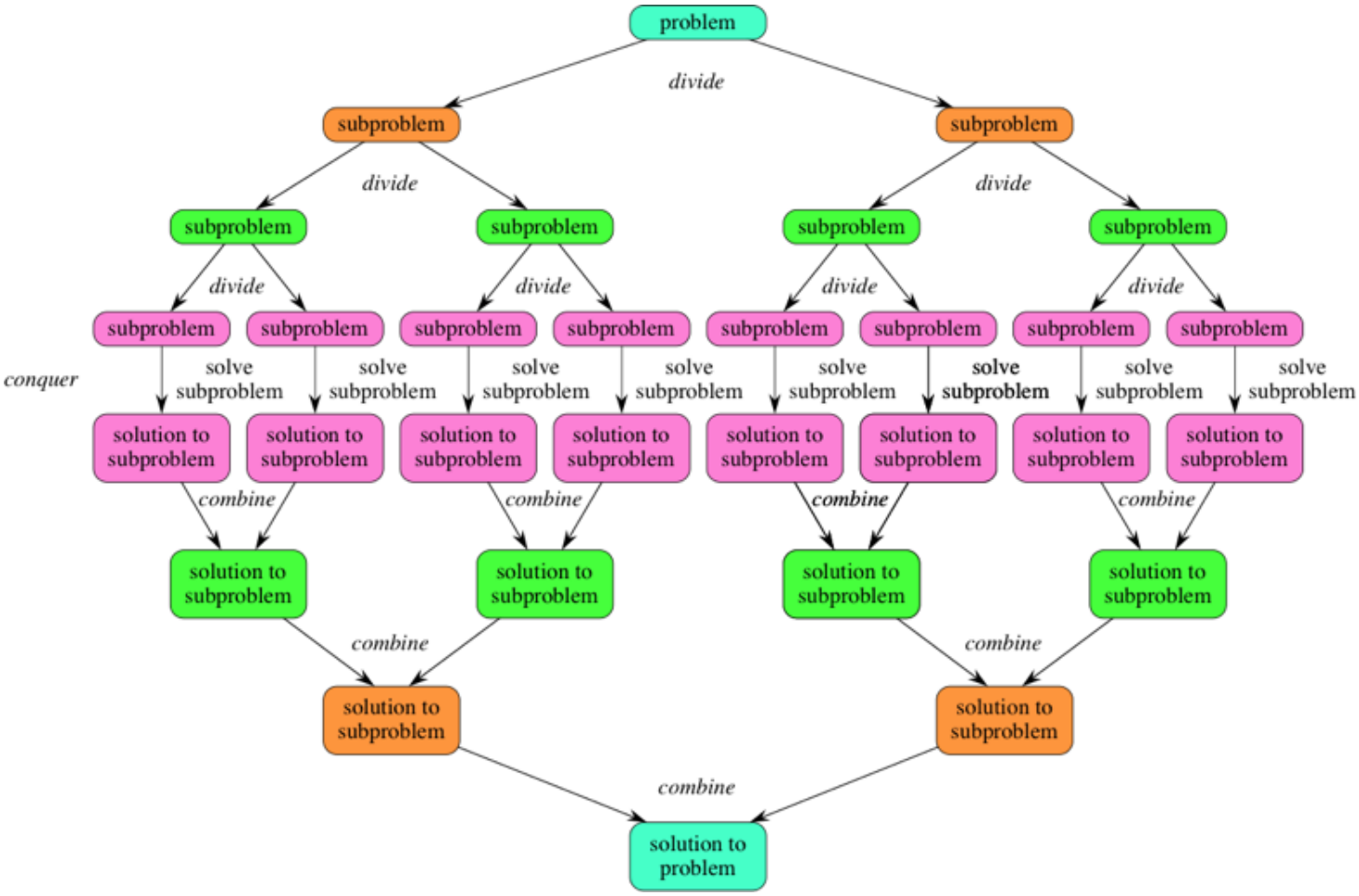
A typical Divide and Conquer algorithm solves a problem using following three steps

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the sub-problems by solving them recursively. If they are small enough, solve the sub-problems as base cases.
3. **Combine** the solutions to the sub-problems into the solution for the original problem.

Parallelism
Divide-and-conquer algorithms are naturally adapted for execution in multi-processor machines, especially shared-memory systems where the communication of data between processors does not need to be planned in advance, because distinct sub-problems can be executed on different processors.

Divide and Conquer (D & C) vs Dynamic Programming (DP)

Both paradigms (D & C and DP) divide the given problem into subproblems and solve sub-problems. How to choose one of them for a given problem? Divide and Conquer should be used when same sub-problems are not evaluated many times. Otherwise Dynamic Programming or Memoization should be used. For example, Binary Search is a Divide and Conquer algorithm, we never evaluate the same subproblems again. On the other hand, for calculating nth Fibonacci number, Dynamic Programming should be preferred.



Following are some standard algorithms that are Divide and Conquer algorithms.

1. **Binary Search** is a searching algorithm. In each step, the algorithm compares the input element x with the value of the middle element in array. If the values match, return the index of middle. Otherwise, if x is less than the middle element, then the algorithm recurs for left side of middle element, else recurs for right side of middle element.
2. **Quicksort** is a sorting algorithm. The algorithm picks a pivot element, rearranges the array elements in such a way that all elements smaller than the picked pivot element move to left side of pivot, and all greater elements move to right side. Finally, the algorithm recursively sorts the subarrays on left and right of pivot element.
3. **Merge Sort** is also a sorting algorithm. The algorithm divides the array in two halves, recursively sorts them and finally merges the two sorted halves.
4. **Closest Pair of Points** The problem is to find the closest pair of points in a set of points in x-y plane. The problem can be solved in O(n^2) time by calculating distances of every pair of points and comparing the distances to find the minimum. The Divide and Conquer algorithm solves the problem in O(n*Logn) time.
5. **Strassen's Algorithm** is an efficient algorithm to multiply two matrices. A simple method to multiply two matrices need 3 nested loops and is O(n^3). Strassen's algorithm multiplies two matrices in O($n^{2.8974}$) time.

6. **Karatsuba algorithm for fast multiplication** of large numbers. It does multiplication of two n-digit numbers in at most $3n^{\log_2 3} \approx 3n^{1.585}$ single-digit multiplications in general (and exactly $n^{\log_2 3}$ when n is a power of 2). It is therefore faster than the classical algorithm, which requires n^2 single-digit products. If n = 2^{10} = 1024, in particular, the exact counts are 3^{10} = 59,049 and $(2^{10})^2$ = 1,048,576, respectively.