# CSCI 270 Lecture 16: Divide and Conquer

## Divide and Conquer

Divide and Conquer is a **runtime improvement technique**. Generally, we have a basic brute-force/greedy/dynamic-programming algorithm, and then we attempt to improve our performance using Divide and Conquer.

Both MergeSort and QuickSort are examples of Divide and Conquer algorithms. You take the input, split it into pieces, recursively solve each piece, and then figure out how to combine the pieces together. Typically the combine phase is the difficult part.

Divide and Conquer recursion is **completely different** than Dynamic Programming recursion.

- DP is sequential (make the first decision, then recursively make the next decisions).
- Divide and Conquer recursion is parallelized: we split it up into several independent pieces which can be solved in parallel.
- DP recursion can be unrolled into a more efficient iterative algorithm.
- Divide and Conquer recursion cannot be unrolled in a straight-forward manner (it can often be done, but it is usually more trouble than it's worth).

## Counting Inversions

You're writing the software for Pandora270. A user ranks $n$ songs, and you want to find other users with similar tastes in music (and recommend music based off these matchings).

Suppose there are 5 songs, conveniently named 1, 2, 3, 4, 5, which I have ranked in that order. You have ranked them in order 1, 3, 4, 2, 5.

How similar are our tastes? In one sense, we have only ranked two songs in the same slot. However, if you look more closely, you'll see we only disagree on one song: song 2.

An **inversion** is a pair of songs $i$ and $j$ such that one user ranks $i < j$ and the other ranks $i > j$. How many inversions are there in the above example?

We want to count the number of inversions between two different rankings. For simplicity, we assume that one of the rankings is the numbers 1 through $n$ in increasing order.

We could just solve this in $\theta(n^2)$ time using a brute-force count. Instead, we will use Divide and Conquer to improve our running time.

Your ranking:

1 5 4 8 10 2 6 9 12 11 3 7
1 5 4 8 10 2 | 6 9 12 11 3 7 (Divide: O(1))
5 inversions | 8 inversions (Recursively Solve! $2f(\frac{n}{2})$)

1. Whats our base case?

2. Can we just return $5 + 8$?

3. How long does our combine phase take?

4. What's the resulting recurrence relation?

5. What's the overall runtime of our algorithm?

Often your first attempt at a Divide and Conquer algorithm will net no improvement. That doesn't mean Divide and Conquer won't work, it simply means you need to improve a part of the algorithm.

1. How can we improve the combine phase?

2. What's the recurrence relation now?

3. Whats the overall runtime?

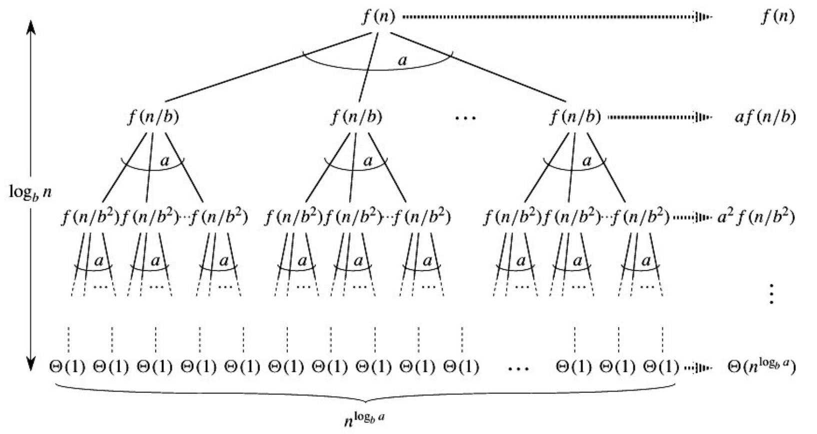
## Master Theorem

- Can solve (almost) any recurrence relation of the form $f(n) = af(\frac{n}{b}) + g(n)$, with constants $a \geq 1, b > 1$.
- Compare $g(n)$ with $n^{\frac{\log a}{\log b}}$.
- Case 1: If $g(n) = \theta(n^{\frac{\log a}{\log b}})$, then $f(n) = \theta(g(n) \log n)$.
- Case 2: If $g(n) = \Omega(n^{\frac{\log a}{\log b}+\epsilon})$, for some $\epsilon > 0$, then $f(n) = \theta(g(n))$.
- What should Case 3 be?

Master Theorem Questions

- How much work is done at the top level?
- How many children does the root node have?
- How much work is required at a child node?
- How many grandchildren are there?
- How much work at a grandchild node?
- What is the depth of the tree?
- How many leaf nodes are there?
- How much work at a leaf node?
- How much work is done at the bottom level?
- Solve: $f(n) = 2f(\frac{n}{2}) + cn$
- Solve: $f(n) = f(\frac{n}{2}) + 1$
- Solve: $f(n) = 8f(\frac{n}{2}) + 1000n^2$
- Solve: $f(n) = 2f(\frac{n}{2}) + n^2$
- Solve: $f(n) = 2f(\frac{n}{2}) + \frac{n}{\log n}$ (This one is trickier than it looks!)

Analyzing Master Theorem by Recursion Tree (credit: GeeksForGeeks)



Extra Problems:

- Chapter 5: exercises 1, 2, 3, 6
- Challenge Problems: Chapter 5, exercise 7