Techniques for Divide and Conquer Algorithms

Winter 2023

[[Have a running example. Perhaps the Skyline problem from the problem archive.]]

1 How to Design a Divide and Conquer Algorithm

The main part of a divide and conquer algorithm is that you are going to want to make recursive calls to solve smaller versions of your original problem as a major part of your algorithm. You need to ask yourself things like:

- What are reasonable subproblems? Often you want to do something like split our problem in half somehow and produce problems that way.
- How do we combine the answers? Sometimes we need to pick our problems carefully so that this can be done. Sometimes we need to come up with some new algorithm to combine them.

2 How to Write a Divide and Conquer Algorithm

The divide an conquer algorithms all follow the same basic format:

1. If the input size is small, use some base case algorithm.
2. Create several smaller subproblems.
3. Recursively solve subproblems.
4. Combine answers to recursive calls to produce answer to the original problem.

Note that the base case in line 1 here is critical, as otherwise you end up with a never-ending chain of recursive calls.

Also note that the recursive calls generally should be a *constant multiple* smaller than the original in size in order to get a polynomial time algorithm. If even a single call is size $n-1$, the algorithm is likely no longer polynomial.

3 How to Prove Correctness of a Divide and Conquer Algorithm

The proof of correctness of a divide and conquer algorithm is usually done by induction on the size of the input.

Our base case is where the input size is small enough that we use our small input special case of the algorithm. In this case, correctness depends on the correctness of this "base case" algorithm.

For the inductive step, we can assume that the recursive calls to our algorithm compute the correct answers and we need to show that we combine them in such a way that we get the correct answer to the original problem.

4 Runtime Analysis

The runtime analysis usually makes use of the Master Theorem. First one needs to come up with a recurrence relation for the runtime $T(n)$. This usually has some overhead term attributed to the finding the subproblems and combining the answers, and a recursive term from the recursive calls.

In most instances we cover in this class, the Master Theorem can be used to solve this recurrence. However, there are some cases (for example where the sizes of the recursive calls are not all the same) where a more complicated version of the Master Theorem is needed.

[[Work some additional examples, maybe deterministic order statistics.]]