CIS 121—Data Structures and Algorithms with Java—Spring 2018

Divide & **Conquer**—Monday, February 5th / Tuesday, February 6th

Learning Goals

During this lab, you will:

- Develop intuition for the divide and conquer paradigm
- Review the mergesort algorithm

Divide and Conquer: An Overview

What does it mean to have a "divide and conquer" algorithm?

Divide the problem into a number of subproblems that are smaller instances of the same problem
Conquer the subproblems by solving them recursively.
Combine the solutions to the subproblems into the solution for the original problem.

How do you recognize situations where "divide and conquer" might work? A natural first question is "Can I break this down into subproblems equivalent to the original problem?" You can then ask "how can I solve these problems and combine them to reach a solution for my original problem?" Usually if you can solve each subproblem and combine them, it involves some sort of recursion. In order to better understand the "divide and conquer" paradigm, we will do an in depth study on a familiar algorithm: Mergesort.

Mergesort

Problem statement: Given an array of length n, sort it in ascending order (could also be descending). You cannot assume anything about the contents of the array.

We can apply the three principles of divide and conquer when thinking about approaching this problem.

Divide: First we ask ourselves: Can we divide this into equivalent subproblems? Yes, sorting two halves of the array of size $\frac{n}{2}$ each is an equivalent subproblem.

Conquer: How can I recursively sort the two halves of the array? That's easy—since I already broke it up into subproblems, I will recurse using mergesort on the two halves, until I hit the base case. The base case of a singleton element means the array is sorted.

Combine: Once I have two sorted arrays, I can combine them into a larger array by interleaving them.

I encourage you to review your notes on the pseudocode of the algorithm and identify which parts of the code correspond to these categories. Many algorithms of this sort have a similar structure, so intimately understanding this one will help you build your own divide and conquer algorithms later.

Proof of running time: We have a lot of practice with this! This is a straightforward recurrence relation. With an array of size n, you solve it by recursing on both halves of the array (size $\frac{n}{2}$. In order to combine it, you interleave those two sorted arrays, which takes $O(n)$ time. Therefore, the recurrence relation is $T(n) = 2T\left(\frac{n}{2}\right) + n$. Employ whatever method of solving recurrences that you are comfortable with, and you will get $O(n \lg n)$.

Proof of correctness:

Author's Note: I highly recommend reading the CLRS section 2.3.1 for an excellent overview of this topic.

Problems

Local maximum

Problem 1. You are given an integer array with the following properties:

- Integers in adjacent positions are different
- $arr[0] < arr[1]$
- $arr[arr.length - 2] > arr[arr.length - 1]$

A position i is referred to as a local maximum if $arr[i]$¿$arr[i-1]$ and $arr[i] > arr[i+1]$.

Example: You have an array $[0, 1, 5, 3, 6, 3, 2]$. There are multiple local maxes at 5 and 6.

Propose an efficient algorithm that will find a local maximum and return its index.

Maximum subarray sum

Problem 2. Given an integer array (contains positive and negative values), return the sum of the largest contiguous subarray which has the largest sum.

Maximum Profit

Problem 3. Given, in an array, the prices of a stock for each day for n days, what is the maximum profit you can make with exactly one buy transaction and one sell transaction?

Design a divide and conquer algorithm to solve this problem.

Element index matching

Problem 4. You are given a sorted array of n distinct integers A$[1...n]$. Design an $O(\lg n)$ time algorithm that either outputs an index i such that A$[i] = i$ or correctly states that no such index i exists.

Counting duplicates

Problem 5. Given an unsorted array of n integers, design an algorithm to remove all the duplicate elements.