CS 161 Winter 22 - Section 2

Divide and Conquer

Smallest Missing Element

Design an algorithm to find the smallest missing element in a sorted array of non-negative distinct integers.

Maximum Sum Subarray

Given an array of integers $A[1..n]$, find a contiguous subarray $A[i,..j]$ with the maximum possible sum. The entries of the array might be positive or negative.

1. What is the time complexity of a brute force solution?
2. The maximum sum subarray may lie entirely in the first half of the array or entirely in the second half. What is the third and only other possible case?
3. Using the above apply divide and conquer to arrive at a more efficient algorithm.
   (a) Prove that your algorithm works.
   (b) What is the time complexity of your solution?
4. Advanced (Take Home) - Can you do even better using other non-recursive methods? ($O(n)$ is possible)

Space Complexity

Given an array of size $n-1$ containing all the integers between 1 and n except for one (not necessarily sorted), design an algorithm to find the missing number using $O(1)$ extra space.

Recurrence Relations

Recall the Master theorem from lecture:

Theorem 0.1 *Given a recurrence $T(n) = aT(\frac{n}{b}) + O(n^d)$ with $a \geq 1$, and $b > 1$ and $T(1) = \Theta(1)$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

What is the Big-Oh runtime for algorithms with the following recurrence relations?

1. $T(n) = 3T(\frac{n}{2}) + O(n^2)$
2. $T(n) = 4T(\frac{n}{2}) + O(n)$
3. $T(n) = 2T(\sqrt[3]{n}) + O(\log n)$