CS 161 (Stanford, Winter 2023) Section 2

1 Master Theorem

Recall the Master theorem from lecture:

Theorem 1 *Let $T(n) = aT(\frac{n}{b}) + O(n^d)$ be a recurrence where $a \geq 1$, and $b > 1$. 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?

1.1 $T(n) = 3T(\frac{n}{2}) + O(n^2)$

1.2 $T(n) = 4T(\frac{n}{2}) + O(n)$

1.3 $T(n) = 2T(\sqrt{n}) + O(\log n)$

2 Single-dimensional Tarski's fixed point theorem

Given a 1-indexed sorted array A of n integers such that $A[1] \geq 1$ and $A[n] \leq n$, a (very) special case of Tarski's fixed point theorem says that there is some i such that $A[i] = i$.

2.1 Algorithm Design

Design an algorithm for finding such an i.

2.2 Runtime Analysis

Analyze the runtime of your algorithm in 2.1.

3 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.

3.1 Brute Force

What is the complexity of a brute force solution?

3.2 Divide-and-Conquer

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?

3.3 Algorithm Design

Use the cases in 3.2 to arrive at a more efficient algorithm. What is the complexity of your algorithm?

3.4 Further Optimization (Optional)

Can you do even better using other non-recursive methods? ($O(n)$ is possible)

4 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.