CS 161 (Stanford, Winter 2025) Section 2

Divide and Conquer

Finding a fixed point of an array

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 $A[i] = i$. Design an algorithm for finding such an i.

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

Select algorithm

In lecture, we encountered the Select algorithm to find the kth smallest element in an array. We saw that the time complexity of this algorithm depends on how close our pivot element is to the median of the array. This problem explores this dependence in more detail.

1. If we are guaranteed to select exactly the median as our pivot (i.e. we get a 50-50 split), what is the runtime of Select? What if we have no guarantees on our pivot?
2. If we are guaranteed to select a pivot that gives us at worst a c-$(1-c)$ split, where $\frac{1}{2} \leq c < 1$ is some constant that doesn't depend on n, what is the runtime of Select?
3. Assume that in the worst case, it takes more than a constant number of splits to cut the size of our array in half. (This means that the number of splits required is $\Omega(1)$ but not $\Theta(1)$, i.e. $\omega(1)$.) Show that the runtime of Select in this case must be larger than $O(n)$.