Homework 1 — Divide and Conquer

Due: January 19th

A few of these questions may appear on or help you prepare for the quiz on **January 19th**.
Note that these selected solutions and comments do not necessarily represent complete answers.

1. Using the Tree Method, solve each of the following recurrence relations, then sort them from smallest to largest by their Big-O estimates. Where possible, verify your answer using the Master Theorem.
   (a) $T(n) = T(n/2) + O(1) = O(\log n)$
   (b) $T(n) = T(n/2) + O(n^2) = O(n^2)$
   (c) $T(n) = 3T(n/3) + O(n) = O(n \log n)$
   (d) $T(n) = 2T(n/4) + O(1) = O(\sqrt{n})$
   $O(\log n) < O(\sqrt{n}) < O(n \log n) < O(n^2)$

2. Consider the following divide and conquer algorithm:

```
FindFirst$(x, A = (a_0, a_1, \ldots, a_{n-1}), \text{low}, \text{high})$
Input: An integer $x$ and a finite, non-empty sequence $A$ of $n$ integers, sorted in ascending
    order, where initially low $= 0$ and high $= n$
Output: The index of the first occurrence of $x$ in $A$, or $-1$ if $x$ is not in $A$
1: let mid be $\lfloor \frac{\text{low}+\text{high}}{2} \rfloor$
2: if $a_{\text{low}} = x$ then
3:     return low
4: else if high $-$ low $= 1$ then
5:     return $-1$
6: else if $a_{\text{mid}} \geq x$ then
7:     return FindFirst$(A, \text{low}, \text{mid})$
8: else
9:     return FindFirst$(A, \text{mid}+1, \text{high})$
```

This algorithm finds the first occurrence of an element in a sequence. For example, given $x = 2$ and $A = (1, 2, 2, 2, 4, 4, 12, 20, 20, 20)$, it returns 1. Further consider the following lemma:

Lemma: Let A be a finite, non-empty sequence of n integers, sorted in ascending order. If there exists i such that $a_i = x$ and either $i = 0$ or $a_{i-1} \neq x$, then FindFirst$(x, A, 0, n)$ returns $i \geq 0$.

Suppose that you are asked to prove this lemma by induction.

(a) Which variable should be inducted over, and why should it *not* be n?

Generally speaking, a proof should induct over the size of the problem, such that the Inductive Hypothesis can be used to justify the correctness of any recursive calls. Often, the size of the problem is, in fact, n.

However, in order to preserve the elements' original indices, FindFirst does not reduce the size of A when recursing, so inducting over n would make the hypothesis inapplicable. Instead, a proof should induct over (high $-$ low), which represents the true size of the problem.

(b) Would proving this lemma be enough to establish the correctness of this algorithm?

No; since it is also possible that $x \notin A$, the inverse must also be proven: if there exists no i such that $a_i = x$, then FindFirst$(x, A, 0, n)$ returns $i < 0$.

3. Suppose that you are given a finite, sorted sequence of n distinct integers $A = (a_0, a_1, \ldots a_{n-1})$.

   (a) Give pseudocode for an efficient divide and conquer algorithm that determines whether or not there exists an i such that $a_i = i$.

   For example, given $A = (-10, -4, 2, 41)$, your algorithm should return 2. Given $A = (4, 7, 19, 20)$, it should indicate that there exists no such i.

```
FINDINDEX($A = (a_0, a_1, \ldots, a_{n-1})$, low, high)
Input: A finite sequence $A$ of $n$ distinct integers, sorted in ascending order, where
    initially low $= 0$ and high $= n$
Output: An index $i$ such that $a_i = i$, or $-1$ if no such $i$ exists
1: let mid be $\lfloor \frac{\text{low}+\text{high}}{2} \rfloor$
2: if high $-$ low $= 0$ then
3:     return $-1$
4: else if $a_{\text{mid}} = \text{mid}$ then
5:     return mid
6: else if $a_{\text{mid}} > \text{mid}$ then
7:     return FINDINDEX($A$, low, mid)
8: else
9:     return FINDINDEX($A$, mid $+ 1$, high)
```

   Consider: since the elements are sorted and distinct, they must increase at least as fast as their indices. If, for example, $a_{mid} < mid$, then any i would have to be in the right half, where the faster growing elements have the opportunity to "catch up" to the indices.

   (b) Prove that your algorithm is correct.

   Lemma:

   Let A be a finite sequence of n distinct integers, sorted in ascending order. There exists an i such that $a_i = i$ if and only if FINDINDEX$(A, 0, n)$ *returns $i \geq 0$.*

   Proof:

   - *Let A be a finite sequence of n distinct integers, sorted in ascending order. Further suppose there exists an $low \leq i < high$ such that $a_i = i$. Note then that $high - low \geq 1$.*
   - Basis Step:
     - *Let $high - low = 1$. Since there exists an $low \leq i < high$ such that $a_i = i$, it must be the case that $a_{mid} = mid$, thus,* FINDINDEX *correctly returns $mid \geq 0$.*
   - Inductive Hypothesis:
     - *Suppose that, for all $1 \leq high - low \leq k$,* FINDINDEX *returns $i \geq 0$.*
   - Inductive Step:
     - *Let $high - low = k + 1$. Two possibilities exist:*
       - *If $a_{mid} = mid$, then* FINDINDEX *correctly returns $mid \geq 0$.*
       - *If $a_{mid} \neq mid$, without loss of generality, suppose $a_{mid} > mid$. Since the elements of A are distinct, it must be the case that $\forall_{mid<j<high}(a_j > j)$. Then, since there exists an $low \leq i < high$ such that $a_i = i$, it must be the case that $low \leq i < mid$, where $mid - low \leq k$. By the hypothesis,* FINDINDEX *correctly returns $i \geq 0$.*
     - *In all possible cases,* FINDINDEX *correctly returns $i \geq 0$.*
   - *Thus, if there exists an i such that $a_i = i$, then* FINDINDEX *returns $i \geq 0$.*

Proof (cont.):

- *Suppose now, for contraposition, that there exists no i such that $a_i = i$. Note then that $high - low \geq 0$.*
- Basis Step:
  - *Let $high - low = 0$. Then* FindIndex *correctly returns $-1 < 0$.*
- Inductive Hypothesis:
  - *Suppose that, for all $0 \leq high - low \leq k$,* FindIndex *returns $i < 0$.*
- Inductive Step:
  - *Let $high - low = k + 1$. Since there exists no i such that $a_i = i$, $a_{mid} \neq mid$. Without loss of generality, suppose $a_{mid} > mid$.*
  - *Then it must be the case that $\forall_{low \leq j < mid}(a_j \neq j)$ where $mid - low \leq k$. By the hypothesis,* FindIndex *correctly returns $i < 0$.*
- *Thus, by contraposition, if* FindIndex *returns $i \geq 0$, then there exists an i such that $a_i = i$.*
- *Therefore, by the Principle of Mathematical Induction,* FindIndex*$(A, 0, n)$ returns $i \geq 0$ if and only if $a_i = i$.* $\square$

(c) Set up and solve a recurrence relation estimating the time complexity of your algorithm.

$T(n) = a \cdot T(m) + O(g(n)) = 1 \cdot T(n/2) + O(1) = O(\log n)$

4. A sequence is said to have a *majority element* if more than half of its elements are equal.

   Suppose that you are given a finite sequence of n elements $A = (a_0, a_1, \ldots, a_{n-1})$. Further suppose that its elements are of unknown type, and they can *only* be compared for equality. That is, "$a_0 < a_1$" is prohibited, but "$a_0 = a_1$" can done in constant time.

   (a) Give pseudocode for an efficient divide and conquer algorithm that finds the majority element if and only if it exists.

   For example, given $A = (-1, 8, \lambda, 8, \star, 8, \star, 8, 8)$, your algorithm should return 8. Given $A = (2, \aleph, \text{"foo"}, 2)$, it should indicate that there exists no majority element.

   Suppose the sequence was divided in two and each half's subproblem was solved. We would then have, as subsolutions, the majority element of the left (if one exists) and the majority element of the right (if one exists).

   Then there are four remaining possibilities: neither half has a majority, one half has a majority and the other does not, the halves have different majorities, or the halves have the same majority. Note that the majority overall cannot possibly be a minority of both halves.

   (b) Prove that your algorithm is correct.

   (c) Set up and solve a recurrence relation estimating the time complexity of your algorithm.

5. Suppose that you are given a finite sequence of n numbers $A = (a_0, a_1, \ldots, a_{n-1})$, where each number represents the price of a stock at some point in time, and the sequence is sorted by time.

   (a) Give pseudocode for an efficient divide and conquer algorithm that identifies the best (*buy*, *sell*) pair, such that if you were to buy the stock at *buy* price and sell it at *sell* price, you would maximize your profit. For the sake of simplicity, you may assume that stocks can be sold at the same time they are bought, but they cannot be sold before they are bought.

   For example, given $A = (10, 4, 3, 7, 6, 15)$, your algorithm should return $(3, 15)$.

   Suppose the sequence was divided in two and each half's subproblem was solved. We would then have, as subsolutions, the best buy-sell pair on the left and the best buy-sell pair on the right.

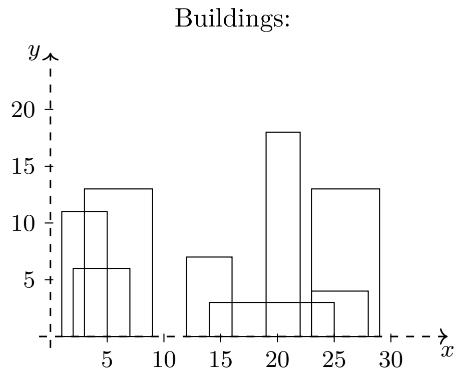
   Then the only remaining possibility to consider is the pair formed by buying on the left and selling on the right, since it would be nonsensical to sell on the left and buy on the right. This is simply the minimum element on the left paired with the maximum element on the right.

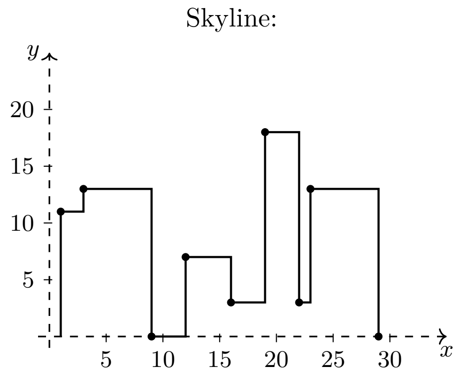
   (b) Prove that your algorithm is correct.

   (c) Set up and solve a recurrence relation estimating the time complexity of your algorithm.

6. Let a *building* be represented by a triple $b_i = (l_i, h_i, r_i)$, where l_i and r_i are its left and right x-coordinates and h_i is its height.

   Suppose that you are given a set of n buildings, $S = \{b_0, b_1, \ldots, b_{n-1}\}$. Their *skyline* is a sequence of (x, y) pairs defining their collective visible horizon. For example:





   Given the buildings $\{(3, 13, 9), (1, 11, 5), (12, 7, 16), (14, 3, 25), (19, 18, 22), (2, 6, 7), (23, 13, 29), (23, 4, 28)\}$, the corresponding skyline is $((1, 11), (3, 13), (9, 0), (12, 7), (16, 3), (19, 18), (22, 3), (23, 13), (29, 0))$.

   (a) Give pseudocode for an efficient divide and conquer algorithm that computes such skylines.

   Suppose the set was divided in two and each half's subproblem was solved. We would then have, as subsolutions, the skyline on the left and the skyline on the right. Note that, by definition, a skyline is sorted by x-coordinate.

   Then, for any one given point in the left skyline, the only remaning possibility to consider is whether or not that point's x-coordinate is less than that of the next point in the right while its y-coordinate is greater than that of the previous point in the right.

   (b) Prove that your algorithm is correct.

   (c) Set up and solve a recurrence relation estimating the time complexity of your algorithm.