This document is an example of the kind of discourse that should go in your problem set write-ups. Attempt 1, we developed together during class on 16 September 2022. Attempt 2 was my try at a write-up in the 45 minutes before class that same day. Neither is perfect, but they're pretty good and at the right level of specificity and completeness that I'll be expecting.

Here's the (very simple) problem we're talking about. Given an array $A[0...n-1]$ of integers where $n > 0$, find the *range* of the integers. That is, find the difference between the largest and the smallest integers in A.

Attempt 1

Time spent: 45 minutes

Collaborators: the whole class

This is the write-up we assembled during class on September 16.

The Algorithm

1. Initialize $Max := A[0]$, $Min := A[0]$
2. For each $e \in A$, if $e > Max$, set $Max := e$
3. For each $e \in A$, if $e < Min$, set $Min := e$
4. Return $Max - Min$

Correctness of the Algorithm

The main question in evaluating the correctness of this algorithm is whether Step 2 actually ends with Max equal to the largest value found in A.

Let M be defined as the maximum value of $A[0], A[1], ..., A[n-1]$. Then $\exists j \in 0, 1, ..., n-1$ such that $M = A[j]$. So during the iteration of Step 2 when $e = A[j] = M$, Max will be assigned the value M. Furthermore, in all later iterations Max will already be equal to M, so "the larger of e and Max" will still be M. Thus, by the end of Step 2 and throughout the remainder of the algorithm, $Max = M$.

We could get super-pedantic about the "Furthermore" bit and prove it by induction or contradiction, but let's not.

Analysis of Runtime

Step 1 takes $O(1)$ time, since it consists of two array lookups and two assignments regardless of the value of n.

Step 2 requires $n-1$ iterations, each of which is constant time (i.e., an if-statement and maybe an assignment), so Step 2 takes $O(n)$ time.

Step 3 is just like Step 2, and thus $O(n)$.

Step 4 involves one subtraction, and is thus $O(1)$.

Putting it all together, the runtime is dominated by Steps 2 and 3, which are each $O(n)$, so the algorithm as a whole takes $O(n)$ time.

The argument that this algorithm is $\Omega(n)$ and thus $\Theta(n)$ is nearly identical to the above, so we will not include it here.

Attempt 2

This is the write-up I did on my own before class.

Time spent: 1 hour

Collaborators: None

The Algorithm

We will use L to represent the largest value from A, and S to represent the smallest value.

1. Initialize $L = S = A[0]$
2. For each $k \in 1, ..., n-1$, set $L :=$ the larger of $A[k]$ and L
3. For each $k \in 1, ..., n-1$, set $S :=$ the smaller of $A[k]$ and S
4. Return $L - S$

Correctness of the Algorithm

Obvious or not? Audience.

The main question in evaluating the correctness of this algorithm is whether Step 2 actually ends with L equal to the largest value found in A.

Let M be defined as the maximum value of $A[0], A[1], ..., A[n-1]$. Then $\exists j \in [0...n-1]$ such that $M = A[j]$. So during the iteration of Step 2 when $e = A[j] = M$, L will be assigned the value M since $A[j]$ is the maximum value found in A. Furthermore, in all iterations $k = j...n-1$, L will already be equal to M, so "the larger of $A[k]$ and L" will still be L. Thus, by the end of Step 2 and throughout the remainder of the algorithm, $L = M$.

We could get super-pedantic about the "Furthermore" bit and prove it by induction or contradiction, but let's not.

Analysis of Runtime

Step 1 takes $O(1)$ time, since it consists of one array lookup and two assignments, regardless of the value of n.

Step 2 involves $n - 1$ iterations, each of which takes constant time (i.e., an array lookup, a comparison, and an assignment). Therefore Step 2 takes $O(n)$ time.

Step 3 involves the same number of operations as Step 2, and thus also takes $O(n)$ time.

Step 4 involves one subtraction regardless of the value of n, and thus takes $O(1)$ time.

Putting all the steps together, Steps 2 and 3 dominate the runtime at $O(n)$ each, so the overall algorithm takes $O(n)$ time.

The argument that this algorithm is $\Omega(n)$ and thus $\Theta(n)$ is nearly identical to the above, so we will not include it here.