Cosc 30: Discrete Mathematics

Prishita Dharampal

Problem 1. Let $A[1..n]$ be an array of n distinct integers (which can be negative, positive, or zero), sorted in increasing order; in other words,

$$A[1] < A[2] < \cdots < A[n].$$

The following algorithms, on the input $A[1..n]$, should either correctly compute an index i such that $A[i] = i$, or report that there is no such index.

```
FINDMATCH1(A[lo .. hi]):
    if lo > hi return no such
   index exists
    m <- { ⌈(lo+hi)/2⌉ }
    if A[m] = m:
        return m
    if A[m] < m:
        FINDMATCH1(A[lo .. m-1])
    if A[m] > m:
        FINDMATCH1(A[m+1 .. hi])
```

```
FINDMATCH2(A[lo .. hi]):
    if lo > hi return no such
   index exists
    m <- { ⌈(lo+hi)/2⌉ }
    if A[m] = m:
        return m
    if A[m] > m:
        FINDMATCH2(A[lo .. m-1])
    if A[m] < m:
        FINDMATCH2(A[m+1 .. hi])
```

For each algorithm, decide whether it correctly performs the task. For incorrect algorithms, construct an input where the algorithm fails. For correct algorithms, prove correctness using induction.

Solution.

1. `FINDMATCH1` fails with the input `[-2,0,1,4,20]`. At $m = 3$, we have $A[3] = 1 < 3$, so the algorithm recurses on $A[1..2]$, but the correct solution $A[4] = 4$ lies in the discarded right half.

2. We prove the correctness of the proposition $P(A[1 \ldots n])$ for every integer array of length at least 1 satisfying the input conditions, by induction:

   Proposition.

   $P(A[lo \ldots hi])$: If $A[lo \ldots hi]$ is a strictly increasing array of integers, then `FINDMATCH2`$(A[lo \ldots hi])$ correctly returns an index m such that $A[m] = m$ if such an index exists, and otherwise correctly reports that no such index exists.

Let A be an arbitrary integer array of length $n \geq 1$ sorted in increasing order. Assume statement P is true for every array $P(A[lo..hi])$ shorter than A.

(a) If $A[m] > m$, then for all $n > m$ we have

$$A[n] \geq A[m] + (n - m) > n,$$

so no index $n > m$ can satisfy $A[n] = n$. Thus any solution must lie in $A[lo \ldots m-1]$. `FINDMATCH2` recurses on $A[lo \ldots m-1]$, and by induction hypothesis returns a correctly output since $A[lo \ldots m-1]$ is smaller than $A[lo \ldots hi]$.

(b) If $A[m] < m$, then for all $n < m$ we have

$$A[n] \leq A[m] - (m - n) < n,$$

so no index $n < m$ can satisfy $A[n] = n$. Thus any solution must lie in $A[m+1 \ldots hi]$. `FINDMATCH2` recurses on $A[m+1 \ldots hi]$, and by induction hypothesis correctly returns an output since $A[m+1 \ldots hi]$ is smaller than $A[lo \ldots hi]$.

(c) Otherwise if $A[m] = m$ `FINDMATCH2` correctly returns index m (by definition).

Base case: If $lo > hi$, the array is empty and no index can satisfy $A[i] = i$.

`FINDMATCH2` correctly tells us that no such index exists.

Hence, the base case also holds. And `FINDMATCH2` correctly solves the problem for all valid arrays, i.e. $P(A[lo \ldots hi])$ is true.

Problem 2. Let $A[1..n]$ be an array of n distinct integers (which can be negative, positive, or zero).
The following algorithms, on the input $A[1..n]$, should sort the elements in array A in increasing order; that is,

$$A[1] < A[2] < \cdots < A[n].$$

```
AWKWARDSORT1(A[lo .. hi]):
    if hi <= lo + 1: ⟨⟨if n <= 2⟩⟩
        if A[lo] > A[hi]:
            swap A[lo] and A[hi]
        return A
    else:  ⟨⟨if n > 2⟩⟩
        AWKWARDSORT1(A[lo .. hi-1])
        AWKWARDSORT1(A[lo+1 .. hi])
        return A
```

```
AWKWARDSORT2(A[lo .. hi]):
    if A[lo] > A[hi]:
        swap A[lo] and A[hi]
    if hi <= lo + 1: ⟨⟨if n <= 2⟩⟩
        return A
    else:  ⟨⟨if n > 2⟩⟩
        AWKWARDSORT2(A[lo .. hi-1])
        AWKWARDSORT2(A[lo+1 .. hi])
        return A
```

For each algorithm, decide whether it correctly performs the task. For incorrect algorithms, construct an input where the algorithm fails. For correct algorithms, prove correctness using induction.

Solution.

1. `AWKWARDSORT1` breaks with the input [3, 2, 1].

   `AWKWARDSORT1([3, 2, 1]) = [2, 1, 3]`,

   after the recursive calls, the element 1 never moves past 2, so the array remains unsorted.

2. We prove the correctness of the proposition $P(A[1 \ldots n])$ for every integer array of length greater than 2 satisfying the input conditions, by induction:

   Proposition.

   $P(A[lo \ldots hi])$: If $A[lo \ldots hi]$ contains distinct integers, then `AWKWARDSORT2`$(A[lo \ldots hi])$ returns an array sorted in increasing order.

   Assume the statement holds for all arrays $A[1 \ldots k]$ of length k where $3 \leq k < n$. Then, for $A[1 \ldots n]$:

   (a) If $A[1] > A[n]$: the first `if` block swaps them. This ensures that the smaller of $A[1]$ and $A[n]$ is placed at position 1 and the larger at position n, which is required so that the ordering of the endpoints is compatible with the results of the recursive calls.

   (b) The second if block fails (by assumption $n \geq 3$), and the else block recurses with `AWKWARDSORT2`$(A[1 \ldots n-1])$, and `AWKWARDSORT2`$(A[2 \ldots n])$

Since both of these arrays have length less than n, by the induction hypothesis `AWKWARDSORT2` correctly sorts them.

So from the first recursive call we get,

$$A[1] < A[2] < \cdots < A[n-1]$$

and from the second recursive call, we get

$$A[2] < A[3] < \cdots < A[n]$$

Combining these inequalities with $A[1] < A[n]$, we conclude that

$$A[1] < A[2] < \cdots < A[n],$$

so the entire array $A[1 \ldots n]$ is sorted. Therefore, `AWKWARDSORT2` correctly sorts arrays of length n.

Base case: If $n = 2$, there are only two cases:

(a) $A[lo] < A[hi]$, the first `if` block is skipped and since A has length 2, it is already sorted. The array is returned in the second `if` block.

(b) $A[lo] > A[hi]$, the first `if` block swaps the two integers, and returns the swapped array (in the second `if` block).

Hence, the base case also holds. And `AWKWARDSORT2` correctly sorts all arrays, i.e. $P(A[lo \ldots hi])$ is true.