Using induction to design and debug

(a bad sorting algorithm)

Glencora Borradaile

We are going to design and debug (using induction to help us) a (bad) in-place sorting algorithm based on the following idea for recursion:

> If we (recursively) sort the first two thirds of the array, the middle third of the resulting array are guaranteed to be in the final two thirds of the array. Likewise, if we then sort the final two thirds of the array, the middle third of the resulting array are guaranteed to be in the first two thirds of the array. Sorting the first two thirds of the array (again) should result in a completely sorted array.

Let's try to design an algorithm `sort` based on this idea and, in lockstep, try to prove its correctness using induction.

Attempt 1

```
sort(A[0,...,n-1])
    m = floor(n/3)
    sort(A[0,...,2m-1])
    sort(A[m,...,n-1])
    sort(A[0,...,2m-1])
    return
```

Claim: `sort` correctly sorts an array of length n.

Proof by induction:

Base case. Ooops ... there isn't a base case. Let's add one.

We will highlight the changes that we make to the pseudocode and the proof in blue.

Attempt 2

```
sort(A[0,...,n-1])
    if n = 1
        return
    else
        m = floor(n/3)
        sort(A[0,...,2m-1])
        sort(A[m,...,n-1])
        sort(A[0,...,2m-1])
        return
```

Claim: `sort` correctly sorts an array of length n.

Proof by induction:

Base case, `n = 1`. Since an array of length 1 is trivially sorted, the statement is true for $n = 1$.

Inductive hypothesis. `sort` correctly sorts an array of length $\leq n - 1$.

Applying the axiom of induction. So that we can use the I.H., we want to make sure that the three recursive calls are of a size strictly smaller. (Algorithmically, we want to make sure we don't have an infinite loop.) But notice that if $n = 2$, then $m = 0$ and the second recursive call is on an array of length n. You might also notice that this algorithm never makes a comparison. We can solve both issues by adding a base case!

Attempt 3

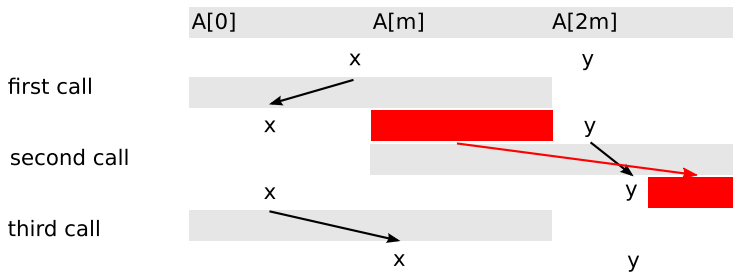
```
sort(A[0,...,n-1])
    if n <= 1
        return
    if n = 2
        if A[0] > A[1]
            swap A[0] and A[1]
        return
    else
        m = floor(n/3)
        sort(A[0,...,2m-1])
        sort(A[m,...,n-1])
        sort(A[0,...,2m-1])
        return
```

Claim: `sort` correctly sorts an array of length n.

Proof by induction: **Base case, `n <= 2`.** Since an array of length 0 or 1 is trivially sorted, the statement is true for $n \leq 1$. Clearly the algorithm correctly sorts arrays of length 2.
Inductive hypothesis. `sort` correctly sorts an array of length $\leq n-1$.
Applying the axiom of induction. The I.H. applies to all three recursive calls since $m \geq 1$ (since $n \geq 3$ when we recurse). Therefore, we may assume that after each of the recursive `sort` calls, the corresponding part of the array is sorted.

We will complete the proof with a proof by contradiction. That is, this is a proof by contradiction *within* an inductive proof. Suppose, that our algorithm is incorrect and that, after these three recursive calls, there are two numbers x and y of the array out of order: $x > y$ but x is ordered before y. Consider the three parts of the array `A1 = A[0,...,m-1], A2 = A[m,...,2m-1], A3 = A[2m,...,n-1]`.

By the correctness of the third call (using the I.H.), both x and y cannot be in `A1+A2` immediately before the third call. Therefore y must be in `A3` immediately before the third call. By the correctness of the second call (using the I.H.), both x and y cannot be in `A2+A3` immediately before the second call. Therefore x must be in `A1` immediately before the second call. By the correctness of the first call (using the I.H.), both x and y cannot be in `A1+A2` immediately before the first call. Therefore y must be in `A3` immediately before the first call. This is pictured to the right.



Also by the correctness of the first call, x must be less than all the elements in `A2` (red). Since $y < x$, y is also less than all these elements. By the correctness of the second call all the red elements must be placed after y, and so y along with all the elements in `A2` must be moved into `A3` by the second call. There are m elements in `A2` and $n - 2m - 1$ spots in `A3` (the -1 comes from the spot that y must take up). Therefore, for these red elements to fit in `A3`, we need $m \leq n - 2m - 1$ or, equivalently, $3m \leq n - 1$. Since $\frac{m=\lfloor n}{3\rfloor}$, though, this can happen. In particular, if $n = 4$, $3m = 3$ and $n - 1 = 3$. So in fact, we have shown that this *can* happen. Our algorithm must have a bug. But notice, to get this proof to work, we must make it so the red elements cannot fit into `A3`. That is, we want the middle third to be, if anything, bigger than the final third. We need to change the sizes of the recursive calls, without introducing an infinite loop, that is, while maintaining that we can still use the I.H. to argue that our three subproblems correctly sort their parts of the array. We can do so by making the first and last third of size equal to m.

Attempt 4

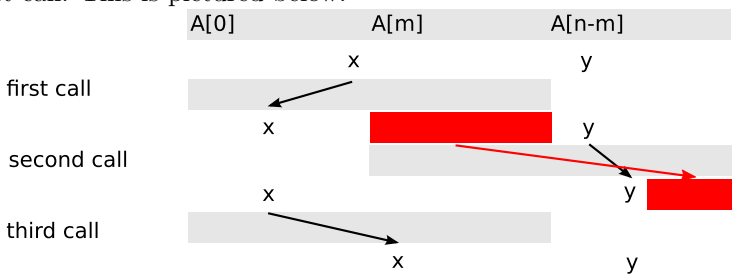
```
sort(A[0,...,n-1])
    if n <= 1
        return
    if n = 2
        if A[0] > A[1]
            swap A[0] and A[1]
        return
    else
        m = floor(n/3)
        sort(A[0,...,n-m-1])
        sort(A[m,...,n-1])
        sort(A[0,...,n-m-1])
        return
```

Claim: `sort` correctly sorts an array of length n.

Proof by induction: **Base case, `n <= 2`.** Since an array of length 0 or 1 is trivially sorted, the statement is true for $n \leq 1$. Clearly the algorithm correctly sorts arrays of length 2.
Inductive hypothesis. `sort` correctly sorts an array of length $\leq n-1$.
Applying the axiom of induction. The I.H. applies to all three recursive calls since $m \geq 1$ (since $n \geq 3$ when we recurse). Therefore, we may assume that after each of the recursive `sort` calls, the corresponding part of the array is sorted.

We will complete the proof with a proof by contradiction. Suppose, that our algorithm is incorrect and that, after these three recursive calls, there are two numbers x and y of the array out of order: $x > y$ but x is ordered before y. Consider the three parts of the array `A1 = A[0,...,m-1]`, `A2 = A[m,...,n-m-1]`, `A3 = A[n-m,...,n-1]`.

By the correctness of the third call (using the I.H.), both x and y cannot be in `A1+A2` immediately before the third call. Therefore y must be in `A3` immediately before the third call. By the correctness of the second call (using the I.H.), both x and y cannot be in `A2+A3` immediately before the second call. Therefore x must be in `A1` immediately before the second call. By the correctness of the first call (using the I.H.), both x and y cannot be in `A1+A2` immediately before the first call. Therefore y must be in `A3` immediately before the first call. This is pictured below.



Also by the correctness of the first call, x must be less than all the elements in `A2` (red). Since $y < x$, y is also less than all these elements. By the correctness of the second call all the red elements must be placed after y, and so y along with all the elements in `A2` must be moved into `A3` by the second call. There are $n-2m$ elements in `A2` and $m-1$ spots in `A3`. Therefore, for these red elements to fit in `A3`, we need $n-2m \leq m-1$ or, equivalently, $n \leq 3m-1$. But $n \geq 3m$. Combining these two inequalities gives us the contradiction $3m \leq n \leq 3m-1$. This can never happen! Therefore, our assumption that our algorithm is incorrect was wrong – our algorithm must correctly sort arrays of length n. □