

Algorithm Design and Analysis
CS 376, Colby College
An Example Proof: LLMin(L)

These notes include an example proof of correctness of an algorithm to find the minimum element in a non-empty LList that contains only integer values.

```
# Input: LList L of integers, known to be non-empty
# Output: The minimum value in L
```

Recall from class notes that every LList L is either:

- empty; or,
- composed of two parts: element $first(L)$; and LList $rest(L)$ that represents all elements on L after the first.

Description of algorithm: Because input L is non-empty, when L has only one element, it must be the minimum element on L . When L has more than one element, we'll recursively call the algorithm on $rest(L)$ to find the minimum value among all but the first element of L . Then, because the minimum value is either $first(L)$ or somewhere in $rest(L)$ —by the definition of LList, there are no other options— $\min(first(L), \min(rest(L)))$ gives the minimum value in L . Here's a recursive algorithm for that:

```
def LLMin(L):
1   if rest(L) == []
2       return first(L)
3   else
4       return min(first(L), LLMin(rest(L)))
```

There's a subtlety that comes up because the base case of this algorithm is not the base case of the definition of LList—it's worth showing that if the initial call to LLMin() meets specifications and is not empty, then LLMin() is never called on an empty LList. This is true because in the recursive case, on any input L , $rest(L)$ cannot be empty—if $rest(L)$ were empty, then the algorithm would not have entered the recursive case. Because the initial call and the recursive case's call LLMin($rest(L)$) are the only calls to LLMin(), it is never called on empty input, so this definition handles all input LLists it might receive.

A correctness argument for LLMin() can proceed by induction on the structure of the algorithm:

Base case In the base case, $rest(L)$ is empty, so $first(L)$ is the only element of L and thus the smallest. The algorithm returns $first(L)$, meeting specifications.

Inductive case As the inductive hypothesis, we assume

IH: The recursive call LLMin($rest(L)$) on line 4 of the algorithm meets specifications, returning the minimum value in $rest(L)$.

Then, we need to prove the **Claim:** In the recursive case of the code, i.e., when $rest(L)$ is not empty, $LLMin(L)$ returns the minimum value in L .

To prove this, note that in the recursive case, we know $rest(L)$ is not empty. By the reasoning in the algorithm description above, we know the minimum value in L is either the first value of L or the smallest value in $rest(L)$, whichever is less. By the IH, we know $LLMin(rest(L))$ is the smallest value in $rest(L)$, so the returned value $\min(first(L), LLMin(rest(L)))$ is the smallest value in L , so the recursive case of the algorithm meets specifications.

Termination The algorithm terminates, as the base case returns an element every time, and in the recursive case, all recursive calls are on inputs closer to the base case than the overall input—i.e., $rest(L)$ is closer to the base case than L is.