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.

```python
def LLMin(L):
    if rest(L) == []:
        return first(L)
    else:
        return min(first(L), LLMin(rest(L)))
```

Recall from class notes that every LList $L$ is either:

- empty; or,
- composed of two parts: element $\text{first}(L)$; and LList $\text{rest}(L)$ 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 $\text{rest}(L)$ to find the minimum value among all but the first element of $L$. Then, because the minimum value is either $\text{first}(L)$ or somewhere in $\text{rest}(L)$—by the definition of LList, there are no other options—$\min(\text{first}(L), \min(\text{rest}(L)))$ gives the minimum value in $L$. Here’s a recursive algorithm for that:

A correctness argument for $\text{LLMin}(\cdot)$ can proceed by induction on the structure of the algorithm:

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

**Inductive case** As the inductive hypothesis, we assume

\[ \text{IH: } \text{The recursive call } \text{LLMin}(\text{rest}(L)) \text{ on line 4 of the algorithm meets specifications, returning the minimum value in } \text{rest}(L). \]
Then, we need to prove the **Claim**: In the recursive case of the code, i.e., when $\text{rest}(L)$ is not empty, $\text{LLMin}(L)$ returns the minimum value in $L$.

To prove this, note that in the recursive case, we know $\text{rest}(L)$ 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 $\text{rest}(L)$, whichever is less. By the IH, we know $\text{LLMin}(\text{rest}(L))$ is the smallest value in $\text{rest}(L)$, so the returned value $\text{min}(\text{first}(L), \text{LLMin}(\text{rest}(L)))$ 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., $\text{rest}(L)$ is closer to the base case than $L$ is.