CSE373: Data Structures and Algorithms

Induction and Its Applications Part 3:

Proving Recursive Algorithms Correct with Induction

Steve Tanimoto

Autumn 2016

This lecture material is based in part on materials provided by Ioana Sora at the Politechnic University of Timisoara.

Lecture Outline

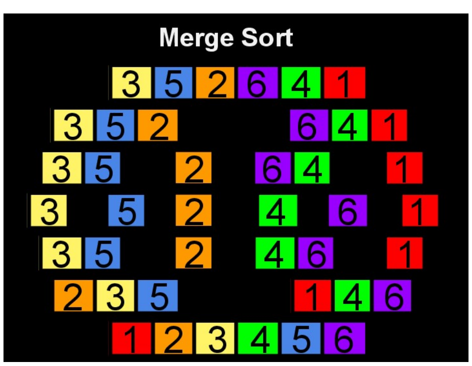
- **Proving the Correctness of Recursive Algorithms.**
- **Induction hypothesis: the recursive calls are correct.**
- **Example: Merge Sort.**
- **The General Method for Induction with Recursive Algorithms.**

Recursion is essential in applications such as creating regular fractal images.

Sierpinski-triangle bedecked umbrella image by Paul Scott.

Proof of Correctness for Recursive Algorithms

- In order to prove recursive algorithms, we have to:
  1. Prove the partial correctness (the fact that the program behaves correctly).
     - *We assume that all recursive calls with arguments that satisfy the preconditions behave as described by the specification, and use it to show that the algorithm behaves as specified.*
  2. Prove that the program terminates.
     - Any chain of recursive calls eventually ends and all loops, if any, terminate after some finite number of iterations.

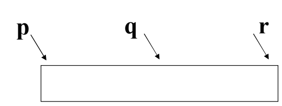


The Merge Sort algorithm is an excellent example of a recursive algorithm.

Image courtesy of cs60.net at Harvard Univ.

Example - Merge Sort

```
MERGE-SORT(A,p,r)
   if p < r
        q = (p+r)/2
        MERGE-SORT(A,p,q)
        MERGE-SORT(A,q+1,r)
        MERGE(A,p,q,r)
```



Precondition:

Array A has at least 1 element between indexes p and r ($p \leq r$).

Postcondition:

The elements between indexes p and r are sorted.

Example - Merge Sort

- MERGE-SORT calls a function MERGE(A,p,q,r) to merge the sorted subarrays of A into a single sorted one.
- The proof of MERGE (which is an iterative function) can be done separately, using loop invariants.
- We assume here that MERGE has been proved to fulfill its postconditions (can do it as a distinct exercise).

```
MERGE (A,p,q,r)

Precondition: A is an array
   and p, q, and r are
   indices into the array
   such that p ≤ q < r.
   The subarrays A[p .. q]
   and A[q+1 .. r] are sorted
Postcondition: The subarray
   A[p..r] is sorted
```


Correctness proof for Merge-Sort

- Number of elements to be sorted: n = r-p+1
- **Base Case**: n = 1
  - A contains a single element (which is trivially “sorted”).
- **Inductive Hypothesis:**
  - Assume that MergeSort correctly sorts n=1, 2, ..., **k** elements.
- **Inductive Step:**
  - Show that MergeSort correctly sorts n = **k + 1** elements.
  - First recursive call n1=q-p+1=(k+1)/2 ≤ k ⇒ subarray A[p .. q] is sorted
  - Second recursive call n2=r-q=(k+1)/2 ≤ k ⇒ subarray A[q+1 .. r] is sorted
  - A, p, q, r fulfill now the precondition of Merge
  - The postcondition of Merge guarantees that the array A[p .. r] is sorted ⇒ postcondition of MergeSort

Correctness proof for Merge-Sort

- **Termination:**
  - To argue termination, we have to find a quantity that decreases with every recursive call: the length of the part of A considered by a call to MergeSort
  - For the base case, we have a one-element array. the algorithm terminates in this case without making additional recursive calls.

Correctness proofs for recursive algorithms

```
RECURSIVE(n) is
   if (n=small_value)
      return simple_answer
   else
        RECURSIVE(n1)
        ...
        RECURSIVE(nr)
        some_code
```

$n_1, n_2, \ldots n_r$ are some values smaller than n but bigger than small_value

- **Base Case**: Prove that RECURSIVE works for n = small_value
- **Inductive Hypothesis (strong induction form):**
  - Assume that RECURSIVE works correctly for n=small_value, ..., **k**
- **Inductive Step:**
  - Show that RECURSIVE works correctly for n = **k + 1**

Lecture (Parts 1, 2, and 3) Summary

- Proving that an algorithm is totally correct means:
  1. Proving that it will ***terminate***
  2. Proving that the list of ***actions*** applied to the input (satisfying the ***precondition***) imply the output satisfies the ***postcondition.***
- How to prove ***repetitive algorithms*** correct:
  - ***Iterative*** algorithms: use ***Loop invariants***, Induction
  - ***Recursive*** algorithms: use induction using as hypothesis the recursive call

Bibliography

- Weiss, Ch. 1 section on induction.
- Goodrich and Tamassia: Induction and loop invariants; see, e.g., http://www.cs.mun.ca/~kol/courses/2711-w09/Induction.pdf)
- Erickson, J. Proof by Induction. Available at: http://jeffe.cs.illinois.edu/teaching/algorithms/notes/98-induction.pdf