Correctness

Definition 2.1. *Algorithm Correctness.*

Break into the key parts...

- Inputs
- Outputs
- Preconditions (Restrictions on the inputs)
- Postconditions (Restrictions on the outputs)
- Step-by-step process specification

(Simple version of code to know what each step is, but not complicated enough to break)

We will then look at each step to ensure that the output will give us what we want.

Correct Algorithms:

On any correct input data. (Correct by precondition)

- The program must terminate
  - Produce correct output. (Correct by postcondition)

Not all algorithms can be proven corerct, because some algorithms cannot be proved to terminate. This used for only basic algorithms.

Example 2.2. Consider the function...

$swap1(x, y)$

$aux := x$

$x := y$

$y := aux$

Precondition:

$x = a$ and $y = b$

Precondition:

$x = b$ and $y = a$.

And prove it correct

Proof. By the precondition, $x = a$ and $y = b$.

By the first step, $aux := x$, $aux = a$.

By the next step, $x := y$, so $x = b$.

And then, $y := aux$, so $y = a$.

Hence, the postconditions are met because $x = b$ and $y = a$.

Therefore, the algorithm terminates and is correct.

□

Definition 2.3. A *Loop Invariant* is a logical predicate, that if true before any single iteration, then it is also true for all subsequent iterations. This idea is useful when perfoming induction when determing complexity of loops. This turns out to the inductive hypothesis.

Example 2.4. Consider the function...

$func(a)$:

$s := 0$
$k := 0$
$while(k < n)do$
$\quad s := s + a[k]$
$\quad k := k + 1$
end

Prove that this function is correct.

Proof. by induction.

Base case: $k = 0$ and $s = 0$. Hence, s is the sum of the first zero numbers of a.

Inductive Assumption: Assume that at step $k = n$, $s = \sum_{i=0}^{n} a[i]$.

Inductive Step: Prove that at step $k = n + 1$, $s = \sum_{i=0}^{n+1} a[i]$.

$k = n$. Hence, by our assumption, $s = \sum_{i=0}^{n} a[i]$. So by the function definition, $k := k+1$ so $k := n + 1$.

So by definition of the function, $s := s + a[k]$. Hence, $s = \sum_{i=0}^{n} a[i] + a[n+1]$. Therefore, by definition of summation, $s = \sum_{i=0}^{n+1} a[i]$.

□

Definition 2.5. *Strong Induction* is when the hypothesis of induction is that for all $k \leq n$, the assumption holds true.

Example 2.6. Consider the function...

$merge\_sort(A, p, r)$
$\quad$ if$(p < r)$
$\quad\quad q = (p + r)/2$
$\quad\quad merge\_sort(A, p, q)\ merge\_sort(A, q + 1, r)\ merge(A, p, q, r)$

Precondition: A has at least 1 elment between indicies p and r.

Postcondition: The elements between p and r are sorted.

Proof. of correctness.

First we will prove that the postcondition is always fufilled for any p and r.

Let $n = r - p$.

Consider the base case: $p = r$. Then the set of values between p and r has one element. Hence, it is sorted.

Inductive Assumption: Assume merge sort corrected sorted $1 \ldots k$ elements.

Inductive Step:

Recursive call 1, attempts to sort $A[p \ldots q]$. This is $(k + 1)/2$ elements which is less than k. Hence, $A[p \ldots q]$ is sorted.

Recursive call 2, attempts to sort $A[q \ldots r]$. This is $(k + 1)/2$ elements which is less than k. hence, $A[q \ldots r]$ is sorted.

This fufills the precondition for merge, hence $merge(A, p, q, r)$ sorts $A[p \ldots q]$.

Therefore, A is k sorted elements.

Hence, $merge\_sort$ can sort an arbitrary integer number of elements and the postcondition is met.

Proof that the algorithm terminates.

We will induct on the length between p and r yet again to show that for any length of array the function returns.

Base Case: $n \leq 1$. Then $p \geq r$. Hence, the program terminates by definition.

Inductive Assumption: Assume that a call of $merge\_sort$ terminates for $r - p \leq n$.

Inductive Step: ...

□