I**terative Algorithm**

An algorithm that successively generate approximate solution for a problem until a full solution has been reached. Hence the following questions follow naturally when designing an iterative algortihm:

1. A what point can we start the approximation process?
2. After each approximation “round”, are we any closer to the solution?
3. After each approximation round, is what we have done so far correct?
4. what constitutes an approximation round?
5. when can we stop, the exit condition?

**Steps to construct an iterative algorithm**

1. define the problem
2. define loop invariant
   1. what facts must hold true before the first iteration, and before any subsequent iteration: start of each loop
   2. loop invariant shall help to prove algorithm is correct so far
3. define making progress
   1. are we any closer to the destination?
   2. usually return an integer that show
      1. how much work has been done so far
      2. how much work left to be done
   3. a progress shall focus on the algorithm state transitions:
      1. from old state to a new state where it is closer to a solution
4. define step
   1. how to make a state transition
5. define exit condition
   1. when to stop? solution has been found. no solution will be found
6. (Implementation)
   1. maintain loop invariant
7. make progress
8. initial condition
   1. what are legal input instances,
   2. what clean up need to do to ensure loop invariant
9. ending

**Code Structure**

begin routine

<pre condition>

<pre loop code>

loop

<loop invariant>

exit when <exit condition is true>

<step>

end loop

<post loop clean up>

<post condition>

end routine

**Iterative Algorithm Types**

|  |  |  |
| --- | --- | --- |
| More of input | construct solution one at a time | make progress: amount of output is constructed  LI: output constructed so far is correct |
| More of output | process input one at a time | make progress: amount of input considered  LI: pretending the input considered so far as a whole, the solution so far is correct |
| Narrow search space | use in searching for an item | make progress: the search space is reduced  LI: if the item to search for is here, then it is in the narrowed search space  (note: for binary search on unsorted list, reducing the search space let us make progress, but it never pass the LI, because we still don’t know which half the item is in. What LI does check for this type of thing) |
| Work done | creative measurement of work done |  |
| Case analysis | what cases work, what don’t |  |