Week 3 Preparation

(Solutions)

> **Useful advice:** The following solutions pertain to the preparation problems. You are strongly advised to attempt the problems thoroughly before looking at these solutions. Simply reading the solutions without thinking about the problems will rob you of the practice required to be able to solve complicated problems on your own. You will perform poorly on the exam if you simply attempt to memorise solutions to these problems. Thinking about a problem, even if you do not solve it will greatly increase your understanding of the underlying concepts.

Problem 1. Show the steps taken by radix sort when sorting the integers 4329, 5169, 4321, 3369, 2121, 2099.

Solution

Step 1: 432**1**, 212**1**, 432**9**, 516**9**, 336**9**, 209**9**.

Step 2: 43**2**1, 21**2**1, 43**2**9, 51**6**9, 33**6**9, 20**9**9.

Step 3: 2**0**99, 2**1**21, 5**1**69, 4**3**21, 4**3**29, 3**3**69.

Step 4: **2**099, **2**121, **3**369, **4**321, **4**329, **5**169.

Problem 2. Consider the following algorithm that returns the number of occurrences of *target* in the sequence *A*. Identify a useful invariant that is true at the beginning of each iteration of the **while** loop. Prove that it holds, and use it to prove that the algorithm is correct.

```
function COUNT(A[1..n], target)
    count = 0
    i = 1
    while i ≤ n do
        if A[i] = target then
            count = count + 1
        end if
        i = i + 1
    end while
    return count
end function
```

Solution

A useful invariant is that, at the start of iteration i, `count` is equal to the number of occurrences of `target` in $A[1..i-1]$, where we consider $A[1..0]$ to be an empty list.

Note: To prove this loop invariant, we will use induction. First we show that the invariant holds at initialisation, at the start of the first iteration of the loop. This is our base case. Next we assume that the invariant holds at the start of some iteration of the loop, and show that it still holds at the start of the next iteration. At this point we are done, since we have shown that the invariant holds at the start of the first loop, and that if it holds at the start of loop i, it also holds at the start of loop $i+1$. This means it holds at the start of every loop, and importantly, that it holds at the start of the loop where the loop condition is false, i.e., it holds when the loop ends.

Proof: At the start of the first iteration, $i = 1$. Also, $\texttt{count} = 0$, so `count` is equal to the number of occurrences of `target` in $A[1..0]$, since $A[1..0]$ is an empty list. So the invariant is true at initialisation.

Assume that the invariant holds at the start of k-th iteration. So `count` is equal to the number of occurrences of `target` in $A[1..k-1]$. Call this number of occurrences c. During this k-th iteration of the loop, if $A[k] = \texttt{target}$, we will increment `count`, so `count` will equal $c+1$, which is the number of occurrences of `target` in $A[1..k]$. If $A[k] \neq \texttt{target}$, then `count` will not be changed, so `count` will equal c, which is equal to the number of occurrences of `target` in $A[1..k]$. Either way the invariant holds at the start of $k+1$-th iteration, that is, `count` is equal to the number of occurrences of `target` in $A[1..k]$. Since we know the invariant holds at the start, by induction it holds for all values of i, including when $i = n+1$, so the invariant holds.

To prove the algorithm is correct, we need to show that at loop termination, `count` is equal to the number of occurrences of `target` in A. The invariant tells us that `count` is equal to the number of occurrences of target in $A[1..i-1]$, but at loop termination, $i = n+1$, so `count` is equal to the number of occurrences of target in $A[1..n]$ which is all of A. Therefore the algorithm is correct.