Computer Science 331

Solutions to Selected Tutorial #4 Questions

Question 1

```
 1 for i from 0 to n-2 do
 2     j := n-1
 3     while j > i do
 4         if A[j] < A[j-1] then
 5             temp := A[j]
 6             A[j] := A[j-1]
 7             A[j-1] := temp
 8         end if
 9         j := j-1
10     end do
11  end do
```

a. Upper bound for the worst-case running time of this algorithm

To find the worst-case run-time of this algorithm, we first determine the worst-case run-time of the inner-most loop and work outwards. Consider the body of the inner-most while loop first. As in class, we will approximate a run time function by counting the number of operations executed during the course of the algorithm. Since we are finding an upper bound for the worst-case, we assume that test condition in the if statement evaluates to 'true' and all the statements will be executed during every iteration.

```
                                  Cost

4  if A[j] < A[j-1] then         2 ops (sub. and comparison)
5     temp := A[j]               1 op  (assignment)
6     A[j] := A[j-1]             2 ops (sub. and assignment)
7     A[j-1] := temp             2 ops (sub. and assignment)
8  end if
```

```
9  j := j-1                          2 ops (sub. and assignment)
```

The loop variant for this while loop is $f(i,j) = j - i$. Substituting in the initial value of $j = n-1$ tells us that the number of iterations of this loop is at most $n-i-1$. The number of operations done in the body of this while loop per execution is at most 9. In addition to the $(n-i-1)$ executions of the loop body, the test condition $j > i$ is executed $(n-i)$ times. The worst-case run-time of the code segment between $3-10$ (the inner while loop) is thus

$$\begin{aligned} T_1(i) &= (n-i) + 9(n-i-1) \\ &= 10(n-i) - 9 \ . \end{aligned}$$

Notice that the cost of this loop for the ith iteration is expressed as a function of the outer loop index i.

The loop variant for the outer-most for-loop is $f(i,n) = n-1-i$. Substituting in the initial value of $i = 0$ tells us that the number of iterations of this loop is at most $n-1$. The number of operations done in the body of this for-loop per execution is at most $T_1(i)+2$ where one additional operation comes from the initialization of variable j (line 2) and an additional operation comes from incrementing i. In addition to the $(n-1)$ executions of the loop body, the test condition ($i \leq n-2$) is executed n times, and each evaluation of the test costs 2 operations in total (one subtraction and one comparison). Before the program enters the for-loop, i is initialized to 0, which counts as 1 operation. Thus, the worst-case run-time is equal to the initialization cost (1) plus the cost for evaluating the tests ($2n$) plus the cost of executing the loop body when $i = 0, 2, \ldots, n-2$ and we obtain

$$\begin{aligned} T(n) &= 1 + 2n + (T_1(0)+2) + (T_1(1)+2) + \cdots + (T_1(n-2)+2) \\ &= 1 + 2n + \sum_{i=0}^{n-2} (T_1(i)+2) \ . \end{aligned}$$

As we are looking for an upper bound on the worst-case running time, we note that for $i = 0, \ldots, n-2$, taking $i = 0$ results in the largest value for $T_1(i)$, so we have

$$T_1(i) \leq T_1(0) = 10(n-0) - 9 = 10n - 9 \ .$$

Thus, we have that

$$\begin{aligned} T(n) &\leq 1 + 2n + \sum_{i=0}^{n-2} (T_1(0)+2) \\ &= 1 + 2n + \sum_{i=0}^{n-2} (10n - 7) \\ &= 1 + 2n + (n-1)(10n-7) \\ &= 10n^2 - 15n + 8 \end{aligned}$$

is an upper bound on the worst-case run time of bubble sort.

Alternative Solution: For this example, it is possible to derive a more precise operation count by evaluating the sum as opposed to using an upper bound on its terms. In particular, we have

$$\begin{aligned}
T(n) &= 1 + 2n + \sum_{i=0}^{n-2} (T_1(i) + 2) \\
&= 1 + 2n + \sum_{i=0}^{n-2} (10(n-i) - 9 + 2) \\
&= 1 + 2n + 10 \left(\sum_{i=0}^{n-2} (n-i) \right) - \left(\sum_{i=0}^{n-2} 7 \right) \\
&= 1 + 2n + 10 \left(\sum_{i=0}^{n-2} n \right) - 10 \left(\sum_{i=0}^{n-2} i \right) - \left(\sum_{i=0}^{n-2} 7 \right) \\
&= 1 + 2n + (10(n-1)n) - 10 \left(\frac{(n-1)(n-2)}{2} \right) - 7(n-1) \\
&= 5n^2 - 2 \ .
\end{aligned}$$

Notice that, for this example, both methods yield functions that are quadratic in the input size n. Looking ahead, we see that, using asymptotic notation, we can say that the worst-case running time of bubble sort in in $O(n^2)$. In addition, using the fact that the second expression for $T(n)$ (obtained by evaluating the sum as opposed to approximating it), serves as an upper and lower bound on the worst-case run time, we can conclude that the worst-case run time of bubble sort is in $\Theta(n^2)$.

b. Lower bound for the best-case running time of this algorithm

To find a lower bound for the best-case run-time of this algorithm, we first determine the best-case run-time of the inner-most loop and work outwards. In the best case the test condition in the if statement will evaluate to "false" and the number of operations done in the body of the while loop per execution will be 4. Thus, following the reasoning from the previous question, the number of steps required to execute the inner while-loop in the best case is

$$\begin{aligned}
T_1(i) &= (n-i) + 4(n-i-1) \\
&= 5(n-i) - 4 \ .
\end{aligned}$$

Notice that the behaviour of the outer for-loop is the same in the worst and best cases, so the total cost of bubble sort is

$$\begin{aligned}
T(n) &= 1 + 2n + (T_1(0) + 2) + (T_1(1) + 2) + \cdots + (T_1(n-2) + 2) \\
&= 1 + 2n + \sum_{i=0}^{n-2} (T_1(i) + 2) \ .
\end{aligned}$$

As we are looking for a lower bound on the best-case running time, we note that for $i = 0, \ldots, n-2$, taking $i = n-2$ results in the smallest value for $T_1(i)$, so we have

$$T_1(i) \geq T_1(n-2) = 5(n-(n-2)) - 4 = 6 \ .$$

Thus, we have that

$$\begin{aligned} T(n) &\geq 1 + 2n + \sum_{i=0}^{n-2} (T_1(n-2) + 2) \\ &= 1 + 2n + \sum_{i=0}^{n-2} 8 \\ &= 1 + 2n + 8(n-1) \\ &= 10n - 7 \end{aligned}$$

is a lower bound on the best-case run time of bubble sort. Using asymptotic notation, we can state that the best-case run time of bubble sort is in $\Omega(n)$.

However, it is important to note that it is possible to obtain a tighter lower bound on the best case run time for this example. If we evaluate the sum exactly as opposed to using a lower bound on the terms in the sum, we obtain

$$\begin{aligned} T(n) &= 1 + 2n + \sum_{i=0}^{n-2} (T_1(i) + 2) \\ &= 1 + 2n + \sum_{i=0}^{n-2} (5(n-i) - 2) \\ &= 1 + 2n + 5\left(\sum_{i=0}^{n-2} n\right) - 5\left(\sum_{i=0}^{n-2} i\right) - \left(\sum_{i=0}^{n-2} 2\right) \\ &= 1 + 2n + 5(n-1)n - 5\frac{(n-1)(n-2)}{2} - 2(n-1) \\ &= \frac{5}{2}n^2 - \frac{5}{2}n - 2 \ . \end{aligned}$$

Thus, we can also conclude, using asymptotic notation, that the best-case run time of bubble sort is in $\Omega(n^2)$. Note that this does not contradict the previous claim that the best case run time is in $\Omega(n)$ — both statements are correct, but the $\Omega(n^2)$ result is more precise. In fact, given these results, we can also conclude that the best-case run time of bubble sort is in $\omega(n)$ and in $\Theta(n^2)$. Combining with the answer to the previous question, we have that the best and worst-case run times of this version of bubble sort are in $\Theta(n^2)$, i.e., the best case does *not* offer a significant improvement over the worst-case.

Note: It is possible to modify this version of bubble sort in such a way that the best case *is* in $\Theta(n)$. As an exercise, think about how this can be done and what characterizes input arrays that give rise to this best case performance.

c: Average case analysis

Given that the worst and best case running times are quadratic functions of n, we conclude that the average case, which must lie between the best and worst cases, is given by a quadratic function of n.

d: Putting assertions

The most helpful assertions to assist in proving bubble sort partially correct are loop invariants corresponding to each of the loops and a post-condition for the inner while-loop. This post-condition, as well as the loop invariants $I(k)$ (corresponding to the outer for-loop) and $I(t)$, corresponding to the inner while-loop are included below.

```
for i from 0 to n-2 do
   // I(k) : i = k; A[l] <= A[l+1] for 0<=l<i-1; 0<=i<n
   j := n-1
   while j > i do
       // I(t): j=n-1-t; A[j] <= A[m] for j<m<n; i<=j<n
       if A[j] < A[j-1] then
           temp := A[j]
           A[j] := A[j-1]
           A[j-1] := temp
       end if
       j := j-1
   end do
   // Post-condition:  A[i] <= A[m] for i<m<n
end do
// Post-condition:  A[i] <= A[i+1] for 0 <= i < n
```

To verify that the second loop invariant (corresponding to the inner loop) is valid, you would need to show (exercise) that:

- $I(t)$ is true when $t = 0$ (i.e., before the first execution of the loop body). With a while-loop, you should imagine the loop invariant being placed immediately before the test for termination.
- If $I(t)$ is true and the loop iterates again, then $I(t + 1)$ is true.

- If $I(t)$ is true and the loop terminates, the the loop's post-condition is satisfied.

Similarly, to verify that the first loop invariant is valid, you would need to show (exercise) that:

- $I(k)$ is true when $k = 0$ (i.e., before the first execution of the for-loop body). With a for-loop, you should imagine the loop invariant being placed immediately after i is initialized but before the test for termination.
- If $I(k)$ is true and the loop iterates again, then $I(k + 1)$ is true. Notice that you need to assume that the inner loop's post-condition is always satisfied in order to prove this, but proving the partial correctness of the inner loop, by demonstrating that $I(t)$ is a valid loop invariant as above, assures that this is in fact true.
- If $I(k)$ is true and the loop terminates, then the post condition (in this case, that the elements of A are sorted) is satisfied.

Thus, a proof, as outlined above, that these loop invariants are correct would constitute a proof of partial correctness of bubble sort.