Weekly Assignment 3

Munia Humaira
Student ID: 21116435
Algorithm Design and Analysis

September 27, 2024

Solution 1:

The correctness of the given bubble sort algorithm can be proved using loop invariant and induction. It can be proved that after each iteration of the outer **foreach** loop, the last k elements are in their correct sorted positions.

Proof by induction:

1. **Base case** ($i = 1$):
   After the first iteration of the outer **foreach** loop, the largest element will be "bubbled up" to the last position. This is because the inner **foreach** loop of the algorithm compares adjacent elements and swaps them if they are out of order, ensuring the largest element reaches at the end of the array.

2. **Induction step:**
   Let's assume the invariant holds for some k, where $1 \leq k < n - 1$. We need to prove it holds for $k + 1$.

   At the beginning of the k-th iteration of the outer **foreach** loop, the largest k elements are already sorted in the last k indices of the array. The inner loop will bubble up the largest element among the first $n - k$ elements to the position $n - k - 1$. It is guaranteed that the element is smaller than or equal to the elements in the last k positions. Therefore, after $k + 1$-th iteration, the last $k + 1$ elements should be in their final sorted position in the array A.

3. **Termination**
   The outer loop terminates after *n-1* iterations. At this stage, the loop invariant ensures that $A[2....n]$ contains the $n - 1$ largest elements in the sorted order. Therefore, the smallest element must be in $A[1]$ which means the entire array A is sorted in ascending order.

Solution 2:

a) Worst-case time-efficiency

To calculate the worst-cast complexity of the bubble sort, we need to count the maximum number of comparisons and swaps are performed in the algorithm.
The nested loop structure look like the following

```
foreach i from 1 to n − 1 do
    foreach j from 1 to n − i do
        // comparison and potential swap
```

In each iteration of the inner **foreach** loop, we perform one comparison ($A[j] > A[j+1]$). The number of comparisons in each outer **foreach** loop iteration:

1st iteration $(i = 1) : n - 1$ comparisons

2nd iteration $(i = 2) : n - 2$ comparisons

3rd iteration $(i = 3) : n - 3$ comparisons

...

$(n - 1)$th iteration $(i = n - 1) : n - (n - 1) = 1$ comparison

The total number of comparisons should be the sum of the comparisons of each level:

$$(n-1) + (n-2) + (n-3) + (n-4) + ... + 3 + 2 + 1$$

This sum can be expressed as:

$$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

In the worst-case, when the array is in reverse order, every comparison will lead to a swap and hence, the number of swaps should be equal to the number of comparisons.

$$\begin{aligned} TotalOperations &= Comparisons + Swaps \\ &= \frac{n(n-1)}{2} + \frac{n(n-1)}{2} \\ &= n(n-1) \\ &= n^2 - n \end{aligned}$$

As in $\mathcal{O}$ notation, we only consider the highest order term and drop the constants therefore the time complexity of the worst-case scenario of bubble sort becomes

$$n^2 - n = \mathcal{O}(n^2)$$

Therefore, the worst-case time-efficiency of bubble sort is $\mathcal{O}(n^2)$

b) Average-case time-efficiency calculation

For average-case time complexity, we are assuming that every permutation of $A[1], ..., A[n]$ is equally likely on input which means there can be $n!$ of possible permutations. As we compare adjacent elements; for any pair of elements (i, j), where $i < j$, we need to determine the probability of they will be compared.

For elements in positions i and j to be compared, all elements between them must not cause them to be swapped earlier. The probability of this happening is 1/2 as there is equal chance of any element being greater or smaller than another in a random permutation.

For each pair (i, j), the expected number of comparison should be:

$$E[comparision for i, j] = \frac{1}{2}$$

The total expected number of comparisons should be the sum of all possible pairs:

$$E[total comparisions] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} = \frac{1}{2}.\frac{n(n-1)}{2} = \frac{n(n-1)}{4}$$

The probability of swap for any comparison is also $\frac{1}{2}$ in a random permutation and hence, the expected number of swaps is half of the expected number of comparisons i.e.

$$E[total\ swaps] = \frac{1}{2}.\frac{n(n-1)}{4} = \frac{n(n-1)}{8}$$

So the total expected operations should be:

$$\begin{aligned} Total\ Operations &= Comparisons + Swaps \\ &= \frac{n(n-1)}{4} + \frac{n(n-1)}{8} \\ &= \frac{3n(n-1)}{8} \end{aligned}$$

As per the rule of $\mathcal{O}$ notation mentioned in previous solution, the time complexity of this scenario of bubble sort becomes

$$\frac{3n(n-1)}{8} = O(n^2)$$

Therefore, the average-case time-efficiency of bubble sort is $\mathcal{O}(n^2)$