

# BLG 335E – Analysis of Algorithms I

## Fall 2017, Recitation 1

26.09.2017

R.A. Doğan Altan  
daltan@itu.edu.tr

R.A. Umut Sulubacak  
sulubacak@itu.edu.tr

Prepared by Atakan Aral & Mustafa Ersen



- Slightly modified version of **Problem 2.2** in *Introduction to Algorithms*, T.H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, MIT Press, 2001
- We will:
  - Write *pseudocode* for the given algorithm
  - Prove that the algorithm actually solves the problem (using *loop invariants*)
  - Compute the *worst case running time* of the algorithm



# Bubble Sort Algorithm

---

6 5 3 1 8 7 2 4

<http://bit.ly/VScGXL>



# Writting the Pseudocode

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```



# Question 1

- What do we need to prove to show that Bubble sort actually sorts?
- We need to prove that:
  - it terminates
  - the elements of  $A'$  form a permutation of the elements of  $A$
  - $A'[1] \leq A'[2] \leq \dots \leq A'[n]$  where  $n = \text{length}[A]$



# Question 2

- State precisely a loop invariant for the for the inner loop, and prove that this loop invariant holds.

```
for  $i \leftarrow 1$  to  $length[A]$   
  { do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then  $exchange\ A[j] \leftrightarrow A[j - 1]$ 
```

# Question 2 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Loop invariant:**

- At the start of each iteration of the inner **for** loop:

- $A[j] = \min \{ A[k] : j \leq k \leq n \}$
- the sub array  $A[j..n]$  is a permutation of the values that were in  $A[j..n]$  at the time that the loop started.

# Question 2 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Initialization:**

- Initially,  $j = n$ , and the sub array  $A[j .. n]$  consists of single element  $A[n]$ .
- The loop invariant trivially holds.





# Question 2 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$ 
  do for  $j \leftarrow length[A]$  downto  $i + 1$ 
    do if  $A[j] < A[j - 1]$ 
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Maintenance (1/3):**
  - Consider an iteration for a given value of  $j$ .
  - $A[j]$  is the smallest value in  $A[j .. n]$ .
  - After the exchange,  $A[j - 1]$  will be the smallest value in  $A[j - 1 .. n]$



# Question 2 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Maintenance (2/3):**
  - We know that:
    - Rest of the sub array remains the same
    - The sub array  $A[j .. n]$  is a permutation of the values that were in  $A[j .. n]$



# Question 2 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Maintenance (3/3):**

- We see that:

- $A[j - 1 .. n]$  is a permutation of the values that were in  $A[j - 1 .. n]$  at the time that the loop started
    - Decrementing  $j$  for the next iteration maintains the invariant



# Question 2 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$ 
  do for  $j \leftarrow length[A]$  downto  $i + 1$ 
    do if  $A[j] < A[j - 1]$ 
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Termination:**
  - The loop terminates when  $j$  reaches  $i$
  - Loop invariant holds



# Question 3

- Using the termination condition of the loop invariant proved in *Question 2*, state a loop invariant for the for the outer loop that will allow you to prove:
  - $A'[1] \leq A'[2] \leq \dots \leq A'[n]$  where  $n = \text{length}[A]$

```
for  $i \leftarrow 1$  to  $\text{length}[A]$   
  do for  $j \leftarrow \text{length}[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

# Question 3 (Solution)

```
for  $i \leftarrow 1$  to  $\text{length}[A]$   
  do for  $j \leftarrow \text{length}[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Loop invariant:**
  - At the start of each iteration of the outer loop:
    - the sub array  $A[1 .. i-1]$  consists of the  $i-1$  smallest values originally in  $A[1 .. n]$ , in sorted order
    - $A[i .. n]$  consists of the  $n-i+1$  remaining values originally in  $A[1 .. n]$ .



# Question 3 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Initialization:**

- Before the first iteration of the loop:

- $i = 1$
- The sub array  $A[1 .. i - 1]$  is empty
- the loop invariant vacuously holds



# Question 3 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Maintenance (1/2):**
  - Consider an iteration for a given value of  $i$ 
    - $A[1 .. i - 1]$  consists of the  $i$  smallest values in  $A[1 .. n]$ , in sorted order
  - In question 2 we have showed that:
    - after executing the outer loop,  $A[i]$  is the smallest value in  $A[i .. n]$



# Question 3 (Solution)

```
for  $i \leftarrow 1$  to  $\text{length}[A]$   
  do for  $j \leftarrow \text{length}[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Maintenance (2/2):**

- As a result,

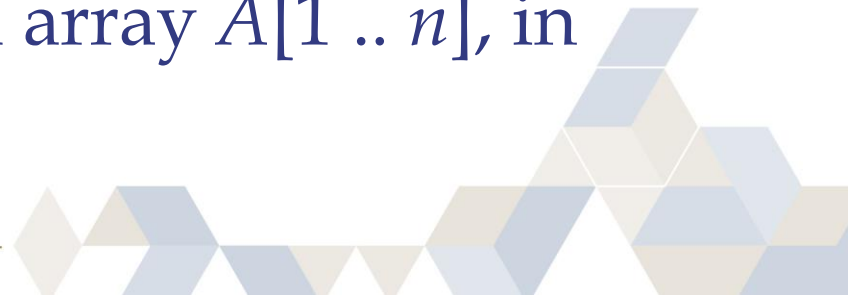
- $A[1 .. i]$  is now the  $i$  smallest values originally in  $A[1 .. n]$ , in sorted order
    - since the inner loop permutes  $A[i .. n]$ , the sub array  $A[i + 1 .. n]$  consists of the  $n - i$  remaining values originally in  $A[1 .. n]$

# Question 3 (Solution)

```
for  $i \leftarrow 1$  to  $\text{length}[A]$   
    do for  $j \leftarrow \text{length}[A]$  downto  $i + 1$   
        do if  $A[j] < A[j - 1]$   
            then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- **Termination:**

- The outer loop terminates when  $i = n+1$ , so that  $i - 1 = n$
- $A[1 .. i - 1]$  is the entire array  $A[1 .. n]$
- it consists of the original array  $A[1 .. n]$ , in sorted order



# Question 4

---

- What is the worst-case running time of bubble sort?
- How does it compare to the running time of insertion sort and merge sort?



# Question 4 (Solution)

```
for  $i \leftarrow 1$  to  $length[A]$   
  do for  $j \leftarrow length[A]$  downto  $i + 1$   
    do if  $A[j] < A[j - 1]$   
      then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

- For a given  $i$ , the inner loop makes  $n - i$  iterations. The total number of iterations, therefore, is:

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

# Question 4 (Solution)

- The running time of bubble sort is  $\Theta(n^2)$  in all cases (best, worst, average)
- Its worst case running time is
  - worse than Merge sort which is  $O(n \log n)$
  - same as Insertion sort



- How can we optimize Bubble sort?
  - Terminate if no swaps are made in a loop step
    - Best case performance is  $O(n)$
  - After every pass, all elements after the last swap are sorted, and do not need to be checked again
    - 50% improvement in comparison counts
    - no improvement in swap counts
  - Cocktail shaker sort (aka. bidirectional bubble sort)
    - Complexity approaches to  $O(n)$  if the list is mostly ordered