A picture containing text

Description automatically generated

**CS435**

*Prof. Emdad Khan*

*September 2019*

*Lab#3*

***Group 1***

*Group members:*

*Asad Ali Kanwal*

*Aser Ahmad Ibrahim Ahmad*

*Jean Wilbert Volsy*

*Zayed Hassan*

1. Problem 1

Actual cost function:

Let amortized cost function be

⸫ amortized cost =

Amortized cost per operation =

1. Problem 2

The code is in the file: BubbleSort1.java

A Boolean flag “swapped” was added and initially set to false. Its value is controlled in the inner loop. If a swap was made, its value is set to true. Otherwise, it remains false to indicate that no swap was made, thus all elements are in sorted order.

Loop invariant for

For the best case (array is already sorted):

For : the inner loop makes *n* loops from to . So, the running time for this loop is .

For : the “swapped” variable will remain false after the first run, indicating that no sorting was needed, and the outer loop will break.

So, the running time in the best case will be the same as the running time of the first run of the outer loop, which is .

1. Problem 3:

The code is in the file: BubbleSort2.java

The termination condition of the inner loop was changed from to . This reflects the fact that after each run of the outer loop the element becomes in its final sorted position, and there is no need to check it again.

Loop invariant for

For the 1st run of the inner loop of the counter, the inner loop runs times.

For the 2nd run of the inner loop of the counter, the inner loop runs times.

For the 3rd run of the inner loop of the counter, the inner loop runs times.

Eventually, at the nth run of the inner loop, the inner loop runs 1 time only.

For the worst case (array reverse-ordered) a swap will always be performed. So, the number of runs of the inner loop is:

, which is half of the run time of the original bubble sorting algorithm.

1. Problem 4:

|  |  |
| --- | --- |
| **Algorithm** sort0\_1\_2\_array (A) | *Count of operations* |
| ***Input:*** array A with elements belonging to the set |  |
| ***Output:*** sorted array A |  |
| zeros = 0 | *1* |
| ones = 0 | *1* |
| twos = 0 | *1* |
| **for** (i = 0; i < A.length; ++i) **do** | *1 + n + 2n* |
| **switch** A[i] | *n* |
| **case** 0 | *n* |
| zeros++ | *2n* |
| break | *n* |
| **case** 1 | *n* |
| ones++ | *2n* |
| break | *n* |
| **case** 2 | *n* |
| twos++ | *2n* |
| **for** (i = 0; i ≤ zeros; ++i) **do** | *1 + n + 2n* |
| A[i] = 0 | *2n* |
| **for** (i = zeros + 1; i ≤ ones; ++i) **do** | *1 + n + 2n* |
| A[i] = 1 | *2n* |
| **for** (i = ones + 1; i ≤ twos; ++i) **do** | *1 + n + 2n* |
| A[i] = 2 | *2n* |
| return A | *1* |