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: $c(S_i) = \begin{cases} i, & i = 2^m, \quad m \in \{1, 2, 3, \dots\} \\ 1 & otherwise \end{cases}$

Let amortized cost function be $\hat{c}(S_i) = i$

$\therefore$ amortized cost = $\sum_{i=1}^{n} \hat{c}(S_i) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Amortized cost per operation = $\frac{\sum_{i=1}^{n} \hat{c}(S_i)}{n} = \frac{n+1}{2}$

2. 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 $0 \leq i \leq n-1$

$I(i)$: $arr[n-i-1] \ldots [n-1]$

For the best case (array is already sorted):

For $I(0)$: the inner loop makes n loops from $arr[0]$ to $arr[n-1]$. So, the running time for this loop is $O(n)$.

For $I(i), i > 0$: 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 $O(n)$.

3. Problem 3:

The code is in the file: `BubbleSort2.java`

The termination condition of the inner loop was changed from $j < n - 1$ to $j < n - i - 1$. This reflects the fact that after each run of the outer loop the $(n - i)^{th}$ element becomes in its final sorted position, and there is no need to check it again.

Loop invariant for $0 \leq i \leq n - 1$

$I(i): arr[n - i - 1] \dots [n - 1]$

For the 1st run of the inner loop of the counter, $i = 0$ the inner loop runs n times.

For the 2nd run of the inner loop of the counter, $i = 1$ the inner loop runs $n - 1$ times.

For the 3rd run of the inner loop of the counter, $i = 2$ the inner loop runs $n - 2$ times.

Eventually, at the nth run of the inner loop, $i = n - 1$ 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: $1 + 2 + 3 + 4 + \cdots n = \sum_{j=1}^{n} j = \frac{n(n+1)}{2}$

$\therefore f(n)\ is\ O\left(\frac{n^2}{2}\right)$, which is half of the run time of the original bubble sorting algorithm.

4. Problem 4:

```
Algorithm sort0_1_2_array (A)                          Count of operations
Input: array A with elements belonging to the set {0,1,2}
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
```

$f(n) = 4 + 3n + 2n + 3n + 1 + 3n + 2n + 1$

$f(n) = 13n + 6$

$$\lim_{n\to\infty} \frac{13n + 6}{n} = \lim_{n\to\infty} \frac{13 + \frac{6}{n}}{1} = 13 \rightarrow f(n)\ is\ O(n).$$