Umeå University
Institution för Datavetenskap

Datavetenskapens byggstenar 7.5 p
DV160HT15

OU4 Analysis of Complexity

Submitted	2015-12-25
Author:	Lorenz Gerber (`dv15lgr@cs.umu.se`)
Instructor:	Lena Kallin Westin / Johan Eliasson

Contents

1 Introduction

The aim with this laboration was to apply experimental and asymptotic complexity analysis of algorithms.

What is complexity analysis. Experimental, asymptotic. What is big O notation, what does it mean.

1.1 The 'Big O' notation'

'Ordo' or 'Big O' notation is a mathematical definition on the complexity of an algorithm. It can be written as shown in equation (1)[1, pp. 245].

$$f(n) \Rightarrow O(g(n)) \text{ if } f(n) \leq c \times g(n) \text{ for } n \geq n_0 \text{ and } c > 0 \text{ and } n_0 \geq 1 \tag{1}$$

2 Material and Methods

2.1 Experimental Complexity Analysis

Describe experiment, describe the rules.

2.2 Asymptotic Complexity Analysis

Describe what was given and the rules to analyse.

Listing 1 The given pseudo code of a bubble sort.

```
Algorithm bubbleSort(numElements, list[])
input:  numElements, the number of elements in the list
        list, a list of numbers to be sorted
output: the sorted list

    done <- false
    n <- 0
     while (n < numElements) and (done = false)
        done <- true
        for m <- (numElements -1) downto n
            if list[m] < list[m - 1] then
                tmp <- list[m]
                list[m] <- list[m - 1]
                list[m - 1] <- tmp
                done <- false
      n <- n + 1
    return list
```

3 Results

3.1 Experimental Complexity Analysis

Show formulas, C, n0

3.2 Asymptotic Complexity Analysis

Worst Case

Best case

show plot

4 Discussion

4.1 Experimental Complexity Analysis

4.2 Asymptotic Complexity Analysis

Worst Case

Line 1, 2 and 12 run just once. Line 3 runs `numElements + 1` times. Lines 4 and 11 run `numElements` times. Line 5 runs `(numElements * (numElements - 1)) / 2) + 1` times. The lines 6 to 10 run `numElements * (numElements - 1) / 2` times. The `downto` in `for m <- (numElements - 1) downto n` was interpreted as 'larger than' condition.

References

[1] L.E. Janlert and T. Wiberg. *Datatyper och algoritmer*. Studentlitteratur, 2000.

Listing 2 Determining the ‘worst case’ complexity for the given ‘bubblesort’ algorithm. The line numbers correspond to those in the listing xxx

```
1:  1 * [<-] +
2:  1 * [<-] +
3:  (numElements + 1) *
    (3 * [get] + 1 * [<] +  1 * [=] + 1 * [AND]) +
4:  numElements * (1 * [<-]) +
5:  init:
    (numElements * (numElements -1) / 2 + 1) *
    (1 * [get] + 1 * [-] + 1 * [<-]) +

    cond success + counter:
    numElements * (numElements - 1) / 2) *
    (2 * [get] + 1 * [>] + 1 * [--]) +

    cond fail:
    numElements *
    (2 * [get n] + 1 * [>]) +

6:  (numElements * (numElements - 1) / 2) *
    (2 * [get] + 2 * [list[]] + 1 * [-] + 1 * [<] +
7:      1 * [get] + 1 * [list[]] + 1 * [<-] +
8:      1 * [get] + 1 * [-] + 1 * [list[]] + 1 * [<-] +
9:      2 * [get] + 1 * [-] + 1 * [ <-] +
10:     1 * [<-] ) +
11: numElements * (1 * [get] + 1 * [+] + 1 * [<-] +
12: 1 * return

set numElements = x

1:  1 +
2:  1 +
3:  (x + 1) * 6
4:  x * 1
5:  (x * (x-1) / 2 + 1) * 3 +
    (x * (x - 1) / 2) * 4 +
    x * 3 +
6:  (x * (x - 1) / 2) * (6 +
7:    3 +
8:    4 +
9:    4 +
10:   1) +
11: x * 3 +
12: 1

Hence:
1 + 1 + 6x + 6 + x + 1.5x^2 - 1.5x + 3 + 2x^2 - 2x +
3x + 9x^2 - 9x + 3x + 1

= 12.5x^2 + 4.5x + 12
```

Listing 3 Determining the ‘best case’ complexity for the given ‘bubblesort’ algorithm. The line numbers correspond to those in listing xxx

```
1:  1 * [<-] +
2:  1 * [<-] +
3:  2 * (3 * [get] + 1 *[<] + 1 * [and] + 1* [==]) +
4:  1 * [<-] +
5:  init:
    1 * [get] + 1 * [-] + 1 * [<-] +

    cond success + counter:
    (numElements - 1) * (1 * [get] + 1 * [>]) + 1 * [--]) +

    cond fail:
    1* [get] + 1 * [>] +
6:  (numElements - 1) *
    (2 * [get] +  2 * list[] + 1 * [-] + 1 * [<]) +
11: 1 * [get] + 1 * [+] + 1 * [<-]
12: 1 * [return]

set numElements = x

1:  1 +
2:  1 +
3:  2 * 6 +
4:  1 +
5:  3 +
    (x - 1) * 3 +
    2 +
6:  (x - 1) * 6 +
11: 3 +
12: 1

Hence:
1 + 1 + 12 + 1 + 3 + 3x - 3 + 2 + 6x -6 + 3 + 1
= 9x + 15
```