CGI-ACT

July 10, 2016

Algorithms and Algorithmic Problems

Workshop Sessions

Session 1 - Introduction to Algorithms

Exercises and Solutions

1 General remarks

1. The algorithms in this document consider the first index of the array or list to be 1.

2 Exercises

Exercise 1

Determine which characteristics of an algorithm the following functions lack?

1. **function** DOUBLE(n: positive integer)
   while $n > 0$ **do**
   $n \leftarrow 2n$
   return n
2. **function** CHOOSE(a, b: integers)
   $x \leftarrow$ either a or b
   return x

Exercise 2

Determine whether each of these functions is $O(x^2)$, $\Omega(x^2)$, and whether it is $\Theta(x^2)$.

1. $f(x) = 17x + 11$
2. $f(x) = x^2 + 1000$

3. $f(x) = x \log x$
4. $f(x) = x^4/2$
5. $f(x) = 2^x$
6. $f(x) = \lfloor x \rfloor \cdot \lceil x \rceil$

Exercise 3

Which Big-O estimate applies to the worst-case running time of the following algorithm? (For the function g in your estimate $O(g(x))$, use a simple function g of smallest order.)

```
procedure SORT(A: array of integers)
    n ← A.length
    for i ← 1, n − 1 do
        for j ← 1, n − i do
            if A[j] > A[j + 1] then
                exchange A[j] with A[j + 1]
```

Exercise 4

Which Big-O estimate applies to the worst-case running time of the following algorithm? (For the function g in your estimate $O(g(x))$, use a simple function g of smallest order.)

```
function MAX(a_1, a_2, ..., a_n: integers)
    max ← a_1
    for i ← 2, n do
        if max < a_i then
            max ← a_i
    return max
```

Exercise 5

Which Big-O estimate applies to the worst-case running time of the following algorithm? (For the function g in your estimate $O(g(x))$, use a simple function g of smallest order.)

```
function SEARCH(key: integer, A: array of increasing integers)
    lo ← 1
    hi ← A.length
    while lo ≤ hi do
        mid = ⌊(lo + hi)/2⌋
        if key < A[mid] then
            hi = mid − 1
        else if key > A[mid] then
            lo = mid + 1
        else
            return mid
    return −1
```

Exercise 6

List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

$\sqrt{n}$	n	2^n
$n \log n$	$n - n^3 + 7n^5$	$n^2 + \log n$
n^2	n^3	$\log n$
$n^{\frac{1}{3}} + \log n$	$(\log n)^2$	$n!$
$\ln n$	$\frac{n}{\log n}$	$\log \log n$
$(1/3)^n$	$(3/2)^n$	6

Exercise 7

Consider the following code fragment.

```
procedure PRINT
    for i ← 1, n do
        for j ← i, 2i do
            output "simplify me"
```

Let $T(n)$ denote the number of times 'simplify me' is printed as a function of n.

1. Express $T(n)$ as a summation (actually two nested summations).
2. Simplify the summation.

Exercise 8

1. The conventional algorithm for evaluating a polynomial $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ can be expressed in pseudocode by

   ```
   function POLYNOMIAL(x, a_0, a_1, ..., a_n: real numbers)
       power ← 1
       y ← a_0
       for i ← 1, n do
           power ← power · x
           y ← y + a_i · power
       return y
   ```

   where the final value of y is the value of the polynomial. Exactly how many multiplications and additions are used to evaluate a polynomial of degree n? (Do not count additions used to increment the loop variable.)

2. There is a more efficient algorithm (in terms of the number of multiplications and additions used) for evaluating polynomials than the conventional algorithm described in the 1st question. It is called Horner's method. This pseudocode shows how to use this method to find the value of $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$.

   ```
   function HORNER(x, a_0, a_1, ..., a_n: real numbers)
   ```

$y \leftarrow a_n$
for $i \leftarrow 1, n$ **do**
    $y \leftarrow y \cdot x + a_{n-i}$
return y

Exactly how many multiplications and additions are used to evaluate a polynomial of degree n? (Do not count additions used to increment the loop variable.)

Exercise 9

Magic Square Puzzle: fill the 3x3 table with nine distinct integers from 1 to 9 so that the sum of the numbers in each row, column, and corner-to-corner diagonoal is the same.

?	?	?
?	?	?
?	?	?

Write the algorithm (using exhaustive search) in pseudocode that solves the Magic Square Puzzle. The algorithm has a start procedure named solveMagicSquarePuzzle and prints out each solution on a single line. Each line contains 9 values which represent the value chosen for each cell. The cells are numbered as following:

1	2	3
4	5	6
7	8	9

The values are written in increasing order of the cell index and are separated from each other by a whitespace. For example:

8 1 6 3 5 7 4 9 2

which represents the square:

8	1	6
3	5	7
4	9	2

3 Solutions

Answer of exercise 1

1. Finiteness. The algorithm does not terminate in a finite number of steps.

2. Definiteness. The step '$x \leftarrow$ either a or b' of the algorithm is not precisely defined.

Answer of exercise 2

1. $O(x^2)$
2. $O(x^2), \Omega(x^2), \Theta(x^2)$
3. $O(x^2)$
4. $\Omega(x^2)$
5. $\Omega(x^2)$
6. $O(x^2), \Omega(x^2), \Theta(x^2)$

Answer of exercise 3

$O(n^2)$

Answer of exercise 4

$O(n)$

Answer of exercise 5

$O(\log n)$

Answer of exercise 6

$(1/3)^n < 6 < \log\log n < \log n = \ln n < (\log n)^2 < n^{\frac{1}{3}} + \log n < \sqrt{n} < \frac{n}{\log n} < n < n\log n < n^2 = n^2 + \log n < n^3 < n - n^3 + 7n^5 < (3/2)^n < 2^n < n!$

Answer of exercise 7

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

Answer of exercise 8

1. 2n multiplications and n additions.
2. n multiplications and n additions.

Answer of exercise 9

Algorithm 1 Solving MagicSquare(3x3) using exhaustive search

```
1: procedure solveMagicSquare
2:     chosenNumbers ← empty list
3:     solveUsingExhaustiveSearch(chosenNumbers)

4: procedure solveUsingExhaustiveSearch(chosenNumbers: a list of integers)
5:     if chosenNumbers.length == 9 then
6:         checkForValidSolution(chosenNumbers)
7:     else
8:         numbers ← buildNumbers(chosenNumbers)    ▷ numbers is a list of integers

9:         for i ← 1, numbers.size() do
10:            chosenNumbers.add(numbers[i])
11:            solveUsingExhaustiveSearch(chosenNumbers)
12:            chosenNumbers.remove(chosenNumbers.size())

13: function buildNumbers(chosenNumbers: a list of integers)
14:    numbers ← empty list
15:    for i ← 1, 9 do
16:        numbers.add(i)
17:    numbers.removeAll(chosenNumbers)
18:    return numbers

19: procedure checkForValidSolution(candidateSolution: a list of integers)
20:    if isValid(candidateSolution) then
21:        println(candidateSolution)

22: function isValid(candidateSolution: a list of integers)
23:    sum ← 0
24:    scTL ← 0                          ▷ corner sum (top left to bottom right)
25:    scTR ← 0                          ▷ corner sum (top right to bottom left)
26:    for i ← 1, 3 do
27:        sR ← 0                                              ▷ the sum of row
28:        sC ← 0                                           ▷ the sum of column
29:        for y ← 1, 3 do
30:            listIndex ← calculateListIndex(i, y)
31:            sR ← sR + candidateSolution.get(listIndex)

32:            listIndex ← calculateListIndex(y, i)
33:            sC ← sC + candidateSolution.get(listIndex)
```

```
34:         if sum == 0 then
35:             sum ← sR

36:         if (sum != sR) OR (sum != sC) then
37:             return false

38:         listIndex ← calculateListIndex(i, i)
39:         cTL ← cTL + candidateSolution.get(listIndex)
40:         listIndex ← calculateListIndex(i, 4 − i)
41:         cTR ← cTR + candidateSolution.get(listIndex)

42:     return (sum == cTL) AND (sum == cTR)

43: function CALCULATELISTINDEX(rowIndex, columnIndex)
44:     return (rowIndex − 1) · 3 + (columnIndex − 1)
```