Exercise Sheet 2

Handout: September 16th- Deadline: September 23rd, 4pm

Question 2.1(0.3 marks)

Express the following running times in Θ -notation. Justify your answer by referring to the definition of Θ (i.e. work out suitable c_1, c_2, n_0).

a) $3n^2 + 5n - 2$

b) 42

c) $4n^2 \cdot (1 + \log n) - 2n^2$

Answer:

$\Theta(g(n)) = \{f(n) :$ there exist constants $0 < c_1 \leq c_2$ and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

a) $f(n) = \Theta(n^2)$
for all $n > 10$, $3n^2 < f(n) < 4n^2$
b) $f(n) = \Theta(1)$
for all $n > 1$, $1 * 1 < 42 < 100 * 1$
c) $f(n) = \Theta(n^2 logn)$
$f(n) = 2n^2 + 4n^2 logn$
for all $n > 4$, $4 < \frac{2}{logn} + 4 < 5, 4n^2 logn < 2n^2 + 4n^2 logn < 5n^2 logn$

Question 2.2(0.7 marks)

(a) Indicate for each pair of functions $f(n), g(n)$ in the following table whether $f(n)$ is O, o, Ω, ω ,or Θ of $g(n)$ by writing"yes" or"no" in each box.

Table 2

$f(n)$	$g(n)$	O	o	Ω	ω	Θ
$\log n$	$\sqrt{n}$	yes	yes	no	no	no
n	$\sqrt{n}$	no	no	yes	yes	no
n	$n \log n$	yes	yes	no	no	no
n^2	$n^2 + (\log n)^3$	yes	no	yes	no	yes
2^n	n^3	no	no	yes	yes	no
$2^{n/2}$	2^n	yes	yes	no	no	no
$\log_2 n$	$\log_{10} n$	yes	no	yes	no	yes

Hints: the book states that every polynomial of $\log n$ grows strictly slower than every poly-nomial n^ε , for constant $\varepsilon > 0$. For example, $(\log n)^{100} = o(n^{0.01})$. Likewise, every polynomial grows slower than every exponential function 2^{n^ε} , for example $n^{100} = o(2^{n^{0.01}})$.

To convert the base of a logarithm, use $\log_x(n) = \log_y(n)/\log_y(x)$.

Question 2.3(0.3 marks)

State the number of"foo" operations for each of the following algorithms in Θ -notation. Pay attention to indentation and how long loops are run for. Justify your answer by stating constants $c_1, c_2, n_0 > 0$ from the definition of $\Theta(g(n))$ in your answer.

Example: Line 1 is executed once and line 3 is executed n-4 times. So the number of foos is $1+n-4 = n-3 = 0(n)$ as $c_1 n \leq n - 3 \leq c_2 n$ for all $n \geq n_0$ when choosing,say, $n_0 = 6, c_1 = 1/2, c_2 = 1$.

Example Algorithm
1: foo
2: for i=1 to n-4 do
3: foo

ALGORITHM A

```
1: foo
2: for i = 1 to n do
3:    for j = 1 to n-2 do
4:       foo
5:       foo
6:       foo
```

Answer: Line 1 is executed once and line 4-6 are executed n(n-2) times each. So the number of foos is $1 + 3n(n-2) = 3n^2 - 6n + 1 = 0(n^2)$ as $c_1 n^2 \leq 3n^2 - 6n + 1 \leq c_2 n^2$ for all $n \geq n_0$ when choosing,say, $n_0 = 6, c_1 = 2, c_2 = 4$.

ALGORITHM B

```
1: foo
2: for i=1 to n do
3:    foo
4: for j=1 to n/2 do
5:    foo
6:    foo
```

Answer: Line 1 is executed once and line 3 is executed n times and line 5-6 are executed n/2 times each. So the number of foos is$1 + n + 2 * \frac{n}{2} = 1 + 2n = 0(n)$ as $c_1 n \leq 1 + 2n \leq c_2 n$ for all $n \geq n_0$ when choosing,say, $n_0 = 6, c_1 = 2, c_2 = 3$.

ALGORITHM C

```
1: foo
2: for i=1 to n do
3:    for j=1 to i do
4:       foo
5:    foo
6: foo
```

Answer: Line 1 is executed once and line 4 is executed $\frac{n(n+1)}{2}$ times,line5 is executed n times, line 6 is executed once. So the number of foos is$1 + \frac{n(n+1)}{2} + n + 1 = 0.5n^2 + 1.5n + 2 = 0(n^2)$ as $c_1 n^2 \leq 0.5n^2 + 1.5n + 2 \leq c_2 n^2$ for all $n \geq n_0$ when choosing,say, $n_0 = 10, c_1 = 1/2, c_2 = 2$.

Question 2.4(0.3 marks)

Recall from Lecture 2 that a statement like $2n^2 + \Theta(n) = \Theta(n^2)$ is true if no matter how the anonymous functions are chosen on the left of the equal sign, there is a way to choose the anonymous functions on the right of the equal sign to make the equation valid. You might want to think of the $\Theta(n)$ on the left-hand side being a placeholder for some (anonymous) function that grows as fast as n.

For each of the following statements, state whether it is true or false. Justify your answers.

$$1.\, O(\sqrt{n}) = O(n)$$

True. Any function growing not faster than $\sqrt{n}$ do not grow faster than n.

$$2.\, n + o(n^2) = \omega(n)$$

False. $1 = o(n^2)$, but $n + 1$ grow as fast as n, i.e.$n + 1 = \Theta(n)$ b

$$3.\, 3n \log n + O(n) = \Theta(n \log n)$$

True. Any function growing not faster than n must not grow faster than $3nlogn$, hence $3n \log n + O(n) = \Theta(n \log n)$ for the slower term can be ignored.

Also, explain why the statement"The running time of Algorithm A is at least $O(n^2)$ " is meaningless.

$f(n) = O(n^2)$ means $f(n)$ grow at most as fast as n^2, which make the statement "at least as fast as $f(n)$" meaningless.

Question 2.5(0.3 marks)

The following algorithm computes the product C of two $n \times n$ matrices A and B, where $A[i, j]$corresponds to the element in the i-th row and the j-th column.

```
    Matrix-Multiply(A,B)
1: for i=1 to n do
2:    for j=1 to n do
3:       C[i,j]:=0
4:       for k=1 to n do
5:          C[i, j] := C[i, j] + A[i, k] · B[k, j]
6: return C
```

Give the running time of the algorithm (number of operations in a RAM machine) in Θ-notation. Justify your answer. Feel free to use the rules on calculating with Θ-notation from the lecture.

Answer: Each of the for loop execute $\Theta(n)$ times,and each operation in the loop execute $\Theta(1)$ times for a single loop. $T(n) = \Theta(n)\Theta(n)(\Theta(1) + \Theta(n)\Theta(1)) = \Theta(n)\Theta(n)\Theta(n) = \Theta(n^3)$

Question 2.6(marks 0.75)

BubbleSort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order. The effect is that small elements"bubble" to the left-hand side of the array, accumulating to form a growing sorted subarray.(You might want to work out your own example to understand this better.)

```
Bubble-Sort(A)
1: for i=1 to A.length-1 do
2:    for j= A.length downto i+1 do
3:       if A[j] < A[j − 1]then
4:          exchange A[j] with A[j-1]
```

Prove the correctness of BubbleSort and analyse its running time as follows. Try to keep your answers brief.

1. The inner loop"bubbles" a small element to the left-hand side of the array. State a loop invariant for the inner loop that captures this effect and prove that this loop invariant holds, addressing the three properties initialisation, maintenance, and termination.

2. Using the termination condition of the loop invariant for the inner loop, state and prove a loop invariant for the outer loop in the same way as in part 1. that allows you to conclude that at the end of the algorithm the array is sorted.

3. State the runtime of BubbleSort in asymptotic notation. Justify your answer.

Answer:

1. Loop invariant: at the start of each iteration of line 2, A[j] is the smallest among the subarray A[j,n].
   Initialization: At the start of the first loop, A[j]=A[n] consists of only one element, so the loop invarient holds trivilly.
   Maintenance: At the start of iteration k, if A[k] is the smallest element among the subarray A[k,n],then the algorithm will move the smaller one between A[k] and A[k-1] to the k-1 position, which make A[k-1] the smallest among the subarray A[k-1,n].
   Termination: The loop ends at j=i.The loop invarient says that A[i] is the smallest among A[i,n].

2. Loop invariant: at the start of each iteration of line 1 , the subarray A[1...i-1] is sorted and all elements in A[1...i-1] is smaller than other elements in the remains part of the array.
   Initialization: At the start of the first loop, the subarray is empty, so the loop invarient trivilly holds.
   Maintenance: At the start of iteration k, if A[1...k-1] is sorted and all elements are smaller than those in A[k...n], then after the iteration, the inner loop would "buubles" the smallest element in A[k...n] to A[k] as the largest element in A[1...k], making the subarray A[1...k] sorted and all elements are smaller than those in A[k+1...n].
   Termination: The loop ends at i = n+1.The loop invarient says that A[1...n] is sorted.

3. Loop in line 1 is executed $\Theta(n)$ times, loop in line 2 is executed $\Theta(n)$ times, line 3&4 are executed $\Theta(1)$ times each iteration.So the runtime is $\Theta(n) * \Theta(n) * \Theta(1) = \Theta(n^2)$. To justify (assuming each line takes time 1 if executed), for the best case which the array is already sorted, line 3&4 are executed 0 times, the runing time is $n + (\frac{n(n+1)}{2}) = \Theta(n^2)$, while for the worst case which the array is sorted in inverse order, line 3&4 are executed every time, the runing time is $n + \frac{n(n+1)}{2} + \frac{n(n-1)}{2} * 2 = \Theta(n^2)$