CSE530 Algorithm and Complexity : Assignment I (due 3/19)

JongYun Kim (20185053)

PROBLEM I

The answers are given in the table I below.

TABLE I
THE ANSWER OF PROBLEM 1

Problem	$f(n) = o(g(n))$	$f(n) = \Omega(g(n))$
(a)	***correct***	-
(b)	-	***correct***
(c)	***correct***	-
(d)	-	***correct***
(e)	-	***correct***

PROBLEM II

II-(a)

Proof: $\forall n \;\; f(n), g(n) \geq 0, \; f(n) = \Theta(g(n)) \rightarrow f(n) + g(n) = \Theta(g(n))$

$f(n) = \Theta(g(n))$ is given so that there exist 3 constants $c_1 > 0$, $c_2 > 0$, $n_0 \in \mathbf{N}$ such that $n \geq n_0$ implies Inequality (1) below.

$$c_1|g(n)| \leq |f(n)| \leq c_2|g(n)| \tag{1}$$

Let $h(n) = f(n) + g(n)$. It is obvious that $h(n) \geq 0$. There exist 3 constants $c_1^* = c_1 + 1$, $c_2^* = c_2 + 1$, $n_0^* = n_0$ such that $n \geq n_0^*$ implies

$$c_1^*|g(n)| \leq |h(n)| \leq c_2^*|g(n)| \tag{2}$$

We can obtain Inequality (2) by adding $g(n)$ at every side of Inequality (1), considering the following equations below.

$$c_1|g(n)| + g(n) = (c_1 + 1)|g(n)|$$

$$|f(n)| + g(n) = |h(n)|$$

$$c_2|g(n)| + g(n) = (c_2 + 1)|g(n)|$$

Therefore $h(n) = f(n) + g(n) = \Theta(g(n))$.

∎

II-(b)

The example of function f and g is given as follows.

$$f(n) = -sin(n)$$

$$g(n) = sin(n)$$

Note that $h(n) = f(n) + g(n) = 0$. The functions still hold the condition $f(n) = \Theta(g(n))$. Because Equation (1) yields $\frac{1}{2}|sin(n)| \leq |-sin(n)| \leq 2|sin(n)|$ for $n \geq n_0$ if we choose $c_1 = \frac{1}{2}$, $c_2 = 2$, $n_0 = 1$.

However, the function f and g do NOT hold Inequality (2) anymore no matter what $c_1^* > 0$, $c_2^* > 0$, and $n_0^* \in \mathbf{N}$ are.

$$c_1^*|g(n)| \leq |h(n)| = 0 \leq c_2^*|g(n)| \tag{3}$$

There dose NOT exist constant c_1^* satisfying the left side Inequality (3) because c_1^* can not become zero.

PROBLEM III

III-(a)

Note that I followed similar steps and descriptions of the solution of exercise 1 of this course

Proof: Algorithm 1 is correct (loop invariant method)

We are going to use the following loop invariant: For every $i \geq 2$, at the start of the *i*th iteration of the loop, the subarray $A[1...i-1]$ contains x which is the smallest number of the subarray $A[1...i-1]$ or the second smallest number of the whole array $A[1...n]$.

Let me prove the three properties of this loop invariant. Let s_1, s_2 be the smallest and the second smallest number of $A[1...n]$ respectively. **Initialization.** $x = A[1]$ at the second iteration of the loop. It means that the subarray A[1] contains the smallest number of the subarray (*i.e.* loop invariant is true).

Maintenance. Suppose that the invariant is true before *i*th iteration of the loop for $i \geq 2$. So there are x in the subarray $A[1...i-1]$. And we head to the start of the $i+1$th iteration of the loop. Then the invariant is still true. For the reason we can think into two ways. First, if $x = \min(A[1...i-1]) \neq s_2$ at the beginning of ith iteration, then still $x = \min(A[1...i])$ $\big($because $A[i] > x_{i_1} \rightarrow x_i = x_{i-1} = \min(A[1...i-1]) = \min(A[1...i])\big)$. Second, if $x = s_2$ at the start of ith iteration, then x holds s_2 at the start of the next iteration. Because y should be s_1. In other words, $x + y$ will not become smaller (*i.e.* no update). Note that if s_1 is prior to s_2, x sticks to s_1 which is not s_2.

Termination. When the loop terminates, we have $i = n$, and thus the subarray $A[1...n-1]$ contains x which is s_2 or $\min(A[1...n-1])$. If $x = s_2$, then $y = s_1$. And if $x = \min(A[1...n-1]) \neq s_2$, it implies $x = s_1$ which obviously yields $y = s_2$ ∎

III-(b)

Note that $T_k(n)$ is how many times k line was implemented and e_ij is the binary number indicating whether

TABLE II
RUNNING TIME OF EACH LINE OF ALGORITHM 1

Line	Cost	Times
1	c_1	$T_1(n) = 1$
2	c_2	$T_2(n) = 1$
3	c_3	$T_3(n) = \sum_{i=1}^{n}(1) = n$
4	c_4	$T_4(n) = \sum_{i=1}^{n}\sum_{j=i+1}^{n+1}(1) = \frac{n^2}{2} + \frac{n}{2} - 1$
5	c_5	$T_5(n) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(1) = \frac{n^2}{2} - \frac{n}{2}$
6	c_6	$T_6(n) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(e_{ij}) \leq \frac{n^2}{2} - \frac{n}{2}$
7	c_7	$T_7(n) = \sum_{i=1}^{n}\sum_{j=i+1}^{n}(e_{ij}) \leq \frac{n^2}{2} - \frac{n}{2}$
8	c_8	$T_8(n) = 1$

conditional statement *if* on the *line 5* was implemented at *nested for loop* index i and j. It is obvious that $T_6(n) = T_7(n) \leq T_5(n)$ as stated in the table III. Therefore the running time $T(n)$ of the Algorithm 1 is given below.

$$T(n) = \sum_{k=1}^{n} c_k T_k \tag{4}$$

Note that I didn't write the whole term of the running time because a long equation won't be aligned in this environment but you can still get the whole term by referring Table III above.

III-(c)

$$T(n) = \Theta(n^2) \tag{5}$$

The asymptotic notation of the running time of Algorithm 1 is given in Equation (5).

Proof: $T(n) = \Theta(n^2)$

We can divide the running time into 2 cases.

Case I : Best case

The best case means that the conditional statement is not executed. It happens when $A[1]$ and $A[2]$ are the two smallest numbers. In other word, e_ij is always 0. Then the running time is given as follows.

$$T_b(n) = \left(\frac{c_4}{2}+\frac{c_5}{2}\right)n^2+\left(c_3+\frac{c_4}{2}-\frac{c_5}{2}\right)n+(c_1+c_2-c_4+c_8) \tag{6}$$

Case II : Worst case

The worst case means that the conditional statement is executed every time. It happens when A is sorted in decreasing order. In other word, e_ij is always 1. Then the running time is given as follows.

$$T_w(n) = \left(\frac{c_4+c_5+c_6+c_7}{2}\right)n^2+\left(c_3+\frac{c_4}{2}-\frac{c_5+c_6+c_7}{2}\right)n + (c_1+c_2-c_4+c_8) \tag{7}$$

So, the relation between the cases and running time is given as follows.

$$T_b(n) \leq T(n) \leq T_w(n) \tag{8}$$

A property of $\Theta(\cdot)$ is given in the slide of this course. You can see the property below as well.

`If` $f(n)$ `is a` d`-th polynomial, then` $f(n) = \Theta(n^d)$

The property and Equation (6) and (7) yield two equations below because the coefficients of n^2 in Equation (6) and (7) can not be zero.

$$T_b(n) = \Theta(n^2) \tag{9}$$

$$T_w(n) = \Theta(n^2) \tag{10}$$

The definition of $\Theta(\cdot)$ at the Equation (9) and (10) implies that there exist four constants $c_1 > 0$, $c_2 > 0$, n_b, and n_w such that

$$c_1 n^2 \leq T_b(n)\,, \quad n \geq n_b \tag{11}$$

$$T_2(n) \leq c_2 n^2\,, \quad n \geq n_w \tag{12}$$

Inequality (11) and (12) can be combined with Inequality (8).

$$c_1 n^2 \leq T_b(n) \leq T(n) \leq T_w(n) \leq c_2 n^2 \tag{13}$$

, where $n \geq n_0 = \max[n_b, n_w]$. The existence of $c_1 > 0$, $c_2 > 0$, and n_o at Inequality (13) directly means $T(n) = \Theta(n^2)$.

■

III-(d)

TABLE III
FASTER ALGORITHM THAN ALGORITHM 1

Algorithm 2
01: **procedure** FAST 2-MIN($A[1...n]$))
02: $key \leftarrow A[1]$, IDx $\leftarrow$ 1
03: **for** $i \leftarrow 2,n$ **do**
04: **if** $A[i] < key$ **then**
05: $key \leftarrow A[i]$, IDx $\leftarrow i$
06: x $\leftarrow key$
07: **if** IDx==1 **then**
08: $key \leftarrow A[2]$
09: **else**
10: $key \leftarrow A[1]$
11: **for** $j \leftarrow 2,n$ **do**
12: **if** j == IDx **then**
13: continue;
14: **if** $A[j] < key$ **then**
15: $key \leftarrow A[j]$
16: y $\leftarrow key$
17: **return** x,y

The numbers of implementation of each line are only one except for loops (*line 4∼5* and *line 11∼15*). The two for loop are parallel instead of being mutually nested. So the running time of this algorithm is $\Theta(n)$.

PROBLEM IV

IV-(a)

IV-(b)