CS-GY 6033: Homework #1

Due on September 27, 2023

Professor Yi-Jen Chiang

Runze Li
Nxxxxxxxx
rl50xx@nyu.edu

Tzu-Yi Chang
Nxxxxxxxx
tc39xx@nyu.edu

Jiayi Li
Nxxxxxxxx
jl156xx@nyu.edu

September 24, 2023

1.

(a) Yes, if $f(n)=O(g(n))$, it is possible that $f(n)=\Omega(g(n))$. When $f(n)=O(g(n))$, it implies that $f(n)$ has an upper bound on the growth rate of $g(n)$. In addition to this, if $f(n)=\Omega(g(n))$, $f(n)$ has a lower bound on the growth rate of $g(n)$. In this case, $f(n)=\Theta(g(n))$. However, it is not always true that $f(n)=\Omega(g(n))$ when $f(n)=O(g(n))$. For example, $f(n)$ could grow slower than $g(n)$ or even have no well-defined lower bound in terms of $g(n)$.

(b) Yes, if $f(n)=o(g(n))$, it is possible and always true that $f(n)=O(g(n))$. When $f(n)=o(g(n))$, it implies that $f(n)$ grows strictly slower than $g(n)$, which also satisfies the definition of $f(n)=O(g(n))$, where $f(n)$ grows no faster than $g(n)$.

(c) Yes, if $f(n)=\Theta(g(n))$, it is possible and always true that $f(n)=O(g(n))$. By definition, $f(n)=\Theta(g(n))$ implies that there exist constant c_1, c_2 and n_0 such that $0\leq c_1 g(n)\leq f(n)\leq c_2 g(n)\ \forall n\geq n_0$, which also satisfies $f(n)=O(g(n))$, where $0\leq f(n)\leq c_2 g(n)\ \forall n\geq n_0$.

(d) No, $f(n)=\omega(g(n))$ and $f(n)=O(g(n))$ cannot be true simultaneously. For $f(n)=\omega(g(n))$, $f(n)$ grows strictly faster than $g(n)$. On the other hand, $f(n)=O(g(n))$ implies that $f(n)$ is upper-bounded by a constant multiple of $g(n)$. These conditions are mutually exclusive.

(e) It is possible that $f(n)+g(n)=\Theta(\min(f(n),g(n)))$. Let's say $f(n)=2n$, $g(n)=n$, $f(n)+g(n)=2n+n=3n$, satisfies $f(n)+g(n)=\Theta(n)$. However, it is not always true. For example, when $f(n)=n^2$, $g(n)=n$, $f(n)+g(n)=n^2+n$. In this case, $f(n)+g(n)$ grows faster than $\min(f(n),g(n))$. Therefore, $f(n)+g(n)=\Theta(\min(f(n),g(n)))$ is not always true.

2. ① $n^{\frac{7}{2}} = f_1(n)$

$\forall C_1$ s.t. $n^{\frac{7}{2}} > C_1 \cdot n^3$

$C_1 < n^{\frac{1}{2}}$

we can find n_0:

$n_0 > C_1^2$

$\therefore f_1(n) = \omega(n^3)$

$n^{\frac{7}{2}} < C_2 n^4$

$C_2 > \frac{1}{\sqrt{n}}$

we can find n_0:

$n_0 > \frac{1}{C_2^2}$

$\therefore f_1(n) = o(n^4)$

② $n^3 \log n = f_2(n)$

$n^3 \log n > C_1 \cdot n^3$ $\quad n_0 > 2^{C_1}$ $\quad (\forall C_1)$

$\therefore f_2(n) = \omega(n^3)$

$n^3 \log n < C_2 \cdot n^4$

$$\lim_{n \to +\infty} \frac{n^3 \log n}{n^4} = \lim_{n \to +\infty} \frac{\log n}{n}$$

$$= \lim_{n \to +\infty} \frac{(\log n)'}{n'}$$

$= 0$

$\therefore f_2(n) = o(n^4)$

3.

	f(n) A	g(n) B	O	o	Ω	ω	Θ
c.	$\sqrt{n}$	$n^{\sin n}$	no	no	no	no	no
d.	2^n	$2^{\frac{n}{2}}$	no	no	yes	yes	no
e.	$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes
f.	$\lg(n!)$	$\lg(n^n)$	yes	no	yes	no	yes

c. $\because \sin n \in [-1, 1]$, it oscillates.

$\therefore n^{\sin n} \in (\frac{1}{n}, n)$

$\forall n>0, n>\sqrt{n}$ & $\frac{1}{n}<\sqrt{n}$ $(n>1)$

when we try to find $\sqrt{n} \underset{(<)}{\leq} c \cdot n^{\sin n}$, there is no such c.

we need to satisfy:
$\sqrt{n} \leq c \cdot \frac{1}{n}$ $(\forall n \geq n_0)$
$c \geq n \cdot \sqrt{n}$

also, when we try to find $\sqrt{n} \underset{(>)}{\geq} c \cdot n^{\sin n}$, there is no c.

$\therefore$ the result is all "no"

d. when $n>0$, $n>\frac{n}{2}$

when trying to find c s.t. $2^n > c \cdot 2^{\frac{n}{2}}$

$\forall c>0$, s.t. $2^n > c \cdot 2^{\frac{n}{2}}$, $\forall n \geq 2\lg c+1$ $\therefore 2^n = \omega(2^{\frac{n}{2}})$

this does meet "$\exists c>0$" in the definition of $\Omega(\cdot)$

obviously, $O(\cdot)$ & $o(\cdot)$ is not true here

$\therefore \Theta(\cdot)$ is also wrong.

e. $f(n) = n^{\lg c}$ $g(n) = c^{\lg n}$

try to find a constant m times $g(n)$: $mg(n)$.

$\lg(f(n)) = \lg c \cdot \lg n$ $\lg(mg(n)) = \lg m + \lg c^{\lg n}$

$= \lg m + \lg n \cdot \lg c$

when $m \geq 1$, $\lg m \geq 0$, $m g(n) \geq f(n)$, $\forall n \geq 1$

$f(n) = O(g(n))$

when $m \leq 1$, $\lg m \leq 0$, $m \cdot g(n) \leq f(n)$, $\forall n \geq 1$

$f(n) = \Omega(g(n))$

However, it doesn't meet the definition of $o(\cdot), \omega(\cdot)$ as there is no "$\forall$ constant m".

f. First: $n! = 1 \times 2 \times 3 \times \cdots \times n \leq n \times n \times n \times \cdots \times n = n^n$

$$n! \geq \underbrace{1 \times 1 \times 1 \times \cdots \times \frac{n}{2} \times \frac{n}{2} \times \cdots \times \frac{n}{2}}_{\frac{n}{2}} = \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$\therefore \lg(n!) \leq \lg n^n \Rightarrow \lg(n!) = O(n \lg n)$

and $\lg(n!) \geq \lg\left(\frac{n}{2}\right)^{\frac{n}{2}} \Rightarrow \lg(n!) = \Omega(n \lg n)$

$\therefore \lg(n!) = \Theta(n \lg n)$

Then: To prove $o(\cdot)$ and $\omega(\cdot)$ are both wrong here.

$\because \lg(n!) = O(n \lg n) = O(\lg n^n)$

$\therefore \exists c$ s.t. $\lg(n!) \leq c \cdot \lg(n^n)$

$\therefore$ It can not meet $\forall c'$ s.t. $\lg(n!) > c' \cdot \lg(n^n)$

$\therefore o(\cdot)$ is wrong. Same as $\omega(\cdot)$.

4. (a). $S_3(n) = 1^3+2^3+3^3+\cdots+n^3$

$S_3(n) \le n^3+n^3+n^3+\cdots+n^3 = n\cdot n^3 = n^4$

$S_3(n) = 1^3+2^3+3^3+\cdots+(\frac{n}{2}-1)^3+(\frac{n}{2})^3+(\frac{n}{2}+1)^3+\cdots+n^3$

$\ge 0 \qquad +\underbrace{(\frac{n}{2})^3+(\frac{n}{2})^3+\cdots+(\frac{n}{2})^3}_{\frac{n}{2}}$

$=(\frac{n}{2})^4$

$=\frac{n^4}{16}$

$\therefore \frac{n^4}{16} \le S_3(n) \le n^4$

$\therefore S_3(n) = \Theta(n^4)$

(b) $S_4(n+1) = \sum_{k=1}^{n+1} k^4 = \sum_{k=1}^{n} k^4 + (n+1)^4 = S_4(n) + (n+1)^4$ (a)

$S_4(n+1) = \sum_{k=0}^{n} (k+1)^4 = \sum_{k=0}^{n} (k^4+4k^3+6k^2+4k+1)$

$= 0 + S_4(n) + 0 + 4S_3(n) + 0 + 6S_2(n)$

$+ 0 + 4S_1(n) + n+1$ (b)

(a)=(b): $S_4(n)+(n+1)^4 = S_4(n)+4S_3(n)+6S_2(n)+4S_1(n)+(n+1)$

$\Rightarrow S_3(n) = \frac{(n+1)^4 - 6S_2(n) - 4S_1(n) - (n+1)}{4}$

$\because S_2(n) = \frac{n(n+1)(2n+1)}{6}$ and $S_1(n) = \frac{n(n+1)}{2}$

$\therefore S_3(n) = \frac{(n+1)^4 - n(n+1)(2n+1) - 2n(n+1) - (n+1)}{4}$

$= \frac{(n+1)[(n+1)^3 - n(2n+1) - 2n - 1]}{4}$

$= \frac{n^2(n+1)^2}{4}$

Problem 5

Given a sequence of n unordered real numbers, design and analyze an algorithm that find **both** the **largest** and the **second largest** numbers among them using at most $3\lfloor n/2 \rfloor$ comparisons. Your algorithm should describe what to do when n is even and when n is odd. You need to show that your algorithm gives the correct answers, and that it uses at most $3\lfloor n/2 \rfloor$ comparisons.
(**Note:** Here *comparisons* refer to comparisons between any two items in the input sequence of n real numbers.)

Solution

If we want to use at most $3\lfloor \frac{n}{2} \rfloor$ comparisons to find the largest and the second largest number, we can compare them every two numbers. **We can take out two adjacent numbers in the array every time to compare three times**.

Let the array be *arr* (from 0 to n-1), the largest number be *firstMax* and the second largest number be *secondMax*.

First of all, we compare *arr*[i] with *arr*[i+1], then let the larger one be *tempMax* and the smaller one be *tempMin*.

Second, we need to compare *tempMax* with *firstMax* to find the larger one.

At that time, if *firstMax* is larger than *tempMax*, we need to assign the larger one between *tempMax* and *secondMax* as *secondMax*.

Otherwise, we assign the larger one between *firstMax* and *secondMax* as *secondMax*, and assign *tempMax* as *firstMax*.

Because we compare two adjacent numbers every time, **we need to consider whether n is odd or even**.

If n is even, we don't need to compare anything else. We divide the sequence into $\frac{n}{2}$ subsequences, and the length of each is 2. We need $\frac{n}{2}$ comparisons in the first step, and $\frac{n-2}{2}$ comparisons in the second step and $\frac{n-2}{2}$ comparisons in the third step. So the total comparison is $\frac{n}{2} + \frac{n-2}{2} + \frac{n-2}{2} = 3\frac{n}{2} - 2 \leq 3\lfloor \frac{n}{2} \rfloor$.

If n is odd, we need to take out *arr*[n-1] and compare it with *firstMax* and *secondMax*. We divide the sequence into $\frac{n-1}{2}$ subsequences and we compare *arr*[n-1] individually. We need $\frac{n-1}{2}$ comparisons in the first step, $\frac{n-3}{2}$ comparisons in the second step, $\frac{n-3}{2}$ comparisons in the third step, and plus at most 2 comparisons when we compare *arr*[n-1] individually. So the total comparison is $\frac{n-1}{2} + \frac{n-3}{2} + \frac{n-3}{2} + 2 = 3\frac{n}{2} - \frac{3}{2} \leq 3\lfloor \frac{n}{2} \rfloor$.

The pseudo code **Find-First-and-Second-Largest** is as follows:

Problem 5 continued on next page...

Algorithm 1 Find-First-and-Second-Largest

Input: arr, n

Output: $firstMax, SecondMax$

```
1: if arr[0] > arr[1] then                          ▷ First comparison between arr[0] and arr[1]
2:     firstMax ← arr[0]
3:     secondMax ← arr[1]
4: else
5:     firstMax ← arr[1]
6:     secondMax ← arr[0]
7: end if
8:  i ← 2
9: while i < n − 1 do
10:    if arr[i] > arr[i + 1] then                                  ▷ First comparison
11:        tempMax ← arr[i], tempMin ← arr[i + 1]
12:    else
13:        tempMax ← arr[i + 1], tempMin ← arr[i]
14:    end if
15:    if tempMax > firstMax then                                   ▷ Second comparison
16:        if tempMin > firstMax then                               ▷ Third comparison(a)
17:            secondMax ← tempMin
18:        else
19:            secondMax ← firstMax
20:        end if
21:        firstMax ← tempMax
22:    else
23:        if tempMax > secondMax then                              ▷ Third comparison(b)
24:            secondMax ← tempMax
25:        end if
26:    end if
27:    i ← i + 2
28: end while
29: if i == n − 1 then                                  ▷ Consider whether n is odd or even
30:    if arr[i] > firstMax then
31:        secondMax ← firstMax
32:        firstMax ← arr[i]
33:    else if arr[i] > secondMax then
34:        secondMax ← arr[i]
35:    end if
36: end if
```