Problem Set #2 CSC236 Fall 2018

Si Tong Liu, shuo Yang, Jing Huang

Oct/12/2018

We declare that this assignment is solely our own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters.

This submission has been prepared using LaTeX.

Problem 1.

(Warmup - this problem will NOT be marked).

Show that $\log n! \in \mathcal{O}(n \log n)$.
(Here $m!$ stands for m factorial, the product of first m non-negative integers. By convention, $0! = 1$.)

Solution:

According to the question, n is a non-negative integers. So $n >= 0$. Then we can get that $n! <= n^n$ and $\log n! <= \log n^n$. By the property of log, we can write $\log n^n$ as $n \log n$. So we get $\log n! <= n \log n$.

This completes the proof, $\log n! \in \mathcal{O}(n \log n)$ for c = 1 and $n_0 = 0$.

Problem 2.

(4 Marks) Suppose you are coding an algorithm for finding the maximum sum of two elements in a list of positive integers. Suppose you have access to a helper function sort(L) that takes in a list of positive integers and returns a list of the same elements but sorted in non-decreasing order. Moreover, suppose `sort(L)` runs in time $\Theta(n \log n)$ (e.g., `mergesort`). Write a Python program `fastMaxSum` calling `sort(L)` as a helper function that runs in time $\Theta(n \log n)$. Justify why it has this running time.

`solution:`

```
def fastMaxSum(L:list):
    sortedList = sort(L)
    return sortedList[-1] + sortedList[-2]
```

Since the running time for L[-1] is O(1) and the running time for L[-2] is O(1). The total tuntime for the python function fastMaxSum is $2 + n \log n$

To prove this function runs in time $\Theta(n \log n)$, we divide into two parts.

Firstly, the Big O proof:

$2 + n \log n <= \log n + n \log n$ (if $\log n >= 2$, which implies to $n >= e^2$)

$<= n \log n + n \log n$ (by basic algorithm, $n \log n > \log n$, where $n >= e^2$)

$<= 2 \log n$

Therefore, we choose c = 2, $n_0 = e^2$ to complete the Big O proof

Secondly, the Big Omega proof:

$2 + n \log n >= n \log n$ (by basic algorithm) ($n >= 1$)

So we choose c = 1 and $n_0 = 1$ to complete the Big Omega proof.

Then, as the O and Ω proofs are now complete, the overall Θ proof is complete.

Problem 3.

(6 Marks) **Practice** Θ.

$$\forall k \in \mathbb{N}, 1^k + 2^k + \cdots + n^k \in \Theta(n^{k+1}).$$

solution: We divide the proof into two parts:

Firstly, the Big O proof:

since $n \in N and\ n >= 1$, each number from 1 to n-1 is less than n.

so we can write

$$1^k + 2^k + \cdots + n^k <= n^k + n^k + \cdots + n^k = n * n^k = n^{k+1}$$

$$<= n^{k+1}$$

So we choose c = 1 and $n_0 = 1$ to complete the Big O proof.

Secondly, the Big Omega proof:

to make

$$1^k + 2^k + \cdots + n^k >= c * (n^k + n^k + \cdots + n^k)$$

valid, we have to make c as small as possible.

So we choose $n_0 = 1$ and $c = 10^{-100000}$ to complete

Then, as the O and Ω proofs are now complete, the overall Θ proof is complete.

Problem 4.

(10 MARKS) **Recursive functions**.

Consider the following recursively defined function:

$$T(n) = \begin{cases} c_0 & n = 0 \\ c_1 & n = 1 \\ aT(n-1) + bT(n-2) & n \geq 2 \end{cases}$$

where a, b are real numbers.

Denote (*) the following relation:

$$T(n) = aT(n-1) + bT(n-2) \quad n \geq 2 \tag{$*$}$$

We say a function $f(n)$ satisfies (*) iff $f(n) = af(n-1) + bf(n-2)$ is a true statement for $n \geq 2$.

Prove the following:

(i) For all functions $f, g : \mathbb{N} \to \mathbb{R}$, for any two real numbers α, β, if $f(n)$ and $g(n)$ satisfy (*) for $n \geq 2$ then also $h(n) = \alpha f(n) + \beta g(n)$ satisfies it for $n \geq 2$.

(ii) Let $q \neq 0$ be a real number. Show that if $f(n) = q^n$ satisfies (*) for $n \geq 2$ then q is a root of quadratic equation $x^2 - ax - b = 0$.

(iii) State and prove the converse of (ii). Use this statement and part (i) to show that if q_1, q_2 are the roots of $x^2 - ax - b = 0$ then $h(n) = Aq_1^n + Bq_2^n$ satisfies (*) for any two numbers A, B.

(iv) Consider $h(n)$ from part (iii). What additional condition should we impose on the roots q_1, q_2 so $h(n)$ serves as a closed-form solution for $T(n)$ with A, B uniquely determined?

(v) Use the previous parts of this exercise to solve the following recurrence in closed form:

$$T(n) = \begin{cases} 5 & n = 0 \\ 17 & n = 1 \\ 5T(n-1) - 6T(n-2) & n \geq 2 \end{cases}$$

solution:

(i) if $f(n)$ and $g(n)$ satisfy (*) for $n \geq 2$, then we can write $f(n) = af(n-1) + bf(n-2)$ and $g(n) = ag(n-1) + bg(n-2)$.

So $h(n) = \alpha(af(n-1) + bf(n-2)) + \beta(ag(n-1) + bg(n-2))$

$= a(\alpha f(n-1) + \beta g(n-1)) + b(\alpha f(n-2) + \beta g(n-2))$

$= ah(n-1) + bh(n-2)$ (by given recursive rule) So we can verify that the statement is true.

(ii) if $f(n) = q^n$, we can plug it into T(n)

$q^n = aq^{n-1} + bq^{n-2}$

$q^n - aq^{n-1} - bq^{n-2} = 0$

$q^2 - aq - b = 0$ (by basic algorithm, since $q \neq 0, q^{n-2} \neq 0$, we can divide q^{n-2} both side)

Then if we assume q = x, we can get $x^2 - ax - b = 0$. So q is a root of the quadratic equation. This completes the proof.

(iii) Firstly, the converse of (ii) we need to prove: if q is a root of quadratic equation $x^2 - ax - b = 0$, then $f(n) = q^n$ satisfies (*) for $n \geq 2$

We plug q into the equation and we get $q^2 - aq - b = 0$

$$q^2 * q^{n-2} - aq * q^{n-2} - b * q^{n-2} = 0 (\text{since } q \neq 0, \text{ we can multiply } q^{n-2} \text{ both side})$$

$$q^n - aq^{n-1} - bq^{n-2} = 0$$

$$q^n = aq^{n-1} + bq^{n-2}$$

if we assume $f(n) = q^n$, we can get $f(n) = af(n-1) + bf(n-2)$, which satisfies (*)

This completes the proof. The converse of (ii) is true.

Secondly, by using the statement we have proven above, If we assume $f(n) = q_1^n$ and $g(n) = q_2^n$, both $f(n)$ and $g(n)$ should satisfy (*) for $n \geq 2$.

Then we can get $h(n) = Aq_1^n + Bq_2^n = Af(n) + Bg(n)$.

According to part(i), $h(n) = Aq_1^n + Bq_2^n$ satisfy (*) with $\alpha = A$ and $\delta = B$

This completes the proof.

(iv) To find the close form of T(n)

when k = 1 , $T(n) = aT(n-1) + bT(n-2)$

when k = 2 , $T(n) = a(aT(n-2)) + b(bT(n-4))$

$= a^2T(n-2) + b^2T(n-4)$

when k = 3 , $T(n) = a(a^2T(n-3)) + b(b^2T(n-6))$

$= a^3T(n-3) + b^3T(n-6)$

...

...

we can conclude that $T(n) = a^kT(n-k) + b^kT(n-2k)$

since $n-2k$ decrease faster than $n-k$. The base case is $T(n) = a^kc_1 + b^kc_0$

So the close form of $T(n)$ is $T(n) = a^nc_1 + b^nc_0$

If we assume $h(n)$ can serve as a closed-form solution for $T(n)$,

$h(n) = Aq_1^n + Bq_2^n = a^nc_1 + b^nc_0$

Then we have $q_1 = a$ and $q_2 = b$, $A = c_1$ and $B = c_0$

In conclusion, if we add condition: $q_1 = a$ and $q_2 = b$, $h(n)$ can be the close form for $T(n)$

(v) According to previous parts , $T(n) = c_1a^n + c_0b^n = 17(5^n) + 5(-6)^n$