Analysis of Algorithms - Home Work 1

CSE 548 Fall '19

Prof. Jie Gao

Narayan Acharya

nacharya@cs.stonybrook.edu

1 Question 1

Big-O notation (10pts) Prove or disprove (i.e., give counter examples) for the following claims. $f(n), g(n)$ are non-negative functions.

1. $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.
   Solution:

2. $o(f(n)) \cap \omega(f(n)) = \emptyset$.
   Solution: $o(f(n))$, by definition, is a function that *grows strictly slower* than $(f(n))$ while $\omega(f(n))$, by definition, is a function that *grows strictly faster* than $f(n)$.

$$o(f(n)) < f(n) : \forall n \tag{1}$$

$$\omega(f(n)) > f(n) : \forall n \tag{2}$$

   Equations 1 and 2 imply that these functions are on the opposite sides of the tightest possible growth rate function and can never be same or overlap. Therefore, the intersection of these functions is rightly an empty set, $\emptyset$ and the statement $o(f(n)) \cap \omega(f(n)) = \emptyset$ **is true**.

3. $(n + a)^b = \Theta(n^b)$, a, b are positive integers.
   Solution: $(n + a)^b$ is a polynomial of order b. The expression expanded with all its terms will look like this:

$$(n+a)^b = \binom{b}{0}.n^b.a^0 + \binom{b}{1}.n^{(b-1)}.a^1 + \ldots + \binom{b}{b}.n^0.a^b \tag{3}$$

   But we know that for terms where order of n is $\leq b$,

$$\binom{b}{1}.n^{(b-1)}.a^1 \leq n \times \binom{b}{1}.n^{(b-1)}.a^1 : \forall n \geq 1 \tag{4}$$

$$\binom{b}{1}.n^{(b-2)}.a^1 \leq n^2 \times \binom{b}{1}.n^{(b-2)}.a^2 : \forall n \geq 1 \ldots \tag{5}$$

   Therefore from equations 3, 4 and 5we can deduce that, $\forall n \geq 1$,

$$\binom{b}{0}.n^b.a^0+\binom{b}{1}.n^{(b-1)}.a^1+\ldots+\binom{b}{b}.n^0.a^b \leq \binom{b}{0}.n^b.a^0+n\times\binom{b}{1}.n^{(b-1)}.a^1+n^2\times\binom{b}{1}.n^{(b-2)}.a^2 \ldots \tag{6}$$

$$\therefore (n+a)^b \leq n^b \times \binom{b}{1}.a^1 + \binom{b}{1}.a^2 \ldots \binom{b}{b}.a^b \tag{7}$$

$$\therefore (n+a)^b \leq c \times n^b, c = \binom{b}{1}.a^1 + \binom{b}{1}.a^2 \ldots \binom{b}{b}.a^b \tag{8}$$

$$\boxed{\therefore (n+a)^b = O(n^b)} \tag{9}$$

   On similar lines, we can also prove $(n + a)^b = \Omega(n^b)$ using the fact that all terms other than of order b contribute positively in equation 3,

$$(n+a)^b \geq \binom{b}{0}.a^0.n^b \tag{10}$$

$$\boxed{\therefore (n+a)^b = \Omega(n^b)} \tag{11}$$

From equations 9 and 11,

$$\boxed{(n+a)^b = \Theta(n^b)} \tag{12}$$

4. $f(n) = O(f(n)^2)$.
Solution: Given that $f(n)$ is a non-negative function,

$$f(n) \leq c \times f(n) : \forall n \geq n_0, c > 0 \tag{13}$$

$$\therefore f(n)^2 \leq c^2 \times f(n)^2 \tag{14}$$

$$\boxed{\therefore f(n)^2 = O(f(n)^2)} \tag{15}$$

$f(n)^2$ may or may not be the tightest bound for $f(n)$ but it is asymptotically upper bound for a non-negative function $f(n)$.

5. $f(n) = O(g(n))$ implies that $2^{f(n)} = O(2^{g(n)})$.
Solution: Say $f(n) = k \times n$, given we have $f(n) = O(g(n))$, $g(n) = n$.

$$\therefore 2^{f(n)} = 2^{(k \times n)}, 2^{g(n)} = 2^n \tag{16}$$

$$\therefore O(2^{f(n)}) = O(2^{(k \times n)} \tag{17}$$

But, $O(2^{g(n)}) = O(2^n) \neq O(2^{(k \times n))}$

$$\boxed{\therefore 2^{f(n)} \neq O(2^{g(n)})} \tag{18}$$

2 Question 2

Sort the following functions from asymptotically smallest to asymptotically largest. That is, the function $f(n)$ and the next function $g(n)$ must always follow that $f(n) \in O(g(n))$. If the two functions have asymptotic the same order, i.e., $f(n) = \Theta(g(n))$, then also indicate that. No need to write down proofs. Remember $\lg n = \log_2 n$.

n, $\lg n$, $\sqrt{n}$, $\sqrt{\lg n}$, $\lg\sqrt{n}$, 2^n, $2^{\sqrt{n}}$, $\sqrt{2^n}$, $2^{\lg n}$, $\lg(2^n)$, $2^{\lg\sqrt{n}}$, $2^{\sqrt{\lg n}}$, $\sqrt{2^{\lg n}}$, $\lg(\sqrt{2^n})$, $\sqrt{\lg(2^n)}$, 3^n, $3^{\sqrt{n}}$, $\sqrt{3^n}$, $3^{\lg n}$, $\lg(3^n)$, $3^{\lg\sqrt{n}}$, $3^{\sqrt{\lg n}}$, $\sqrt{3^{\lg n}}$, $\lg(\sqrt{3^n})$, $\sqrt{\lg(3^n)}$.

Solution: Below is table of the above functions sorted from asymptotically smallest to largest. *Note:* Even though Row 14 onwards all functions are exponential in nature they are asymptotically different and therefore have different values of $g(n) = O(f(n))$.

Row	Function	Simplified Representation	$g(n) = O(f(n))$
1	$\sqrt{\lg n}$	$\sqrt{\lg n}$	$\sqrt{\lg n}$
2	$\lg\sqrt{n}$	$\lg(n)/2$	$\lg n, \Theta(\lg n)$
3	$\lg n$	$\lg n$	
4	$2^{\sqrt{\lg n}}$	$2^{\sqrt{\lg n}}$	$2^{\sqrt{\lg n}}$
5	$3^{\sqrt{\lg n}}$	$3^{\sqrt{\lg n}}$	$3^{\sqrt{\lg n}}$
6	$\sqrt{n}$, $\sqrt{2^{\lg n}}$, $2^{\lg\sqrt{n}}$	$\sqrt{n}$	$\sqrt{n}, \Theta(\sqrt{n})$
7	$\sqrt{\lg(2^n)}$	$\sqrt{\lg(2)} \times \sqrt{n}$	
8	$\sqrt{\lg(3^n)}$	$\sqrt{\lg(3)} \times \sqrt{n}$	
9	$3^{\lg\sqrt{n}}$, $\sqrt{3^{\lg n}}$	$n^{\lg\sqrt{3}}$	$n^{\lg\sqrt{3}}$
10	$\lg(\sqrt{2^n})$	$n \times \lg(2)/2$	$n, \Theta(n)$
11	$\lg(\sqrt{3^n})$	$n \times \lg(3)/2$	
12	n , $\lg(2^n)$, $2^{\lg n}$	n	
13	$\lg(3^n)$	$n \times lg(3)$	
14	$3^{\lg n}$	$n^{\lg 3}$	$n^{\lg 3}$
15	$2^{\sqrt{n}}$	$2^{\sqrt{n}}$	$2^{\sqrt{n}}$
16	$3^{\sqrt{n}}$	$3^{\sqrt{n}}$	$3^{\sqrt{n}}$
17	$\sqrt{2^n}$	$\sqrt{2^n}$	$\sqrt{2^n}$
18	$\sqrt{3^n}$	$\sqrt{2^n}$	$\sqrt{2^n}$
19	2^n	2^n	2^n
20	3^n	3^n	3^n

Narayan Acharya
112734365

3 Question 3

Textbook [Kleinberg & Tardos] Chapter 2, page 67, problem #6.

Solution:

(a) The algorithm given in the question has an upper bound of the order n^3. This can be proved as follows:

The algorithm loops over i n-times. During each iteration over i it loops over j (n-i)-times. For adding all the elements from A[i] through A[j], there needs to be an additional loop, say with index k, (j-i+1)-times.

The above can be represented as a function of n as:

$$f(n) = \sum_{i=1}^{i=n} \sum_{j=i+1}^{j=n} \sum_{k=i}^{k=j} Add! \tag{19}$$

Expanding and simplifying the series:
Note: The 'Add!' operation is considered constant time and hence omitted from the series below as it contributes only a factor of 1 to each term.

$$f(n) = (n-1).(1) + (n-2).(2) + (n-3).(3) + \ldots + (n-n).(n)$$
$$= [(n + 2n + 3n + \ldots + n.n) - (1 + 4 + 9 - \tag{20}$$

Simplifying this equation we get (Using formulas for adding a series n natural numbers and their squares):

$$f(n) = \frac{(n^3 - n)}{6} \tag{21}$$

The function is a polynomial of the order 3.

$$\therefore f(n) = O(n^3) \tag{22}$$

(b) From previous questions, we know that for a polynomial function, we have $O(f(n)) = \Omega(f(n))$

(c) The given algorithm is inefficient as the innermost loop is not needed. Instead of iterating over all elements A[i] through A[j] while adding them, we can keep track of the sum after each iteration over i and use that as our seed for adding as we loop over j. This can be shown in the below algorithm.

```
for i = 1, 2, 3 ... n do
    sumSoFar = A[i];
    for j = i+1, i+2, i+3 ... n do
        sumSoFar += A[j];
        B[i][j] = sumSoFar;
    end
end
```

The running time of this algorithm can be represented as a function of n as:

$$g(n) = \sum_{i=1}^{i=n} \sum_{j=i+1}^{j=n} Add! \tag{23}$$

Expanding the series:

$$\begin{aligned} g(n) &= (n-1) + (n-2) + (n-3) + \ldots + (n-n) \\ &= (n+n+n+\ldots+n) - (1+2+3+\ldots+n) \end{aligned} \tag{24}$$

Simplifying this equation we get:

$$g(n) = \frac{(n^2-n)}{2} \tag{25}$$

To prove that this algorithm is asymptotically better, we need to prove $lim_{n\to\infty}\frac{g(n)}{f(n)} = 0$.

$$\begin{aligned} lim_{n\to\infty}\frac{g(n)}{f(n)} &= lim_{n\to\infty}\frac{\frac{(n^2-n)}{2}}{\frac{(n^3-n)}{6}} \\ &= lim_{n\to\infty}\frac{(n-1)\times 3}{n^2-1} \\ &= lim_{n\to\infty}\frac{(n-1)\times 3}{(n-1)\times(n+1)} \\ &= lim_{n\to\infty}\frac{3}{n+1} \\ &= 0 \end{aligned} \tag{26}$$

$$\therefore lim_{n\to\infty}\frac{g(n)}{f(n)} = 0 \tag{27}$$

Hence, proposed algorithm's running time is asymptotically better.

4 Question 4

Recall that in a heap we keep the elements such that the parent is smaller than children. Thus the root (stored as the first element in the array) is the smallest item in the array. Insertion and of a new element can be done in $O(\lg n)$ time. Deletion of an element can be done in $O(\lg n)$ time. Thus one start from an empty heap and insert elements one by one to build a heap of n elements. Further, we can use it to sort n elements. Once the heap is built, we remove the root (the smallest element), which is the smallest of the n elements. Iterate and we will get all n elements output in the increasing sorted order. This algorithm is called HeapSort.

1. What is the running time for HeapSort? Assume elements come in an arbitrary order and represent the running time in big-O notation. (2pts)
2. What is the running time of HeapSort on n elements in increasing order? Represent the running time in big-Θ notation. (4pts)
3. What is the running time of HeapSort on n elements in decreasing order? Represent the running time in big-Θ notation. (4pts)

5 Question 4

Young Tableaus An $m \times n$ Young tableau is an $m \times n$ matrix such that the entries of each row are in increasing order from left to right and the entries of each column are in increasing order from top to bottom. Some of the entries of each column may be ∞, which are treated as nonexistent elements. A Young tableau with some elements as ∞ is not full.

Give an algorithm that extracts MIN (i.e., delete the minimum element and restore the matrix to be a Young tableau) on a nonempty $m \times n$ Young tableau that runs in $O(m + n)$ time.