CSCI-GA.1170-001, 003 Fundamental Algorithms

Name : Advait Pravin Savant
Net ID : as14634

Homework 1

Problem 1:

For each of the following pairs of functions $f(n)$ and $g(n)$, state whether f is $O(g)$; whether f is $o(g)$; whether f is $\Theta(g)$; whether f is $\omega(g)$.

(a) $f(n) = (n^n)^3$; $g(n) = n^{(n^3)}$

Solution:

- Consider the definition of the Big-Oh notation

We say $f(n)$ belongs to the set of functions $O(g(n))$ if there exists a constant c such that for all $n \geq n_0$,

$$f(n) \leq c\, g(n)$$

Here, $c > 0$ and n_0 is an arbitrary positive value of n which works with our inequality.

From our question,

$f(n) = (n^n)^3 = n^{3n}$

$g(n) = n^{n^3}$

We have,

(1) $3n \leq n^3 \quad \forall\, n \geq \sqrt{3}$ # Since we are dealing with bounds for computational procedures relative to the input size, we have $n > 0$

∴ ∀ for sufficiently large n,

(2) $n^{3n} \leq n^{n^3}$

If $a > 1$, and $x > y$, $a^x > a^y$, we follow equation (2) from exponential identities

Taking $c = 1$, $n_0 = 10$ (since n_0 can be anything greater than the lower bound for the inequality to hold) in equation 1,

We get,

$$(n^n)^3 \equiv O(n^{n^3})$$
$$f(n) \equiv O(g(n))$$ - Point 1

Θ, Ω notations are not applicable for the f, g relationship as defined in the question - Point 2.

Consider the definition of the little-oh notation
little-oh refers to bounds which are not asymptotically tight (as we take upper bounds)

$f(n) \equiv o(g(n))$ if,
$\forall$ constants $c > 0$,
$\exists$ n_0 such that, $f(n) \leq c\, g(n)$ $\forall$ $n \geq n_0$

Let c be an arbitrary positive constant,

Consider the inequality,

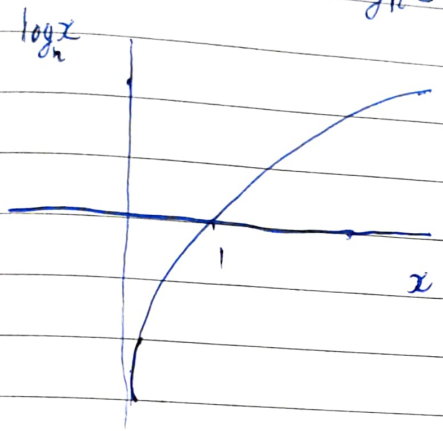
(3) $n^{3n} \leq c \cdot n^{n^3}$, $n > 0$

$$n^{3n} \leq n^{\log_n c} \cdot n^{n^3}$$

$$n^{3n} \leq n^{\log_n c + n^3}$$

From inequalities for exponentials, this extends to,

$$3n \leq \log_n c + n^3 \quad (4)$$



- Diagram 1 ($\log_n x$ vs x)

For any positive constant c, we show that the inequality (4) shall hold for some n.

- Consider the function $h(x) = (x^3 + \log_x c) - 3x$, $x > 0$
  As x tends to infinity, the cubic term will dominate and $h(x)$ will be positive, this can be shown formally in terms of limits. $h(x)$ will be positive for all c.
- As x tends to zero, if for example, $0 < c < 1$, the logarithm will be negative and $h(x)$ will be negative in certain cases of c.
- For such a c, by the intermediate value theorem, there will be some n_0^* such that $h(x) = 0$, and for $n > n_0^*$, the from the calculus of the growth of polynomials, we find a value of n such that the inequality holds for all values greater.
- If $h(x)$ is always positive, the inequality holds for such a c as easily follows from the equation.
- Thus we see that $\forall c$, we can find an n_0 to fit the

definition for the little-oh notation

$$(n^k)^3 \equiv o(n^{n^3})$$

$$f(n) \equiv o(g(n)) \quad - \text{ Point 3.}$$

little-omega definition not applicable - Point 4.

(b) $f(n) = n!$

$g(n) = (n+1)!$

$$g(n) = (n+1)\,n!$$
$$= (n+1)\,f(n)$$

$$n! = O((n+1)!)$$

$$f(n) = O(g(n)) \quad - \text{ Point 1}$$

$$n! = o((n+1)!)$$

$$f(n) = o(g(n)) \quad - \text{ Point 2}$$

$$\# \lim_{n\to\infty} \frac{f(n)}{g(n)} \equiv \frac{n!}{(n+1)!} \equiv \frac{1}{n+1} = 0$$

(c) $f(n) = n^{0.99} + 15(\log_2 n)^{100}$

$g(n) = 4^{\log_{16} n^2}$

$g(n) = (n^2)^{\log_{16} 4} = n$

$\therefore f(n) = O(g(n))$ - Point 1

$f(n) = \sigma(g(n))$ - Point 2

(d) $f(n) = n^{0.00001}$

$g(n) = (\log n)^{100001}$

$f(n) = \Omega(g(n))$ - Point 1

$f(n) = \omega(g(n))$ - Point 2

(e) $f(n) = n^{5/\log_2 n}$ $g(n) = 10000$

$f(n) = \theta(g(n))$ - Point 1

$n^{5 \cdot (\log_2 n)^{-1}} = n^{5 \log_n 2} = n^{\log_n 2^5} = (2^5)^{\log_n n}$

$= 2^5 = 32$

Both are constant time complexity functions

We know, $\frac{1}{\log_a n} = \log_n a$, This comes from the following identity:

$$\log_a b = \frac{\log_c b}{\log_c a}$$

Problem 2 :

Write the following functions in the θ - notation in the following simple form $c^k \cdot n^d (\log n)^r$ where c, d, r are constants

(a)

1. $f(n) = \dfrac{n^2 + 2}{1 + n^3 \, 2^{-n}} \sim \theta(n^2)$ //

We know,

$$\lim_{n \to \infty} \frac{n^3}{2^n} = 0$$

$\therefore f(n) \to n^2$ as $n \to \infty$, we can express asymptotically tight bounds in terms of the theta notation.

2. $f(n) = \log(n^{1/3}) \cdot \log(37n^3 + 45)$

$$= \frac{1}{3} \log n \cdot \log(37n^3 + 45)$$

Consider this, $c_1 \geq 0$, $c_2 > 0$

$$c_2 \log n \leq \frac{1}{3} \log(37n^3 + 45) \leq c_1 \log n \quad \forall \; n \geq n_0 \qquad -(1)$$

To show this, we see the limit,

$$L = \lim_{n \to \infty} \frac{k \cdot \log(an^3 + b)}{\log n}$$

where k, a, b are positive constants

Using L'Hospital's rule,

$$L = \frac{\frac{k}{an^3+b} \cdot 3n^2 \cdot a}{\frac{1}{n}} = \frac{k(3a)n^2}{an^2+b}$$

$$= \frac{3ak}{a + \frac{b}{n^2}}$$

We see that as n tends to infinity, the limit approaches a constant.
We can therefore express $k \log(an^3+b)$ as bounded by $\log n$, as per inequality (1).

Now, $f(n) = \log n \cdot k \log(an^3+b)$

$$f(n) \equiv \Theta((\log n)^2)$$

(c) $f(n) = \log(n!) + 10^{205} n$

$\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \ldots 1)$

$= \log n + \log(n-1) + \log(n-2) \ldots \log(1)$

$\log(n!) \equiv \Theta(n \log n)$

We can see that, $\exists\, c_1$ and $\exists\, c_2$, such that

$c_1 n \log n \leq \log(n!) \leq c_2 n \log n$

$\forall\, n > n_0$

for positive constants c_1, c_2 and some large positive number n_0.

$f(n) = \Theta(n \log n)$ //

$\log 2 < \log 3 < \log 4 \ldots < \log(n-1) < \log n$

$\exists\, c_1,\ c_1 n < \log 2$, for some n, we can find c_1 such that $c_1 n < \log 2$

$\therefore c_1 n$ will be less than all the terms except $\log 1$, the addition will give us the desired inequality c_1 can be calibrated to ascertain the bounds. Hence the notation holds.

(d) $$f(n) = \frac{6^n - 1000}{2^n + 1}$$

$$\lim_{n \to \infty} f(n) = \frac{6^n \cdot \log 6}{2^n \cdot \log 2} = k\, 3^n$$ where k is a constant

$$f(n) = \Theta(3^n)$$

(e) $f(n) = n^{2/\log n} + 1000$

$$f(n) = n^{2 \log_n e} + 1000$$ assuming natural log, similarly for log to the base 2

$$= n^{\log_n e^2} + 1000$$

$$f(n) = \text{Constant}$$

$$f(n) = \Theta(1)$$ //

(f) $f(n) = 2^n + \log n$

$= 2^n \; 2^{\log_2 n}$

$= 2^n \, n$

$f(n) = \theta(2^n n)$ //

(g) $2^n + 10(\log n)^{(\log n)}$

$f(n) = \theta(2^n)$ //

Substitute $\log n = t$, for limits, derivative of x^x used

The exponential term will dominate

Limits using L'Hospital's rule for first term as numerator & 2^{nd} term as denominator

(h) $f(n) = 2^{3n} + 4^n$

$= 2^{3n} + (2^2)^n$

$= 2^{3n} + 2^{2n}$

$= 2^{2n}(1 + 2^n) \sim \theta(2^{3n})$

$f(n) = \theta(8^n)$ //

Problem 2:

(b) List the simple functions derived in part (a) in the order of asymptotic growth, from smallest to largest.

$$\theta(1) \leq \theta((\log n)^2) \leq \theta(n \log n) \leq \theta(n^2) \leq \theta(2^n)$$
$$\leq \theta(2^n \cdot n) \leq \theta(3^n) \leq \theta(8^n)$$

Problem 3

(a) Consider sorting n numbers by iterating through the array and exchanging $A[i]$ and $A[i+1]$ if they are out of order. Write pseudocode for this algorithm which is known as Bubble - sort.

```
Bubble_Sort (A)

1. for i = 1 to (A.length - 1)
2.     for j = 1 to (A.length - i)
3.         if (A[j] > A[j+1])
4.             exchange (A[j], A[j+1])
```

What Bubble sort does is that for each iteration over the array, it sends the corresponding i^{th} largest element to the correct position in the array. We assume our array is to be sorted in ascending order. For example, in the first pass, the largest element is percolated to the end to the array. We assume array indices from one to n for an array of size n.

In the second pass, the second largest element will enter the second last element of our array. After i passes, the last i elements of the array will be sorted. So in the $(i+1)^{th}$ pass, we only need to do comparisions until element $n-(i+1)$. -eg-

$n = 9$, Array elements $a_1, \ldots, a_9$

after 3 passes,

a_7, a_8, a_9 are in sorted order ($a_7 \leq a_8 \leq a_9$)

In the 4^{th} pass, after we compare a_5 with a_6 and swap accordingly we need not make further comparisions. This justifies our loop indices.

Now, we also need to check for the completeness of our sorting procedure so that we avoid extra comparisons over iterations.

Bubble_Sort (A)

```
1.  flag = 1
2.  for i = 1 to A.length - 1
3.      if (flag == 0)
4.          break
5.      flag = 0
6.      for j = 1 to (A.length - i)
7.          if (A[j] > A[j+1])
8.              exchange (A[j], A[j+1])
9.              flag = 1
```

(b) In the worst case, the outer loop of Bubble-Sort only needs to run $(n-1)$ times. Prove this by giving a loop invariant that the algorithm maintains. Give the best case and worst case running times of Bubble-Sort in Θ-notation.

⇒ Loop invariant for bubble sort -

- At the end of iteration i of the outer loop, the elements a_n, a_{n-1} upto a_{n-i+1}, esentially, the last i elements, are in sorted order with a_n being the largest element in array A.

- Iteration i puts the i^{th} largest element in the correct position in the array.

⇒ Initialization

- Prior to the first iteration of our loop, for $i=0$, our statement boils down to saying that before the start of the first iteration, after the zeroth iteration, the last zero elements are in sorted order, such a statement is trivially true.
- For elaboration, consider the case $i=1$. We will prove by contradiction that our invariant must hold.
- Assume that the largest element L is at position $k < n$ at the end of the first iteration.
- During the first iteration, as inner loop variable $j=k$, since L is greater than all other elements, L must be exchanged with element $A[k+1]$. We then increment k, L must now be exchanged with $A[k+2]$ and so on until L is at position n of array A.
- Therefore our assumption cannot be true. L is at position n after iteration $i=1$.

⇒ Maintainance

We assume that after iteration i, we have elements $a_n, a_{n-1} \text{ —— } a_{n-i+1}$ in sorted order for array A such that,

$$a_{n-i+1} < a_{n-i+2} \text{ —— } < a_{n-1} < a_n$$

eg- $n = 9$, $i = 3$,
a_7, a_8, a_9 in sorted order at the corresponding positions.

During iteration $(i+1)$, the inner loop variable j will go from 1 to $n-(i+1)$,
For elements $a_1, a_2, \text{ —— }, a_{n-i}$, assume that the largest element is at position k, $k \in [1, n-i]$
largest among the subset of elements in consideration.
For the boundary case, if the largest element is at position $(n-i)$, our condition is already satisfied and our maintainence proof is complete.

For $k \in [1, n-(i+1)]$, as j reaches the value of k, the largest element will be exchanged with subsequent elements ahead in that iteration until j reaches $n-(i+1)$, at this point the largest element is exchanged with the element at position $(n-i)$.

Hence, elements $a_{n-i}, a_{n-i+1} \text{ —— } a_n$ are in sorted ascending order.
This follows from our i^{th} iterations assumptions for our maintainence proof.
We see that after iteration $(i+1)$, given our assumptions on iteration i, elements $a_n, a_{n-1}, \text{ —— } a_{n-(i+1)+1}$

are in sorted order such that,

$a_{n-i} \leq a_{n-i+1} < \cdots < a_{n-1} < a_n$

i.e $a_{n-(i+1)+1} < a_{n-i+1}$

This completes our maintainance argument. Such a formulation is analogous to a proof by induction.

⇒ Termination:

At the end of iteration $(n-1)$, $(n-1)$ elements of array A are in sorted order from position 2 to position n.

$a_2 < a_3 \cdots < a_{n-1} < a_n$

The element a_1, which is the last remaining element, is therefore the smallest element in our array since our loop invariant entails that elements $a_2, \ldots, a_n$ are the $(n-1)$ largest elements in our array.

Hence, in the worst case, the outer loop runs only $(n-1)$ times.

Worst case running time of Bubble sort - $\underline{\theta(n^2)}$

For an array sorted in reverse order, $(n-1)$ comparisions, $(n-1)$ exchanges in the first iteration.

$(n-2)$ comparisions, $(n-2)$ exchanges in the second iteration and so on and so forth. We add up the atomic operations for every iteration.

The arithmetic progression,

$1 + 2 + 3 \ldots n = \frac{n(n+1)}{2}$ when used for our case, gives us, the desired result.

Here $n = n-1$

In the best case, the running time of Bubble sort is $\Theta(n)$. This is when the array is already sorted. The verification takes place as we discussed in the first pass over the array.

(c) Let $A[1, \_\_, n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A.
What is the relationship between the number of exchanges in Bubble sort and the number of inversions I in the input array.

→ The number of exchanges that our bubble sort algorithm does during the procedure is equal to the number of inversions in the array.

→ The number of inversions in an array serves as a lower bound for the running time of our bubble sort.

→ We use the following arguments,

1. If there exists an inversion $(i, i+k)$, there must be a corresponding exchange between $A[i]$ the element and $A[i+k]$ the element during our bubble sort. Albeit, these elements may be at different positions during the time of the exchange in our procedure.

2. If there is an exchange between elements $A[j], A[j+1]$ during our procedure, there must be a corresponding inversion which existed in our array originally.

1. For array A, suppose $A[i] > A[i+k]$ in the initial order. We have $A[i], A[i+1], \_\_, A[i+k]$
   - Consider any iteration r where these positions hold for the corresponding elements. If $A[i]$ is greater than all other elements, the swap with $A[i+k]$ the element will happen in iteration r itself as the inner loop variable j proceeds.
   - If their is an element greater than $A[i]$, let $A[L]$ be the greatest element, position of $A[L]$ is between $A[i], A[i+k]$

- Element A[i+k], will be subsequently swapped with element A[i] and it will come one position behind.
- A[i] will go to positions ahead as it is swapped with elements less than it based on the input context.
- What we arrive at is that after a finite amount of iterations A[i+k] the element will be exchanged with A[i] (the
- Bubble sort exchanges 2 elements at a time which are adjecent ones. Since A[i] is larger than A[i+k], it must be put ahead of it in the final array. For that to happen, there must be a corresponding exchange at some point during the procedure.

2. If there is an exchange between elements A[j], A[j+1] during our procedure,

$A[j] > A[j+1]$

let x, y be the corresponding elements at $A[j]$ and $A[j+1]$

$x > y$

Now, before it arrives at position j, x can be either at position j or behind any position that y was holding. If x was ahead of y, there would have been no exchange between x and y, since x is greater, and y would not be ahead of x (at j+1) in the first place. Therefore, in the original array, the position of x must be behind the position of y, an inversion must exist.

Based on these arguments, our premise is valid.

$(p \rightarrow q) \wedge (q \rightarrow p) \equiv p \leftrightarrow q$

(d) Give an example of an array where the running time of bubble sort is $\theta(n^2)$ but the number of inversions is $\theta(n)$

$$A = [8, 15, 17, 20, 25, 1]$$

$n = 6$

$N(\text{inversions}) = 6 = \theta(n)$

$\{(8,1), (15,1), (17,1), (20,1), (25,1)\}$

The $\theta(n^2)$ running time follows from our loop invariant and the nature of bubble sort as discussed previously.

In iteration 1, 25 will occupy the last position after $(n-1)$ comparisions. Element 1 will be in the second last position.
In iteration 2, 20 will occupy the second last position. Element 1 will be in the third last position.
Summing over the atomic operations for all iterations, we get $\theta(n^2)$ time complexity.

(e) Give an example of an array for which insertion sort runs in time $\Theta(n)$ but bubble sort runs in time $\Theta(n^2)$.

$A = [8, 15, 19, 20, 25, 1]$

(f) Can you give an example of an array for which Bubble Sort runs in time $\Theta(n)$ but insertion sort runs in time $\Theta(n^2)$. Justify your answer.

For an array A, the number of inversions range from 0 to $\frac{n(n-1)}{2}$, n is the size of the array.

Let I be the total number of inversions
For element A[d], the number of inversions we associate with A[d] can be considered,

$N(x)$ such that $A[x] > A[d]$ and $x \in (1, d-1)$
We denote $N(x)$ as $i(d)$. $i(d)$.
We are essentially counting inversions relative to the elements before A[d].
Note that our nomenclature is exhaustive, if A[d] is involved in inversions with elements ahead of it, these inversions will be counted relative to that succeeding element.
What we want to get to is,

$$\sum_{d=1}^{n} i(d) = I$$

- For any iteration j in insertion sort, elements 1 to (j-1) are sorted. The j^{th} element is put in the sorted correct position in this array.
- The number of comparisions we make in this loop is equal to $i(j)$ as we defined it. After $i(j)$ comparisions we will insert the element at position j into the correct position for that iteration and move on to the next iteration.
- As we sum up the operations across all iterations, we see that insertion sort runs in $\Theta(n+I)$.
- We have also seen that the number of inversions I in an Array serves as a lower bound for the running time of bubble sort.
- Since, that many exchanges must take place in the bubble sort proceduce.
- To claim that bubble sort runs in $\Theta(n)$ would be to claim that the number of inversions is linear proportional to n.
- Therefore, insertion sort would also run in linear time proportional to n.
- Hence, it is not possible to find a permutation of input such that bubble sort is $\Theta(n)$ and insertion sort is $\Theta(n^2)$

(g) Let $A[1, \_\_, n]$ be a random permutation of $\{1, 2, \_\_, n\}$. What is the expected number of inversions of A. What can you conclude about the average case running time of Bubble-Sort (where the average is taken over the choice of permutation A of size n)?

We continue with our previous nomenclature as we introduce random variables.

X - random variable denoting total number of inversions

X_i - random variable denoting number of inversions relative to element i.
$N(x)$ such that $A[x] > A[i]$ and $x \in (1, i-1)$

$$X = X_1 + X_2 + X_3 \_\_ X_N$$

For elements $A[i]$ and $A[j]$, the Probabability $p(i,j)$ of one inversion is $\frac{1}{2}$.

$$X_i = \text{X}$$

$$X_i = I_{i1} + I_{i2} \_\_ I_{i(i-1)}$$

I is an indicator variable for inversions.

$$E[X_i] = E[I_{i1}] + E[I_{i2}] \_\_ E[I_{i(i-1)}]$$

I_{xy} denotes the existence of an inversion for positions x, y.

Note that,

$$E[X] = \sum_{x} p(X = x) \; x$$

For the indicator R.V,

$$p(I = 0) = 1/2$$

$$p(I = 1) = 1/2$$

$$E[I] = \frac{1}{2}$$

$$E[X_i] = \frac{i-1}{2}$$

$$E[X] = \sum_{i=1}^{n} \frac{i-1}{2}$$

$$= \frac{1}{2} \sum_{i=1}^{n} i - \sum_{i=1}^{n} \frac{1}{2}$$

$$= \frac{1}{2} \cdot \frac{n(n+1)}{2} - \frac{n}{2}$$

$$= \frac{n^2 + n - 2n}{4}$$

$$E[X] = \frac{n(n-1)}{4}$$

The expected number of inversions is $\frac{n(n-1)}{4}$