Problem Set 1

→ Problem 1-1

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

$f = o(g)$
$f = O(g)$

(b) $f(n) = n!$; $g(n) = (n+1)!$

$f = o(g)$
$f = O(g)$

(c) $f(n) = n^{0.99} + 15(\log_2 n)^{100}$; $g(n) = 4^{\log_{16} n^2}$

$f = o(g)$
$f = O(g)$

(d) $f(n) = n^{0.000001}$; $g(n) = (\log n)^{1000001}$

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

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

$f = \Theta(g)$

$f = O(g)$

$f = \Omega(g)$

Problem 1-2

(a)

(1) $$\frac{n^2+2}{1+n^3 \cdot 2^{-n}} = \frac{n^2+2}{1+\frac{n^3}{2^n}} \sim n^2$$ //

Reason: $\frac{n^3}{2^n}$ term tends to zero as $n \to \infty$.

Applying limit test numerator becomes zero.

Hence, for our function, denominator becomes 1, and we get n^2 in Θ notation.

(2) $\log(\sqrt[3]{n}) \cdot \log(37n^3 + 45) \sim (\log n)^2$.

Reason : $\log n^3 \left(37 + \frac{45}{n^3}\right)$

$= \log n^3 + \log\left(37 + \frac{45}{n^3}\right)$

$\simeq 3 \log n$

and 1st term is $\frac{1}{3} \log n$.

(3) $\log(n!) + 10^{205} n \sim n \log(n)$

Reason: Citing equation 3.19 from CLRS book where they prove that $\log(n!) = \Theta(n \log n)$ using Stirling's approximation.

Also, $10^{205} n = o(n \log n)$
& if $f(n) = g(n) + o(g(n))$
then $f(n) = \Theta(g(n))$
from Recitation slides.

(4) $\frac{6^n - 1000}{2^n + 1} \sim 3^n$ //

Reason: Applying limits as $n \to \infty$, we can ignore the constants in numerator & denominator.

(5) $n^{2/\log n} + 1000 \sim 1004$ //

Reason: $n^{2/\log n} = n^{2\log_n 2}$

$= n^{\log_n 4}$

$= 4$ //

(6) $2^{n+\log n} \sim n2^n$ //

Reason: $2^{\log_2 n} = n$

Hence our term becomes $n2^n$.

(7) $2^n + 10(\log n)^{\log n} \sim 2^n$ //

Reason: $(\log n)^{\log n} = 2^{\log(\log n)^{\log n}}$

$= 2^{\log n \cdot \log(\log n)}$

&

$n > \log n > \log(\log n)$

↳ from recitation slides.

(8) $2^{3n} + 4^n \sim 8^n$ //

Reason: $2^3 = 8$

& $8 > 4$

(b) In order of asymptotic growth:

$1004 < (\log n)^2 < n \log n < n^2 < 2^n < 3^n < 8^n < n2^n$ //

Problem 1-3

(a) Non-Recursive Algorithm:

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

↳Reference CLRS book 2.2.

(b) To Prove: In worst case, the outer loop of BUBBLE SORT only needs to run n-1 times.
using Loop invariant.

Proof -

Loop invariant - After every iteration of outer loop, the sub-array of values [A.length+1-i, A.length] is sorted.

Base case → i=1. Subarray of values is [A.length, A.length] which is only a single value and it is naturally

sorted. Hence loop invariant holds for base case.

Maintenance - Assuming inner loop is correct in shifting larger element towards right, After every iteration, the larger elements are placed at the rightmost side.

So for $i=2$, [A.length - 1, A.length] will be sorted.

for $i=3$, [A.length - 2, A.length] will be sorted.

and so on.

Hence, the loop invariant is true and preserved.

Termination - The iteration ends with $i =$ A.length - 1.

For $i =$ A.length - 1, sub array [2, 3, ..., A.Length] will be sorted.

and the 1st term is the smallest term left. Hence, there is no need to have one more iteration, as the array is actually sorted.

Hence, we have proved that the outer loop only needs to run $n-1$ times, using a loop invariant.

Best case running time for BUBBLE SORT
$= \Theta(n)$ since it scans the array only once.

Worst case run time for BUBBLE SORT
$= \Theta(n^2)$

since it needs to travel the array $n-1$ times & for n elements.

(c) For an array $A[1, \ldots, n]$
pair (i, j) is an inversion if
$i < j$ and $A[i] > A[j]$.

For number of exchanges in BUBBLE SORT,
Let total exchanges required be E.
Let exchanges required in outer loop iteration i, be e_i.

$\therefore$ We can say that

$$E = \sum_{i=1}^{n-1} e_i \quad \text{--(1)}.$$

For number of inversions in an array,
Lets consider an element $A[j]$ with j inversions where it is the larger element.
If we sort this array using Bubble sort, it will perform exactly j exchanges ~~inversions~~ to reach its place in the sorted array.

Note: These exchanges shift the smaller element towards left.

⇒ The smaller elements' inversions remain the same since they are calculated towards right.

⇒ If we perform some exchanges, the total number of inversions don't change.

⇒ Since, there is only one sorted permutation of an array, a fixed number of exchanges are required to sort it.

⇒ Hence, we can say that in i^{th} loop, number of inversions of that element
= number of exchanges it performs
= e_i.

⇒ Total number of inversions = Sum of all exchanges taking place at each iteration.

Therefore, we can conclude that

$$I = \sum_{i=1}^{n-1} e_i = E$$

where, I = Total number of inversions.

E = Total number of exchanges.

(d) Consider the situation, when in an array only one element is not sorted. Rest of the array is sorted.

i.e. let array $A = [a_1, a_2, \cdots a_n]$ be sorted.

Consider this permutation of A :

$$A = [a_n, \underbrace{a_1, a_2 \cdots, a_{n-1}}_{\text{sorted.}}]$$

In this situation, number of inversions would actually be $n-1$. or $\Theta(n)$.

But,

Bubble sort's outer loop will run for $n-1$ times and inner loop will also run its length fully.

Though swapping will only occur in one iteration, it will loop through all.

Hence its run time will be $\Theta(n^2)$.

This situation will occur when any one element of an array is out of place while rest is sorted.

(e) Consider a situation, where in an array only one element is not sorted and rest of the array is sorted.

Let $A[a_1, a_2 \dots a_n]$ be the array. (sorted).

We consider this permutation of A:

$A[a_n, \underbrace{a_1, a_2 \dots a_{n-1}}_{\text{sorted}}]$

For insertion sort,

the term a_n will move towards right one element at a time per iteration.

But the effective time it takes will be one for loop or we can say its run time is $\Theta(n)$. //

For Bubble sort,

The term a_n will move towards right in one single outer loop iteration.

But the inner loop will run for all outer loop iterations as well.

So effectively run time is $\Theta(n^2)$ //

(f) NO, ideally there can be no such case where runtime of BUBBLE sort is $\Theta(n)$ while of INSERTION SORT is $\Theta(n^2)$.

d).

This is due to the difference in their algorithms.

The bubble sort compares an element with its neighbors and swaps. This happens in every inner loop iteration. Hence there are a high number of swaps/comparisons. for one outer loop iteration.

On the otherhand, in insertion sort, an element is inserted at its correct place by comparing with the sorted sub array. Once, the element finds its place, it stops & doesn't keep going further.

Thus, by design, insertion sort has less number of exchanges as compared to Bubble sort.

This difference, increases further when the length of array increases.

Furthermore, Bubble sort has a run time of $O(n)$ only when the array is sorted. In that case, insertion sort also has a similar run time.

∴ There is no possible case, where insertion sort maybe $\Theta(n^2)$ while Bubble sort is $\Theta(n)$.

(g) $A[1, \ldots, n]$ is a random permutation of $\{1, 2, \ldots n\}$.

An inversion is a pair (i, j) where $i < j$ & $A[i] > A[j]$.

As the definition is exclusive, it can mean that a pair is either an inversion or its not.

All possible distinct pairs in $A[1, \ldots n]$ would be: $(n-1) + (n-2) + \ldots 1$

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

Using linearity of expectation,

E(inversions + not-inversions)

= E(inversions) + E(not inversions)

Since, there is an equal probability of a pair being an inversion or not, their expectations should be the same.

$\therefore$ E(inversions + not. inversions)
= E(inversions) + E(inversions)

$\therefore \frac{(n-1)n}{2} = 2\,E(\text{inversions})$

$\Rightarrow E(\text{inversions}) = \frac{(n-1)n}{4}$

Expected number of inversions of A is $\frac{(n-1)n}{4}$ //

{Ref. CLRS}

As we noticed in part (c) of some problem,
Number of inversions
= Number of Exchanges in Bubble Sort.

$\therefore$ Expected number of inversions over a permutation of A can give us the average case run time of Bubble sort.

i.e. $\frac{n(n-1)}{4} \sim \Theta(n^2)$ //