Workshop 2 – Week 3 – Worksheet 3

Question 2.1 Given the following functions f(n) and g(n), is f in O(g(n)) or is f in Θ(g(n)), or both?

	f(n)	**g(n)**	**f(n)∈O(g(n))**	**f(n) ∈Θ(g(n))**
a)	n + 100	n + 200	Yes	Yes
b)	$\log_2(n)$	$\log_{10}(n)$	Yes	Yes
c)	2^n	2^{n+1}	Yes	Yes
d)	2^n	3^n	Yes	No

Question 2.2 […] Given the descriptions of the run time in the two left hand columns, what can you say, in each instance, about the relative performance of the two algorithms when:

1. big-O is used in the usual Computer Science sense, as the least upper bound;

2. big-O is used in its strictest sense to mean any upper bound.

Algorithm 1	**Algorithm 2**	**Relative performance (CS-sense)**	**Relative performance (strict sense of big-O)**
O(n log n)	$O(n^3)$	Algorithm 2 grows faster than Algorithm 1	Nothing except that Algorithm 1 ***may*** run faster than Algorithm 2
Θ(n log n)	$O(n^3)$	Algorithm 2 grows faster than Algorithm 1	Again, nothing except that Algorithm 1 ***may*** run faster than Algorithm 2
O(n log n)	$\Theta(n^3)$	Algorithm 2 grows faster than Algorithm 1	Algorithm 2 grows faster than Algorithm 1
Θ(n log n)	$\Theta(n^3)$	Algorithm 2 grows faster than Algorithm 1	Algorithm 2 grows faster than Algorithm 1

NOTE: It is key that we only say one algorithm grows faster than the other, we cannot say one is faster than the other.

Question 2.3 What is the difference between the two following declarations?

int a[10][20];

*int *b[10];*

a is an array of 10 arrays of 20 elements (this is implemented as one array of 200 elements, however the conceptual answer is what we were expecting).

1. How could you use them both as 2-dimensional arrays?

   ```
   int i;

   for(i = 0; i < 10; i++){

           b[i] = (int *) malloc(sizeof(int)*20);

   }
   ```

2. What advantages might there be to declaring an array like *b[] above, instead of like a[][] above?

   The length of each list can be fined tune to match what is needed.
   Lists need not be the same length (consider the case of a set of strings).
   Lists can be swapped using simple assignment.
   …

3. How could you make a variable declared as int **c into a 2-dimensional array? (write the code)

   ```
   int i;

   c = (int **) malloc(sizeof(int *)*10);

   for(i = 0; i < 10; i++){

           c[i] = (int *) malloc(sizeof(int)*20);

   }
   ```

Question 2.4 Give a characterization, in terms of big-O, big-Ω and big-Θ, of the following loops:

```
int p = i;

for (int i = 0 ; i < 2*n ; i++){

        p = p * i;

}
```

O(n), Ω(n), Θ(n) are three options, big-Ω could be Ω(1) or anything of smaller than or equal to the order of n, and big-O could be $O(n^2)$ or anything of larger than or equal to the order of n. Θ(2n) is another likely option.

```
int s = 0 ;
for (int i = 0 ; i < 2*n ; i++){
        for (int j = 0 ; j < i ; j++){
                s = s + i;
        }
}
```

$O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$, with the same kind of generalisation possible as in the previous loop.