CS2383 Assignment 1 (46 marks)

1. **(5 marks)** Given an algorithm whose running time function is $O(n^2)$. Which of the following statements are true? Briefly explain why.

   (a) The running time function is exactly an quadratic function.
   No, because a linear function $3n + 2$ is also $O(n^2)$.

   (b) The running time function could be $2n^3$.
   No, because $2n^3$ is not $O(n^2)$.

   (c) The running time function could be $3n \log n + \log \log n$.
   Yes, because $3n \log n + \log \log n$ is $O(n^2)$.

   (d) The running time function could be $1000n + 100000$.
   Yes, because $1000n + 100000$ is $O(n^2)$.

   (e) The running time function could be $\Omega(n)$.
   Yes. For example, $1000n + 100000$ is $O(n^2)$ and $\Omega(n)$.

2. **(5 marks)** The following functions are comparable by asymptotic growth. Can you put them in order?
   n; 3^n; $nlogn$; n^n; $n - n^3 + 7n^5$; $n^2 + logn$; n^2; $logn$; $n!$

   If you put function $f(n)$ to the left of $g(n)$, then it must be the case that $f(n)$ is $O(g(n))$.

   Solution: $logn$; n; $nlogn$; n^2; $n^2 + logn$; $n - n^3 + 7n^5$; 3^n; $n!$; n^n.

3. (15 marks) Prove or disprove each of the following statements:

   (a) $10n^3 + 8n^2 + 6n + 2$ is $O(n^3)$.
   Proof: (3 marks)

$$\begin{aligned} 10n^3 + 8n^2 + 6n + 2 &\leq 10n^3 + 8n^3 + 6n^3 + 2n^3 \\ &= 26n^3 \end{aligned} \tag{1}$$

   Let $C = 26$ and $n_0 = 1$. We have $10n^3 + 8n^2 + 6n + 2 \leq Cn^3$ for all $n \geq n_0$.

(b) $3(3n+2)^7 + 4(2n+3)^5 + n\log n$ is $O(n^7)$.

Proof: (3 marks)

$$\begin{aligned}
3(3n+2)^7 + 4(2n+3)^5 + n\log n &\leq 3(3n+2n)^7 + 4(2n+3n)^5 + n\log n \\
&= (3\times 5^7)n^7 + (4\times 5^5)n^5 + n\log n \\
&\leq (3\times 5^7)n^7 + (4\times 5^5)n^7 + n^7 \\
&= (3\times 5^7 + 4\times 5^5 + 1)n^7
\end{aligned} \tag{2}$$

Let $C = 3\times 5^7 + 4\times 5^5 + 1$ and $n_0 = 1$. We have $3(3n+2)^7 + 4(2n+3)^5 + n\log n \leq Cn^7$ for all $n \geq n_0$.

(c) $3n^5 - 9n^4\log_2 n + 7n^3 - 15n^2$ is $O(n^5)$

Proof: (3 marks)

$$\begin{aligned}
3n^5 - 9n^4\log_2 n + 7n^3 - 15n^2 &\leq 3n^5 + 7n^3 \\
&\leq 3n^5 + 7n^5 \\
&= 10n^5
\end{aligned} \tag{3}$$

Let $C = 10$ and $n_0 = 1$. We have $3n^5 - 9n^4\log_2 n + 7n^3 - 15n^2 \leq Cn^5$ for all $n \geq n_0$.

(d) n^4 is $O(10^6 n^3 \log_2 n)$

Disproof: (3 marks) To make this true, we should have constants C and n_0, such that for all $n \geq n_0$,

$$\begin{aligned}
n^4 &\leq C10^6 n^3 \log_2 n \\
n &\leq C10^6 \log_2 n \\
\frac{n}{\log_2 n} &\leq C10^6
\end{aligned} \tag{4}$$

Since the growth rate of n is greater than $\log_2 n$ and C is a constant, $\frac{n}{\log_2 n} \geq C10^6$ when n is large. Therefore, it is not possible to find an n_0 to make $\frac{n}{\log_2 n} \leq C10^6$ for all $n \geq n_0$ given any constant C.

(e) $2^{\log_{10} n}$ is $O(n^{\frac{1}{3}})$

Proof: (3 marks)

$$\begin{aligned} 2^{\log_{10} n} &= 2^{\frac{\log_2 n}{\log_2 10}} \\ &= (2^{\log_2 n})^{\frac{1}{\log_2 10}} \\ &= n^{\frac{1}{\log_2 10}} \end{aligned} \tag{5}$$

Since $\log_2 10 > \log_2 8 = 3$, we have $2^{\log_{10} n} < n^{\frac{1}{3}}$. Let $C = 1$ and $n_0 = 1$. We have $2^{\log_{10} n} \leq C n^{\frac{1}{3}}$ for all $n \geq n_0$.

4. (5 marks) What does the following algorithm do? Analyze its worst-case running time, figure out its running time function, and express it using "Big-Oh" notation.

Algorithm Foo (a, n):
Input: two integers, a and n
Output: ?

$k \leftarrow 0$
$b \leftarrow 1$
while $k < n$ **do**
  $k \leftarrow k + 1$
  $b \leftarrow b * a$
return b

Solution: (5 marks) This algorithm computes a^n. The running time of this algorithm is $O(n)$ because

- the initial assignments take constant time
- each iteration of the **while** loop takes constant time
- there are exactly n iterations

5. (5 marks) What does the following algorithm do? Analyze its worst-case running time, figure out its running time function, and express it using "Big-Oh" notation.

Algorithm Bar (a, n):
Input: two integers, a and n
Output: ?

$k \leftarrow n$
$b \leftarrow 1$
$c \leftarrow a$
while $k > 0$ **do**
  if $k \bmod 2 = 0$ **then**
    $k \leftarrow k/2$
    $c \leftarrow c * c$
  else
    $k \leftarrow k - 1$
    $b \leftarrow b * c$
return b

Solution: (5 marks) This algorithm also computes a^n. Its running time is $O(\log n)$ for the following reasons:

The initialization and the **if** statement and its contents take constant time, so we need to figure out how many times the **while** loop gets called. Since k goes down (either gets halved or decremented by one) at each step, and it is equal to n initially, at worst the loop gets executed n times. But we can (and should) do better in our analysis.

Note that if k is even, it gets halved, and if it is odd, it gets decremented, and halved in the next iteration. So at least every second iteration of the **while** loop halves k. One can halve a number n at most $\lceil \log n \rceil$ times before it becomes ≤ 1 (each time we halve a number we shift it right by one bit, and a number has $\lceil \log n \rceil$ bits). If we decrement the number in between halving it, we still get to halve no more then $\lceil \log n \rceil$ times. Since we can only decrement k in between two halving iterations (unless n is odd or it is the last iteration), we get to do a decrementing iteration at most $\lceil \log n \rceil + 2$ times. So we can have at most $2\lceil \log n \rceil + 2$ iterations. This is obviously $O(\log n)$.

6. **(5 marks)** What does the following algorithm do? Figure out its best-case running time function and worst-case running time function, and express them using "Big-Oh" notation.

Algorithm Unknown(A, n):
Input: an array A of n integers
Output: ?
$f \leftarrow 1$
$j \leftarrow 1$
while $f = 1$ and $j \leq (n - 1)$**do**
  $f \leftarrow 0$

```
        for i ← 0 to n − (j + 1) do
            if A[i] > A[i + 1]
                swap A[i] and A[i + 1]
                f ← 1
        j ← (j + 1)
    return A
```

Solution: It sorts A in an ascending order. Best case running time is $\Theta(n)$. Worst case running time is $\Theta(n^2)$.

7. (6 marks) Suppose $f(n)$ is $O(h(n))$ and $g(n)$ is $O(h(n))$.

   (a) Is $f(n) + g(n)$ is $O(h(n))$?

   Proof: (3 marks) Since $f(n)$ is $O(h(n))$, there exist positive constant C_1 and n_{01}, such that $f(n) \leq C_1 h(n)$, for all $n \geq n_{01}$. Similarly, there exist positive constant C_2 and n_{02}, such that $g(n) \leq C_2 h(n)$, for all $n \geq n_{02}$. Let $C = max\{C_1, C_2\}$ and $n_0 = max\{n_{01}, n_{02}\}$. We have $f(n) + g(n) \leq Ch(n)$, for all $n \geq n_0$. Thus, $f(n) + g(n)$ is $O(h(n))$.

   (b) Is $f(n) \times g(n)$ is $O(h(n) \times h(n))$?

   Proof: (3 marks) Since $f(n)$ is $O(h(n))$, there exist positive constant C_1 and n_{01}, such that $f(n) \leq C_1 h(n)$, for all $n \geq n_{01}$. Similarly, there exist positive constant C_2 and n_{02}, such that $g(n) \leq C_2 h(n)$, for all $n \geq n_{02}$. Let $C = C_1 \times C_2$ and $n_0 = max\{n_{01}, n_{02}\}$. We have $f(n) \times g(n) \leq Ch(n) \times h(n)$, for all $n \geq n_0$. Thus, $f(n) \times g(n)$ is $O(h(n) \times h(n))$.

Justify your answers.