CMPS 6610 Problem Set:

Name: Pavan Kumar Sanjay
Student-ID: 287009125.

1. Asymptotic notation

1a Yes $2^{n+1} \in O(2^n)$ as 2^n asymptotically dominates 2^{n+1}. To prove this the following statement can be written as:

$2^{n+1} \leq c \cdot 2^n$ (Defination of asymptotic dominance where c is a constant).

$2^n \cdot 2 \leq c \cdot 2^n$

Here $c = 2$
∴ For any value of c above ~~2~~ or equal to 2
$2^{n+1} \in O(2^n)$.

1b. No the statement $2^{2^n} \in O(2^n)$ is not true.

Proof:

To prove that 2^n does not asymptotically dominate 2^{2^n} we must disprove for ~~two~~ any values of c and n.

$2^{2^n} \leq c \cdot 2^n$

where n is the input size and c is a constant.

To disprove this statement lets take two cases:

(1) $c = 5$ and $n = 3$
LHS (Left Hand side): $2^{2^n} = 2^{2^3} = 2^8$
Here the LHS $= 256$.
RHS (Right hand side): $c \cdot 2^n = 5 \times 2^3 = 40$.

$\therefore$ When the value of $c = 5$ and if $n \geq 3$ the statement $2^{2^n} \leq c \cdot 2^n$ is false.

(2) $c = 6$ and $n = 3$

Similar to before
$$2^{2^n} > c \cdot 2^n$$

$\therefore$ $2^{2^n} \notin O(2^n)$ as 2^{2^n} asyptotically dominates 2^n.

1c. No, $n^{1.01} \notin O(\log^2 n)$ as polynomial growth is ~~always~~ typically larger than lograthmic growth.

$\Rightarrow$ This can be proved by disproving
$$n^{1.01} \leq c \cdot \log^2 n \quad \text{for any } c \text{ and } n \quad -(1)$$

Here let $c = 3$ and $n = 50$

~~LHS:~~

On substituting values of c and n in (1):

$$(50)^{1.01} \leq 3 \cdot (\log 50)^2$$

$$= 51.99 > 8.659$$

$\therefore$ $n^{1.01} \leq c \cdot \log^2 n$ has been disproved hence proving $n^{1.01} \notin O(\log^2 n)$

1 d. Yes, $n^{1.01} \in \Omega(\log^2 n)$. To prove this statement we must prove this:

$n^{1.01} \geqslant c \cdot \log^2 n$ if there exists a c and n_0 such that $n \geqslant n_0$.

Take the example $c=3$ and $n=30$.

$n^{1.01} = 31.037$.
$c \cdot \log^2 n = 3 \cdot (\log 30)^2 = 6.54$.

→ Here $n^{1.01} > c \cdot \log^2 n$ and for any $n > 30$ and with $c=3$ this applies.

→ Similarly for any other c and n_0 this equation can proven true.

$\therefore n^{1.01} \in \Omega(\log^2 n)$.

1 e. Yes, $\sqrt{n} \in O(\log^3 n)$. This can be proved by finding a c and n_0 such that $\sqrt{n} \leqslant c \cdot \log^3 n$ for any $n \geqslant n_0$.

Let $c = 3$ and $n_0 = 11$
→ $\sqrt{n_0} = \sqrt{11} = 3.31$
$3 \cdot \log^3 n_0 = 3 \cdot (\log 11)^3 = 3.38$.

Here for $c=3$ and $n_0 = 11$
$\sqrt{n} \leqslant c \cdot \log^3 n$

→ This behaviour can be proven for any $n > 11$ and ~~$c \geqslant 2$~~ for $c=3$.
$\therefore \sqrt{n} \in O(\log^3 n)$.
for any $c \geqslant 2$.

1f. No $\sqrt{n} \notin \Omega(\log^3 n)$ as proved previously when $c=3$ and $n_0 \geq 11$ $\sqrt{n} \leq c \cdot \log^3 n$ and therefore $\log^3 n$ can not be used as a lower bound for $\sqrt{n}$.

~~1g.~~

1g.

→ As per the defination of ~~[illegible]~~ "little o". If there are two functions $f(n)$ and $g(n)$ then $f(n) \in o(g(n))$ if for every c and ~~[illegible]~~ any n_0 the equation:

$$f(n) < c_1 \cdot g(n) \text{ is satisfied for } n \geq n_0$$

→ This defination is analogous for "little w". Except for every c and a n_0

$$f(n) > c_2 \cdot g(n) \text{ for every } n \geq n_0$$

→ As there ~~are no~~ is no point at which $f(n) = c_1 \cdot g(n)$ or $c_2 \cdot g(n)$. There is no shared common point.

→ ~~This~~ ∴. Therefore there no common values between $o(g(n))$ and $w(g(n))$

→ Hence, $o(g(n)) \cap w(g(n))$ is an empty set.

2.

2b.

→ The function takes two numbers as arguments and returns the larger number between the two numbers

→ For example: if the two chosen numbers were 125and 28. Then 125 would be returned.

2C.

→ This algorithm is similar to the Eucledian algorithm which depends on the number of recursive steps it takes for ~~a % b~~ ~~w~~ a mod b where a is the smaller number to reach zero.

→ The work and span of this algorithm is ~~O(log(min(a,~~ O(log(min(a,b))).

→ Similarly in this algorithm y mod x is being performed where y is the larger number. Although this still means that the smaller number is being driven to zero.

→ ~~There fore~~ Therefore ~~$~~ work and span of this algorithm is O(log(min(a,b))) where a and b are two numbers.

8.

3b.

→ The iterative method works on the basis of sequentially identifying all possible combinations of a key in the array.

→ In almost any case it will iterate through the array so the work and span will both be O(n).

3d.

→ The function works based on splitting the array into two halves and using recursion for them.

→ Therefore the Work = $2W(n/2) + 1$

Here $W(n/2)$ is the work done on one array. Constant time is used to represent the work done before recursion.

→ We can use the masters theorem here:

$$T(n) = aT(n/b) + f(n)$$

Here $a=2$, $b=2$ and $f(n)=1$.

→ As $f(n)=1$, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$.

→ Therefore work done is O(n). Our span will also be O(n) as there is no parallelization.

3e

→ The work will be the same as before which is $O(n)$

→ The span accounts for parallelization so:

$$S(n) = S(n/2) + 1$$

→ In this specific application the divide and conquer method divides our array into branches of two and merges them. This is a direct use of a binary tree.

→ Therefore our span is $O(\log n)$.