Question 1:

<u>In ascending order of Big O growth</u>

$4lg(lg(n))$

$4lg(n)$

$5n$

$n^{n^{1/5}}$

n^4

$n^{lg(n)}$

$(lg(n))^{5lg(n)}$

$(n/4)^{n/4}$

$n^{n/4}$

5^n

5^{5n}

4^{n^4}

4^{4^n}

5^{5^n}

Question 2:

a)

$A(n) = n^2(A(n-1)) = n^2(n-1)^2(A(n-2)) = ... = n^2(n-1)^2 \cdots 2^2(1) = (n!)^2$

Thus $A(n) \in \theta((n!)^2)$

b)

$C(n) = 8C(\lfloor n/2 \rfloor) + 2n^3 + 4n,\ where\ C(0)\ = 6$

Let $a = 8,\ b = 2,\ f(n) = 2n^3 + 4n,\ f(n) = \theta(n^3),\ c = 3$

It follows that $log_b a = log_2 8 = 3 = c$

Thus by Master Theorem: $C(n) = \theta(n^{log_b a} log_b n) = \theta(n^3 log_2 n)$

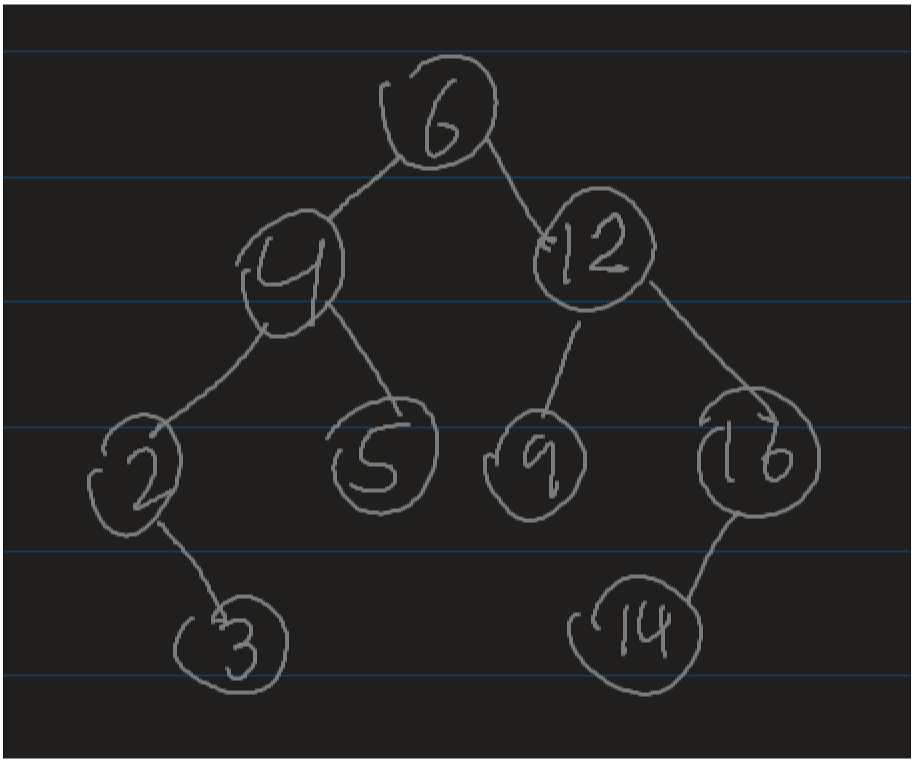
Question 3:

If $f(n) \in O(g(n))$ then by def of Big O
$\exists c,\ n_0 \in N,\ \forall n > n_0,\ 0 \le f(n) \le cg(n)$
$1 \le f(n) \le cg(n)$
$lg(1) \le lg(f(n)) \le lg(cg(n))$
$0 \le lg(f(n)) \le lg(c) + lg(g(n))$

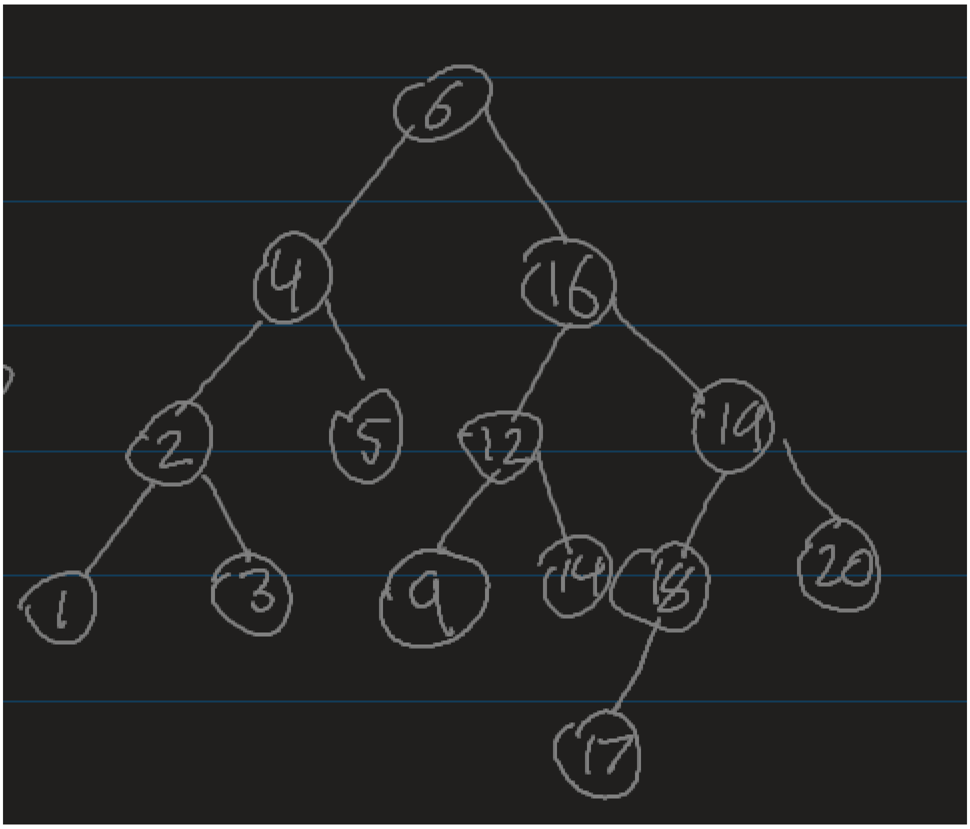
$Let\ c' = 1 + lg(c)$
$c'lg(g(n)) = lg(g(n)) + lg(c)lg(g(n)) \ge lg(g(n)) + lg(c),\ [lg(g(n)\ >=\ 0]$
Thus $0 \le lg(f(n)) \le c'lg(g(n))$, for $n \ge n_0$ and by definition
$lg(f(n)) \in O(lg(g(n))$

Question 4:

After inserting in order 5, 4, 6, 9, 12, 16, 14, 2, 3:



After inserting in order 1, 20, 19, 18, 17:



After deleting 1, 2, 3, 4, 5, 12, 6 in order:

Question 5:

Consider an abstract datatype RECENT that keeps track of recently accessed values.

In this ADT, we consider a set $S = \{1, \ldots, n\}$ and a subset R of S, of size m. Initially, $R = \emptyset$. The only operation is access, which takes a parameter $i \in S$. It adds i to R and, if this causes the size of R to become bigger than m, removes the element from R that was least recently the parameter of an access operation.

Give a data structure for this abstract data type that uses $O(m \log n)$ space (measured in bits) and has worst case time complexity $O(\log m)$.

Justify that your data structure is correct and satisfies the desired complexity bounds.

Assuming each entry takes log(n) bits to store an element of *S*:

Consider a AVL tree to store *R* and an array to store the order

Since there will always be *m* elements in *R*, this ADT will use 2mlog(n) space, which is O(mlog(n))

We know the AVL tree takes O(log(m)) time to insert since the amount of items in the tree is *m*, and we will append this element to the beginning of the array which will take constant time.

However, we should consider the case where it must remove from *R* because the size gets too large. We know the AVL tree will take O(log(m)) to delete, and for the array, we will simply remove the mth element, as it was the least recently inserted element in the array, this should also take O(1) time.