1. Mark the following as `True` or `False`. Briefly but convincingly justify all of your answers using the definition of $O(.)$, $\Theta(.)$ and $\Omega(.)$

   1. $n = O(n\ log(n))$: **True**

      Solution:

      Given: $T(n) = n$

      We know that

      $$1 \leqslant log(n), \quad \text{for all n} \geqslant 2$$

      On multiplying both sides with n we will get:

      $$n \leqslant nlog(n), \quad \text{for all n} \geqslant 2$$

      Since the above line is the summary / definition of $O(\cdot)$ giving us

      $$T(n) = O(nlog(n)), \quad \text{with } n_o = 2 \text{ , c} = 1$$

      Which proves the given statement is `True`.

   2. $n^{1/log(n)} = \Theta(1)$: **True**

      Solution:

      Given:

      $$T(n) = n^{1/log_2(n)} \tag{1}$$

      Since $log_2(2) = 1$, we will replace it in eqn. (1)

      $$T(n) = n^{log_2(2)/log_2(n)} \tag{2}$$

      Applying property of logarithm: $\dfrac{1}{log_b(a)} = log_a(b)$

      $$T(n) = n^{log_n(2)}$$

      Since $log_n(n) = 1$ we get

      $$T(n) = 2^1 = constant$$

      Taking $c_1 = 1$ and $c_2 = 3$:

      $$c_1 \cdot 1 \leqslant T(n) = 2 \leqslant c_2 \cdot 1, \quad \text{for all n} \geqslant 0$$

      Since the above line is the summary / definition of $\Theta(\cdot)$ giving us

      $$T(n) = \Theta(1) \quad \text{with } n_o = 0 \text{ , } c_1 = 1 \text{, } c_2 = 3$$

      Which proves the given statement is `True`.

   3. If $f(n) = \begin{cases} 5^n & \text{if } n < 2^{1000} \\ 2^{1000}n^2 & \text{if } n \geqslant 2^{1000} \end{cases}$ and $g(n) = \dfrac{n^2}{2^{1000}}$,
      then $f(n) = O(g(n))$. **True**

      We know that:

      $$n^2 < 2n^2, \text{ for } n > 2$$

      Multiplying by 2^{2000} on both sides:

      $$2^{2000}n^2 < 2^{2001}n^2, \text{ for } n > 2$$

      $$2^{1000}n^2 < 2^{2001} \cdot (\frac{n^2}{2^{1000}}), \text{ for } n > 2$$

      $$T(n) < 2^{2001} \cdot (g(n)), \text{ for } n > 2^{1000}$$

According to the definition of the $O(\cdot)$ we can say that $T(n) = O(g(n))$, for $c = 2^{2001}, n_0 = 2^{1000}$. Hence The Given statement is `True`.

4. For all the possible functions $f(n)$, $g(n) \geqslant 0$, if $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$: **False**
   Let us consider the following statement to be true:

$$\forall f(n), g(n) \geqslant 0, (f(n) = O(g(n))) \implies (2^{f(n)} = O(2^{g(n)})) \tag{1}$$

   We Will disprove this statement by giving a contradictory example
   Let's consider $f(n) = 2n$ and we know that $f(n) = O(n)$ for $c = 3, n \geqslant 0$ which gives us $g(n) = n$
   Now putting the values of $g(n)$ and $f(n)$ in *eqn.* 1:

$$2^{f(n)} = O(2^{g(n)})$$

$$2^{2n} = O(2^n)$$

   By the definition of $O(\cdot)$ we can say that:

$$2^{2n} \leqslant c \cdot 2^n \quad \text{for some } n_0, c \text{ and } n \geqslant n_0$$

$$2^n \leqslant c \qquad \text{for some } n_0, c \text{ and } n \geqslant n_0$$

   But we know that 2^n is an unbounded function and hence there cannot exist a constant *c* which will be always greater than or equal to the function, which contradicts the argument we made which means our assumption was wrong and
   $2^{f(n)}$ is not $O(2^{g(n)})$
   Hence the given statement is `False`

5. $5^{log\ log(n)} = O(log(n)^2)$: **False**
   Solution:

   Let us consider the given argument to be correct:

$$T(n) = 5^{log\ log(n)} = O(log(n)^2) \tag{1}$$

   Applying property of logarithm: $n^{log_b(a)} = a^{log_b(n)}$

$$T(n) = log(n)^{log_2 5} \approx log(n)^{2.3} \tag{2}$$

   The given argument states that $T(n) = O(log(n)^2)$. So according to definition of $O(\cdot)$ and *eqn.* 2 we can say that

$$log(n)^{2.3} \leqslant c \cdot log(n)^2 \text{ for some } n \geqslant n_0$$
$$log(n)^{0.3} \leqslant c \text{ for some } n \geqslant n_0$$

   But we know that $log(n)$ is an unbounded function and hence there cannot exist a constant *c* which will be always greater than the function, which contradicts the argument we made which means our assumption was wrong and $T(n)$ is not $O(log(n)^2)$
   Hence the given statement is `False`

6. $n = \Theta(100^{log(n)})$: **False**
   Using property of logarithm we can rewrite it as:

$$n = \Theta(n^{log(100)}) \approx \Theta(n^{6.6}) \tag{1}$$

   Since $\Theta(\cdot)$ means both $\Omega(\cdot)$ and $O(\cdot)$ we first check $\Omega(\cdot)$:
   We will use a proof by contradiction to disprove this. Suppose as per the definition of $\Omega(\cdot)$, there is some n_0 and some $c > 0$ such that for all $n \geqslant n_0$,

   $n \geqslant c \cdot n^{6.6}$.

   Let us choose $n = max\{1/c, n_0\} + 1$.
   Then $n > n_0$ but we will have $n > 1/c$ which implies that $c \cdot n^2 > n$ and since $c \cdot n^{6.6} > c \cdot n^2, \text{ for } n > 1$, it also implies that $c \cdot n^{6.6} > n$. Which is a Contradiction to our assumption above about $\Omega(\cdot)$.
   Since $\Omega(\cdot)$ is not correct then $\Theta(\cdot)$ is also Wrong, Hence the given statement is `False`.

2. **n-naught not needed**. Suppose that $T(n) = O(n^d)$, and that $T(n)$ is never equal to ∞. Prove rigorously that there exists a c so that $0 \leqslant T(n) \leqslant c \cdot n^d$ for all $n \geqslant 1$. That is, the definition of $O(\cdot)$ holds with $n_0 = 1$.

   Solution:

   Let $T(n)$ be an arbitrary polynomial function of order *d* and $T(n) = O(n)$:

   $$T(n) = c_0 + c_1 \cdot n^1 + c_2 \cdot n^2 + \cdots + c_{d-1} \cdot n^{d-1} + c_d \cdot n^d$$

   We know that

   $$|T(n)| = |c_d \cdot n^d + c_{d-1} \cdot n^{d-1} + \cdots c_0| \leqslant |c_d| \cdot n^d + |c_{d-1}| \cdot n^{d-1} + \cdots |c_0|$$
   $$|T(n)| \leqslant |c_d| \cdot n^d + |c_{d-1}| \cdot n^{d-1} + \cdots |c_0|$$

   we can also say that:

   $$|T(n)| \leqslant |c_d| n^d + |c_{d-1}| n^d + \cdots + |c_1| \cdot n^d + |c_0| \cdot n^d, \text{for all } n \geq 1$$

   $$T(n) \leqslant \sum_{i=0}^{d} |c_i| \cdot n^d, \text{for all } n \geq 1$$

   Since the above line is the summary / definition of $O(\cdot)$ which states that

   $$T(n) = O(n^d)$$

   Which asserts our assumption, meaning our assumption regarding Big-Oh of $T(n)$ was correct. Hence Proved.

3. Solve the following recurrence relations; i.e. express each one as $T(n) = O(f(n))$ for the tightest possible function $f(n)$, and give a short justification. Be aware that some parts might be slightly more involved than others. Unless otherwise stated, assume $T(1) = 1$.

   1. $T(n) = 2T(n/2) + 3n$
      We can write the above statement as

      $$T(n) = 2T(n/2) + O(n)$$

      We apply the master theorem with $a = b = 2$ and with $d = 1$. We have $a = b^d$, and so the running time is $O(n^d \cdot log_2(n)) = O(n \cdot log_2 n)$.

   2. $T(n) = 3T(n/4) + \sqrt{n}$
      We can write the above statement as

      $$T(n) = 3T(n/4) + O(n^{\frac{1}{2}})$$

      We apply the master theorem with $a = 3, b = 4$ and with $d = \frac{1}{2}$. We have $a > b^d$, and so the running time is $O(n^{log_b(a)}) = O(n^{log_4(3)})$.

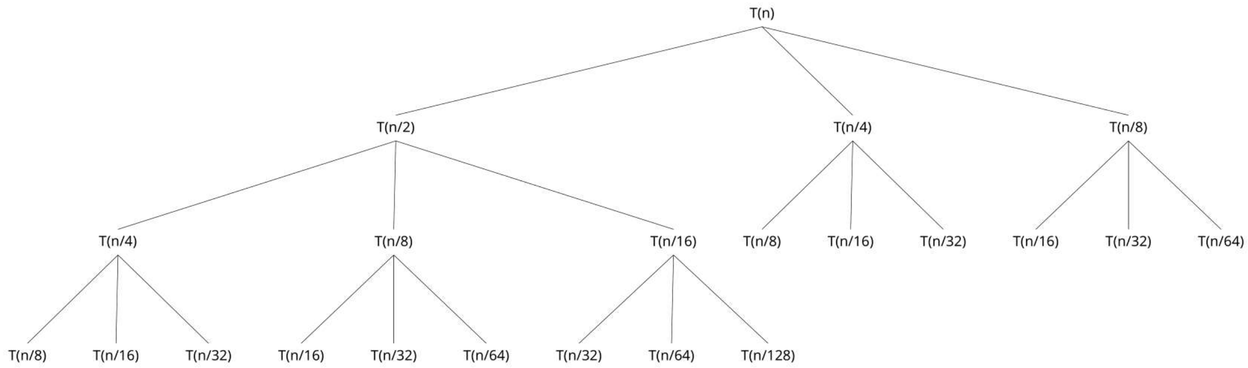
   3. $T(n) = 7T(n/2) + \Theta(n^3)$
      We can write the above statement as

      $$T(n) = 7T(n/2) + \Theta(n^3)$$

      We apply the master theorem with $a = 7, b = 2$ and with $d = 3$. We have $a < b^d$, and so the running time is $O(n^d) = O(n^3)$.

   4. $T(n) = 4T(n/2) + n^2 \cdot log(n)$
      We can write the above statement as

      $$T(n) = 4T(n/2) + n^2 \cdot log(n)$$



We Can See from the tree that at Each level the Problem divides into 4 sub-problems at each level So we will use Substitution : At Second Level:

$$T(n) = 4T(n) + n^2 log(n)$$

$$T(n) = 4(T(n/2) + n^2/4 \cdot log(n/2)) + n^2 log(n)$$

$$T(n) = 16(T(n/4) + n^2/16 \cdot log(n/4)) + n^2(log(n) + log(n/2))$$

$$\dots$$

$$T(n) = n^2(log(n) + log(n/2) + log(n/4) + log(n/8) + \dots)$$

$$T(n) = n^2 \cdot \sum_{i=0}^{log(n)} log\left(\frac{n}{2^i}\right) \approx n^2 \cdot log(n) \cdot log(n+1)$$
$$T(n) = n^2 \cdot (log(n))^2$$

By the definition of of $O(\cdot)$ we came to the conclusion that: $T(n) = O((n \cdot log(n))^2)$, where $n_0 = 2, c = 2$.

5. $T(n) = 2T(n/3) + n^c$, where $c \geqslant 1$ is a constant
We can write the above statement as

$$T(n) = 2T(n/3) + n^c$$

We apply the master theorem with $a = 2, b = 3$ and with $d = c$, where $c \geqslant 1$. We have $a < b^c$, $\forall c \geqslant 1$ and so the running time is $O(n^d) = O(n^c)$.

6. $T(n) = 2T \cdot (\sqrt{n}) + 1$, where $T(2) = 1$
Let us consider $n = 2^{2^k}$
We can write the above statement as

$$T(2^{2^k}) = 2 \cdot T(\sqrt{2^{2^k}}) + 1$$

Which will be

$$T(2^{2^k}) = 2 \cdot T(2^{2^{k-1}}) + 1$$

Now we can apply substitution here

$$T(2^{2^k}) = 2(2^{2^{k-1}}) + 1$$

$$T(2^{2^k}) = 2(2 \cdot T(2^{2^{k-2}}) + 1) + 1$$

$$T(2^{2^k}) = 2^2(2 \cdot T(2^{2^{k-3}}) + 1) + 1 + 2$$

$$T(2^{2^k}) = 2^3(2 \cdot T(2^{2^{k-4}}) + 1) + 1 + 2 + 4$$

$$\dots$$

$$T(2^{2^k}) = 2^k \cdot T(2^{2^0}) + 1 + 2 + 4 + \dots + 2^{k-1}$$
$$T(2^{2^k}) = 2^k + 2^{k-1} + \dots + 4 + 2 + 1$$
$$T(2^{2^k}) = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

Which means that

$$T(2^{2^k}) \leqslant 2^{k+1}$$

$$\text{On substituting back } 2^{2^k} = n$$

$$T(n) \leqslant log_2(log_2(2^{k+1})) = log_2(k+1)$$

$$T(n) \leqslant log_2(k+1) \leqslant log_2(k^2) \ \forall \ k \geqslant 2$$

$$T(n) \leqslant log_2(k^2) = 2 \cdot log_2(k) \ \forall \ k \geqslant 2$$

Since the above line is the summary / definition of $O(\cdot)$ giving us

$$T(n) = O(log(n)) \text{ for } c = 2, n_0 = 2$$

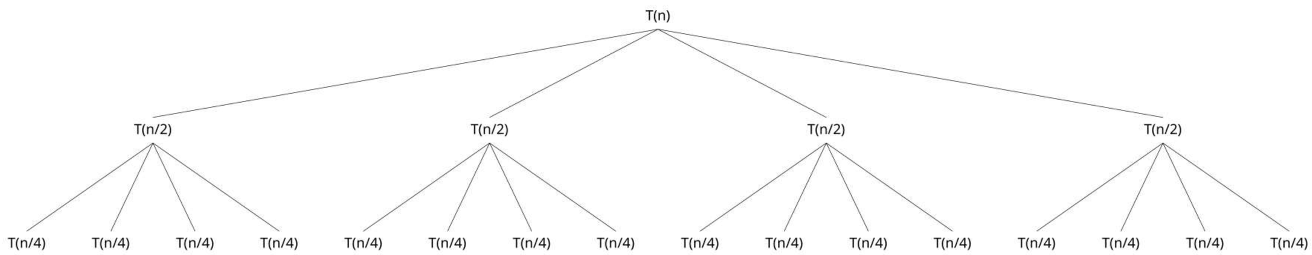
Hence the Algorithm has Complexity $O(log(n))$.

4. Different-sized sub-problems. (6 points) Solve the following recurrence relation. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$ where $T(1) = 1$.
Given Recurrence Relation:

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n \tag{1}$$

Here we use substitution. Upon replacing the values of $T(n/2), T(n/4)$ and $T(n/8)$ we will get:

$$\begin{aligned} T(n) &= \{T(n/4) + T(n/8) + T(n/16) + n/2\} \\ &+ \{T(n/8) + T(n/16) + T(n/32) + n/4\} \\ &+ \{T(n/16) + T(n/32) + T(n/64) + n/8\} + n \end{aligned} \tag{1}$$



At the Top Level we have n operations,

At the Second Level we have $n/2 + n/4 + n/8 = 7n/8$ operations

At the Third Level we have
$\{n/4 + n/8 + n/16\} + \{n/8 + n/16 + n/32\} + \{n/16 + n/32 + n/64\} = 7n/8(n/2 + n/8 + n/16) = 49n/64$

Similarly For each subsequent level the no. of operations will decrease down by a factor of $7/8$
This happens because in the recurrence relation the sum of no. of operations of child Processes are $1/2 + 1/4 + 1/8 = 7/8$ times the parent Process. Which will make the net number of operations:

$$T(n) = n + \frac{7n}{8} + \frac{49n}{64} + \cdots \tag{1}$$

Considering it as an Infinite G.P. the total number of operations will be:

$$T(n) = n \cdot \sum_{\infty}^{i=0} \frac{7^i}{8^i} \tag{1}$$

$$T(n) = \frac{n}{1 - 7/8} = 8 \cdot n$$

Since $T(n) = 8n \leqslant 9$, for all $n \geqslant 1$. By the definition of $O(\cdot)$

We can say that $T(n) = O(n)$.

5. What's wrong with this proof? (9 points) Consider the following recurrence relation: $T(n) = T(n/5) + 10 \cdot n$ for n=5, where $T(0) = T(1) = T(2) = T(3) = T(4) = 1$. Consider the following three arguments.

   1. Claim: $T(n) = O(n)$. To see this, we will use strong induction. The inductive hypothesis is that $T(k) = O(k)$ for all $5 \leqslant k < n$. For the base case, we see $T(5) = T(0) + 10 \cdot 5 = 51 = O(1)$. For the inductive step, assume that the inductive hypothesis holds for all $k < n$. Then

      $$T(n) = T(n-5) + 10 \cdot n$$

      and by induction $T(n-5) = O(n/5)$, so

      $$T(n) = O(n-5) + 10 \cdot n = O(n)$$

      This establishes the inductive hypothesis for n. Finally, we conclude that $T(n) = O(n)$ for all n

   2. Claim: $T(n) = O(n)$. To see this, we will use the Master Method. We have $T(n) = a \cdot T(n/b) + O(n^d)$, for $a = d = 1$ and $b = \dfrac{1}{1 - 5/n}$.
      Then we have that $a < b^d$ (since $1 < 1/(15/n)$ for all $n > 0$), and the master theorem says that this takes time $O(n^d) = O(n)$.

   3. Claim: $T(n) = O(n^2)$. Imagine the recursion tree for this problem. (Notice that it's not really a "tree"," since the degree is 1). At the top level we have a single problem of size *n*. At the second level we have a single problem of size $n - 5$. At the *t*'th level we have a single problem of size $n - 5t$, and this continues for at most $t = \lfloor n/5 \rfloor + 1$ levels. At the t'th level for $t \leqslant \lfloor n/5 \rfloor$, the amount of work done is $10(n - 5t)$. At the last level the amount of work is at most 1. Thus the total amount of work done is at most

      $$1 + \sum_{t=0}^{\lfloor n/5 \rfloor} 10(n - 5t) = O(n^2)$$

   - Which, if any, of these arguments are correct?
     All the arguments are wrong.
   - For each argument that you said was incorrect, explain why it is incorrect. If you said that all three were incorrect, then give a correct argument.
     In the Provided Proofs there are following mistakes:

     1. In First Argument it was said that in Base case $T(5) = T(0) + 10 \cdot 5 = 51 = O(1)$ which can not be said about $T(n)$ since we put value of *n* in $T(n)$ the output is the value of the function and we cannot say anything about the complexity.
     2. In the Second Argument it is mentioned $b = \dfrac{1}{1 - 5/n}$, which is wrong since the value of *b* is not constant. Which is the reason we cannot use Master Theorem.

3. In the Third Argument $1+\sum_{t=0}^{\lfloor n/5\rfloor} 10(n-5t)=O(n^2)$ the Summation will range from *)* to *n/5 - 1*. which is given wrong.

Correct Answer:
Claim: $T(n)=O(n^2)$. Considering the recursion tree for this problem. At the top level we have a single problem of size *n*. At the first level we have a single problem of size $n-5$. At the *(t+1)*'th level we have a single problem of size $4n-5t$, and this continues for at most $t=\lfloor n/5\rfloor$ levels. At the *(t+1)*'th level for $t<\lfloor n/5\rfloor$, the amount of work done is $10(n-5t)$. At the last level the amount of work is at most 1. Thus the total amount of work done is at most

$$1+\sum_{t=0}^{\lfloor n/5\rfloor-1} 10(n-5t)=O(n^2)$$