Time Complexity Questions

Latest Time Complexity MCQ Objective Questions

Question 1:

View this Question Online >

The recurrence equation T(n) = T(n/2) + 1 represents the time complexity of which algorithmic paradigm?

1. Divide and Conquer
2. Greedy Algorithms
3. Dynamic Programming
4. Brute Force

Answer (Detailed Solution Below)

Option 1 : Divide and Conquer

Time Complexity Question 1 Detailed Solution

The correct answer is **Divide and Conquer**

Key Points

- **Divide and Conquer:**
  - This paradigm involves breaking down a problem into smaller sub-problems of the same type.
  - The solution to the original problem is then composed of the solutions to the sub-problems.
  - The recurrence relation $T(n) = T(n/2) + 1$ is characteristic of Divide and Conquer algorithms, where a problem of size 'n' is divided into two sub-problems of size 'n/2', and the "+ 1" term represents the work done at each level or merging step.

Additional Information

- **Greedy Algorithms:**
  - Greedy algorithms make locally optimal choices at each stage with the hope of finding a global optimum.
  - They don't necessarily involve recurrence relations like the one mentioned; instead, they typically involve making the best decision at each step without considering the overall impact.
- **Dynamic Programming:**
  - Dynamic Programming is an algorithmic paradigm where a problem is broken down into smaller overlapping sub-problems, and the solutions to these sub-problems are memoized or stored to avoid redundant computations.
  - The recurrence relation for dynamic programming often involves combining solutions to sub-problems, and it can be expressed in terms of the solutions to smaller instances of the same problem.
- **Brute Force:**
  - Brute Force algorithms exhaustively explore all possible solutions to a problem without using any clever optimization techniques.
  - They do not typically involve recurrence relations in the same systematic way as Divide and Conquer or Dynamic Programming.

Question 2:

View this Question Online >

An algorithm with three nested loops will have a Big-O efficiency of (a size on n).

1. $O(n^3)$
2. $O(3n)$
3. $O(n^4)$
4. $O(n^2)$
5. $O(n)$

Answer (Detailed Solution Below)

Option 1 : $O(n^3)$

Time Complexity Question 2 Detailed Solution

Algorithm Analysis with Big-O Notation:

- Bit-O is a metric used to find algorithm complexity.
- It signifies the relationship between the input to the algorithm & the steps required to execute the algorithm.
- It is denoted by big "O" followed by opening & closing parenthesis, the relationship between the input and steps taken by the algorithm is presented using n.
- $O(n^c)$: Time complexity of nested loops is equal to the number of times the innermost statement is executed.
- An algorithm with three-nested loops will have $O(n^3)$

Question 3:

View this Question Online >

Which of the following statements is/are true?

1. Recurrence relation for number of comparisons in binary search is T(n) = T(n/2) + 2
2. Recurrence relation of merge sort in worst case is T(n) = 2T(n/2) + Θ(n)
3. Recurrence of quicksort in worst case is T(n) = 2T(n/2) + O(1)
4. 3-way merge sort is T(n) = 3T(n/3) + O(n)

Answer (Detailed Solution Below)

Option :

Time Complexity Question 3 Detailed Solution

The correct answer is **option 1, option 2 and option 4.**

Concept:

Option 1:

Let T (n) be the number of comparisons needed in a binary search of a list of n elements.
The recurrence relation is
T(n) = T(n/2) + 2
(2 comparisons because one to determine which half of the list to use and the other to find whether any element of the list remain).

Option 2:

The recurrence relation for merge sort is

$T(n)=2T(n/2) + \Theta(n)$

Option 3:

The worst case of quicksort occurs when the array to be sorted is already in a sorted order. In such case the we need $\Theta(n)$ to fix the pivot and then the problem is divided into two parts, such that one part has (n-1) elements and the other part has 1 element. Therefore the recurrence for worst case of quicksort is
$T(n)= T(n-1) + \Theta(n)$

Option 4:

Three way merge sort is just like two way merge sort, except every range in the array is recursively divided into three parts. So for a subproblem with size n, we need O(n) time to merge them and make a single array and then we divide this subproblem further to subproblems of size (n/3) each. Hence the recurrence is
$T(n)= 3T(n/3) + \Theta(n)$

Hence the correct answer is **option 1, option 2 and option 4.**

Question 4:

View this Question Online >

What will be the time complexity for the following recurrence relation?

$T(n)=8T(n^{1/2})+(\log n)^2$ if $n>2$

$= 1$, Otherwise

1. $\Theta((\log n)^3)$
2. $\Theta(n^2)$
3. $\Theta(n \log n)$
4. $\Theta((\log n)^4)$

Answer (Detailed Solution Below)

Answer (Detailed Solution Below)

Option :

Time Complexity Question 4 Detailed Solution

The correct answer is **option 1 and option 4.**

Concept:

The given recurrence relation is,

T(n) = $8T(n^{1/2}) + (\log m)^2$

Let's take n = 2^m

$T(2^m) = 8T(2^{m/2}) + m^2$

Let's take $T(2^m)$ as S(m)

s(m) $= 85\left(\frac{m}{2}\right) + m^2$

Apply Master's Theorem,

a = 8, b = 2, k = 2, p = 0

Case: a > b^k,

$s(m) = m^{\log_b a} = m^{\log_2 8} = m^3$

Put m as $\log_2 n$

$\therefore T(n) = (\log_2 n)^3$

$T(n) = \theta(\log_2 n)^3$

$\Theta((\log n)^4)$ is more than $\Theta((\log n)^3)$ so it is also time complexity for the given recurrence relation.

Hence the correct answer is **option 1 and option 4.**

Question 5:

View this Question Online >

In the following table, the left column contains the names of standard graph algorithms and the right column contains the time complexities of the algorithms. Here, n and m are number of vertices and edges, respectively. Match each algorithm with its time complexity.

List I		**List II**	
Standard graph algorithms		**Time complexities**	
A.	Bellman-Ford algorithm	I.	O(m*log n)
B.	Kruskal's algorithm	II.	$O(n^3)$
C.	Floyd-Warshall algorithm	III.	O(n*m)
D.	Topological sorting	IV.	O(n + m)

Choose the correct answer from the options given below :

1. A - III, B - I, C - II, D - IV
2. A - II, B - IV, C - III, D - I
3. A - III, B - IV, C - I, D - II
4. A - II, B - I, C - III, D - IV

Answer (Detailed Solution Below)

Option 1 : A - III, B - I, C - II, D - IV

Time Complexity Question 5 Detailed Solution

Solution :

Bellman -Ford algorithm (Finds shortest paths from a single source node to all of the other nodes in a weighted digraph) : O(mn) ,where, n is no of edges , m is no of nodes.

Kruskal's algorithm (Using Greedy approach it find the minimum cost spanning tree) - O(m*log n)

Floyd -Warshall algorithm (Find shortest paths in a directed weighted graph with positive or negative edge weights) - O(n3)

Topological sorting (Find linear ordering of vertices for Directed Acyclic Graph (DAG)) - O(n + m)

Therefore ,option 1 is correct.

Top Time Complexity MCQ Objective Questions

Question 6

View this Question Online >

Consider the recurrence function $T(n) = \begin{cases} 2T(\sqrt{n}) + 1, & n > 2 \\ 2, & 0 < n \le 2 \end{cases}$ Then T(n) in terms of Θ notation is

1. Θ (log log n)
2. Θ (log n)
3. Θ (√n)
4. Θ (n)

Answer (Detailed Solution Below)

Option 2 : Θ (log n)

Time Complexity Question 6 Detailed Solution

$$T(n) = 2\,T(\sqrt{n}) + 1$$

Put n= 2^m, m = $\log_2 n$

$$T\left(2^m\right) = 2T\left(2^{\frac{m}{2}}\right) + 1$$

$$Put\ T\left(2^m\right) = S\left(m\right)$$

$$S\left(m\right) = 2\ S\ \left(\frac{m}{2}\right) + 1$$

Now, calculate $n^{log_b^a}$

$$m^{log_b^a} = m$$

S (m) = m + 1

S(m) = m

Put the value of m

Therefore, T(n) in terms of Θ notation is Θ (logn).

Question 7

View this Question Online >

Consider the following C function.

```
int fun1(int n) {
int i, j, k, p, q = 0;
for (i = 1; i < n; ++i) {
p = 0;
for (j = n; j > 1; j = j/2)
 ++ p;
for (k = 1; k<p; k = k*2)
 ++ q;
}
return q;
```

}

Which one of the following most closely approximates the return value of the function fun1?

1. n^3
2. $n(\log^2)$
3. n log n
4. n log (log n)

Answer (Detailed Solution Below)

Option 4 : n log (log n)

Time Complexity Question 7 Detailed Solution

```
for (i to n) ⟶ n times
    p = 0
    for(j = n; j > 1; j = j/2) ⟶ log n times
        ++p
    for(k = 1; k < p; k = k*2) ⟶ log p times
        ++q
```

$$\therefore p = \log n,\ q = \log p = \log(\log n)$$
$$\therefore q = n.\log(\log n)$$

Question 8

View this Question Online >

Give asymptotic upper and lower bound for T(n) given below. Assume T(n) is constant for $n \leq 2. T(n) = 4T(\sqrt{n}) + log^2 n$

1. T(n) = θ(lg(lg^{2}n)lg n)
2. T(n) = θ(lg^{2}nlgn)
3. T(n) = θ(lg^2 n lg lg n)
4. T(n) = θ(lg(lg n))g n)

Answer (Detailed Solution Below)

Option 3 : T(n) = θ(lg^2 n lg lg n)

Time Complexity Question 8 Detailed Solution

$$T(n) = 4T(\sqrt{n}) + log^2 n$$

Consider n = 2^m

m = $\log_2 n$

n = 2^m

Taking square root

√n = $_2 m/2$

T(2^m) = 4T($2^{m/2}$) + m^2

Put S(m) = T ($2^{m/2}$)

S(m) = 4 S(m/2) + m^2

$S(m) = 4\,S(m/2) + m^2$

Comparing with

$T(m) = a\,T(m/k) + cm^k$

$a = 4, b = 2, k = 2$

$a = b^k = 4$

Now use master's theorem :

$T(m) = O(m^k \log m)$

So, Time complexity will be $O(m^2 \log m)$

Put the value of m

Time complexity will be : $O(\log^2 n \log(\log n))$

Question 9

View this Question Online >

Consider the following C function.

```
int fun(int n) {
        int i, j;
        for (i = 1; i < = n; i++) {
        for (j = 1; j < n; j + = i) {
                    printf (" %d %d", i, j);
        }
        }
}
```

Time complexity of fun in terms of Θ notation is

1. Θ (n√n)

2. Θ (n^2)

3. Θ (nlog n)

4. Θ (n^2log n)

Answer (Detailed Solution Below)

Option 3 : Θ (nlog n)

Time Complexity Question 9 Detailed Solution

We have to check how many times inner loop will be executed here.

For i=1,

j will run 1 + 2 + 3 + ... (n times)

For i=2

j will run for 1,3,5, 7, 9, 11,........... (n/2 times)

For i=3

j will run for 1,4,7,10, 13............... (n/3 times)

So, in this way,

$$T(n) = n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \frac{n}{5} + \frac{n}{6} \ldots + \frac{n}{n}$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} \ldots + \frac{1}{n}\right)$$

So, Time complexity of given program = Θ (n log n)

Question 10 [View this Question Online >](#)

Let f(n) = n and g(n) = $n^{(1 + \sin n)}$, where n is a positive integer. Which of the following statements is/are correct?

I. f(n) = O(g(n))

II. f(n) = Ω(g(n))

1. Only I
2. Only II
3. Both I and II
4. Neither I nor II

Answer (Detailed Solution Below)

Option 4 : Neither I nor II

Time Complexity Question 10 Detailed Solution

Concept:

Sin function value ranges from -1 to + 1. (-1, 0, 1)

Explanation:

Case 1: when sin(n) is -1,

g(n) = $n^{(1 - 1)} = n^0 = 1$

so, for this case f(n) > g(n) i.e. g(n) = O (f(n))

So, statement 1 is incorrect.

Case 2: when sin(n) is +1,

g(n) = $n^{(1 + 1)} = n^2$

so, for this case f(n) < g(n) i.e. f(n) = O(g(n)) but for this, second statement i.e. $f(n) = \Omega(g(n))$ is incorrect.

Both statements are not true for all values of $\sin(n)$.

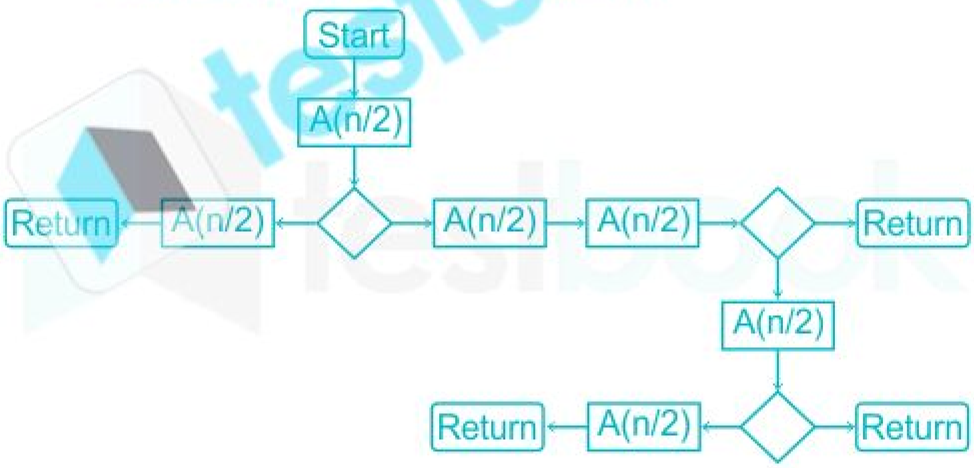
Hence, option 4 is correct answer.

Question 11

View this Question Online >

The given diagram shows the flowchart for a recursive function A(n). Assume that all statements, except for the recursive calls, have O(1) time complexity. If the worst-case time complexity of this function is $O(n^{\alpha})$, then the least possible (accurate up to two decimal position) of α is _______.

Flowchart for Recursive Function A(n)



Answer (Detailed Solution Below) **2.2 - 2.4**

Time Complexity Question 11 Detailed Solution

To find the worst-case time complexity of the function.

Worst case happens in case of recursive. So, find out the recursive calls on the longer route.

There are 5 such recursive calls to A (n/2).

So, recurrence relation will become like:

A (n) = 5A (n/2) + O(1)

where O(1) is constant

By using master's theorem,

a = 5, b =2

$$O\left(n^{\log_2 5}\right) = O(n^{2.32})$$

$$O(n^{\alpha}) = O(n^{2.32})$$

α = 2.32

Question 12 View this Question Online >

Consider the following recurrence relation.

$$T(n) = \begin{cases} T(n/2) + T(2n/5) + 7n & if\ n > 0 \\ 1 & if\ n = 0 \end{cases}$$

Which one of the following option is correct?

1. T(n) = Θ(n log n)
2. T(n) = $\Theta(n^{5/2})$
3. T(n) = $\Theta((\log n)^{5/2})$
4. T(n) = Θ(n)

Answer (Detailed Solution Below)

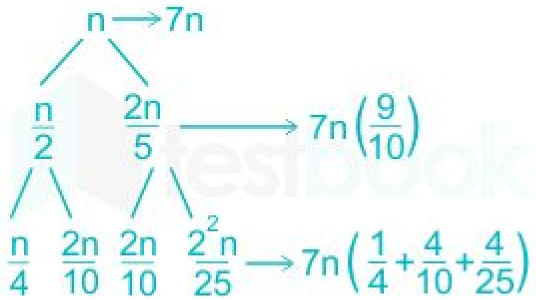
Option 4 : T(n) = Θ(n)

Time Complexity Question 12 Detailed Solution

Answer : Option 4

Explanation :

T(n) = T(n/2) + T(2n/5) + 7n



T(n) = $7n\left(1+\frac{9}{10}+\frac{81}{100}+\ldots+\left(\frac{9}{10}\right)^{\log_2 n}\right)$ (for left most subtree base of log is 2 But for rightmost subtree base will be (5/2))

For rightmost subtree:
T(n) = $7n\left(1+\frac{9}{10}+\frac{81}{100}+\ldots+\left(\frac{9}{10}\right)^{\log_{5/2} n}\right)$

T(n) = $7n(1-n^{\log_2 0.9})$

T(n) = O(n)

and similarly for right most subtree

T(n) = Ω(n)

hence T(n) = θ(n)

Question 13

View this Question Online >

What is the complexity of the following code?

sum=0;

```
for (i=1; I <=  n; i*=2)
        for(j=1; j<=n;j+ +)
                sum++;
```

1. $O(n^2)$
2. O(n log n)
3. O(n)
4. O(n log n log n)

Answer (Detailed Solution Below)

Option 2 : O(n log n)

Time Complexity Question 13 Detailed Solution

Code	Time complexity
sum = 0	O(1)
for (i=1; I <= n; i*=2)	O(logn) because I is incremented exponentially and loop will run for less number of times than n.
for(j=1; j<=n; j++)	O(n) because j is incremented linearly and loop will run for n number of times.
sum++	O(1)

Total time complexity = O(n×logn)

<u>Important Point</u>:

In original ISRO CS 2020, and extra data was present "Which of the following is not a valid string?"

Which made this question ambiguous and hence excluded from evaluation.

Question 14

View this Question Online >

For parameters a and b, both of which are $\omega(1)$, $T(n) = T(n^{1/a}) + 1$, and $T(b) = 1$.

Then T(n) is

1. $\Theta(\log_a \log_b n)$
2. $\Theta(\log_{ab} n)$
3. $\Theta(\log_b \log_a n)$
4. $\Theta(\log_2 \log_2 n)$

Answer (Detailed Solution Below)

Option 1 : $\Theta(\log_a \log_b n)$

Time Complexity Question 14 Detailed Solution

Recurrence relation:

$$T(n) = T\left(n^{\frac{1}{a}}\right) + 1,\ T(b) = 1$$

where 'a' and 'b' are parameters of order $\omega(1)$

Using substitution method to solve recurrences,

$$\mathrm{T}(\mathrm{n}) = \mathrm{T}\left(\mathrm{n}^{\frac{1}{a}}\right) + 1$$

$$= \mathrm{T}\left(\left(\mathrm{n}^{\frac{1}{a^2}}\right) + 1\right) + 1$$

$= \mathrm{T}\left(\mathrm{n}^{\frac{1}{a^2}}\right) + 2$

$= \left(\mathrm{T}\left(\mathrm{n}^{\frac{1}{a^3}}\right) + 1\right) + 2$

$= \mathrm{T}\left(\mathrm{n}^{\frac{1}{a^3}}\right) + 3$

Continuing this for 'm' iterations, we get

$\mathrm{T}(\mathrm{n}) = \mathrm{T}\left(\mathrm{n}^{\frac{1}{a^m}}\right) + \mathrm{m}$

Put $\left(n^{\frac{1}{a^m}}\right) = b$

Taking log on both sides

$\frac{1}{a^m} \log n = \log b$

$a^m = \frac{\log n}{\log b}$

$m = log_a log_b\, n$

Now,

$T(n) = T\left(n^{\frac{1}{a^m}}\right) + m$

$T(n) = b + log_a log_b\, n$

So, the asymptotic order of T(n) is $\Theta(\log_a \log_b n)$

Question 15

View this Question Online >

Match List 1 with List 2 and choose the correct answer from the code given below:

List I (Graph Algorithm)	List II (Time Complexity)
a) Dijkstra’s algorithm	i) $\Theta(E \log E)$
b) Kruskal’s algorithm	ii) $\Theta(V^3)$
c) Floyd-Warshall algorithm	iii) $\Theta(V^2)$
d) Topological sorting	iv) $\Theta(V + E)$

Where V and E are the number of vertices and edges in graph respectively.

1. (a)-(i), (b)-(iii), (c)-(iv), (d)-(ii)
2. (a)-(i), (b)-(iii), (c)-(ii), (d)-(iv)
3. (a)-(iii), (b)-(i), (c)-(ii), (d)-(iv)
4. (a)-(iii), (b)-(i), (c)-(iv), (d)-(ii)

Answer (Detailed Solution Below)

Option 3 : (a)-(iii), (b)-(i), (c)-(ii), (d)-(iv)

Time Complexity Question 15 Detailed Solution

Graph Algorithm	**Time Complexity)**
Dijkstra’s algorithm	$\Theta\ (V^2)$
Kruskal’s algorithm	$\Theta\ (E\log E)$
Floyd-Warshall algorithm	$\Theta(V^3)$
Topological sorting	$\Theta(V + E)$

- Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph and time complexity is $O(V^2)$.
- Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest.
- A topological sort or topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v,u comes before v.
- Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights. A single execution of the algorithm will find the lengths of shortest paths between all pairs of vertices.