



School of Computing

SRM Nagar, Kattankulathur – 603203, Chengalpattu District, Tamil Nadu

Academic Year: 2022-23 (EVEN)

Test: CLA-T1

Date:

Course Code & Title: 21CSC204J Design and Analysis of Algorithms **Duration:** 1 hour 40 min

Year & Sem: II Year / IV Sem

Max. Marks: 50

Course Articulation Matrix: (to be placed)

Course Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	Program Specific Outcomes		
													PSO-1	PSO-2	PSO-3
CO1	2	1	2	1	-	-	-	-		3	-	3	3	1	-
CO2	2	1	2	1	-	-	-	-		3	-	3	3	1	-
CO3	2	1	2	1	-	-	-	-		3	-	3	3	1	-
CO4	2	1	2	1	-	-	-	-		3	-	3	3	1	-
CO5	2	1	2	1	-	-	-	-		3	-	3	3	1	-

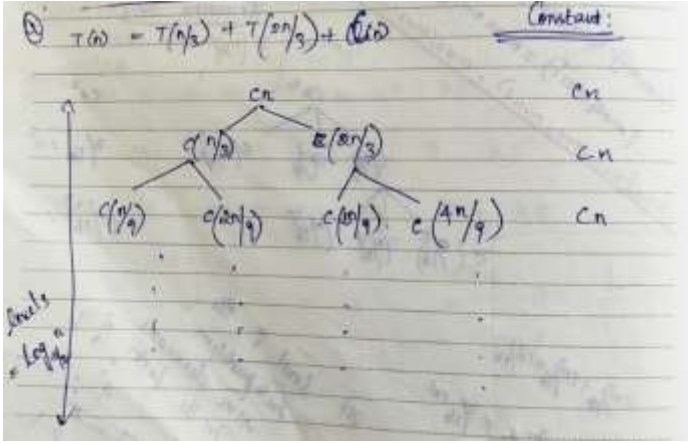
Part - A
(1 x 10 = 10 Marks)

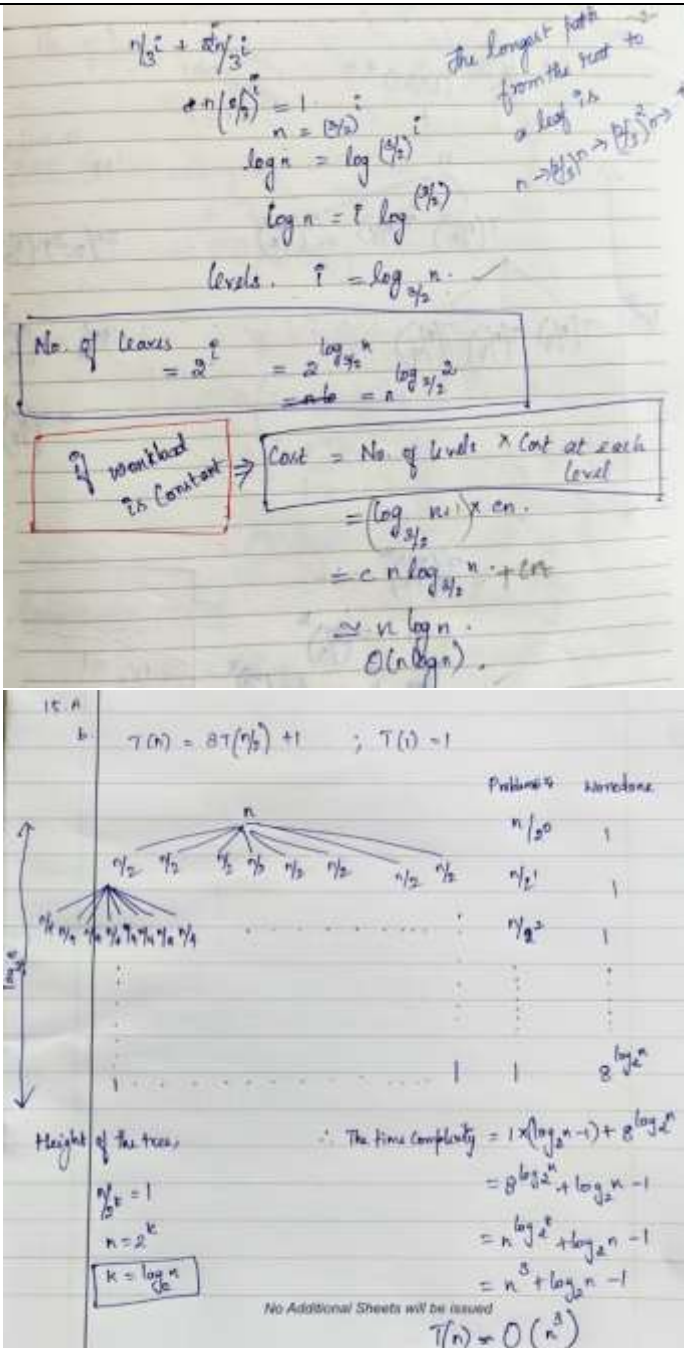
Instructions: Answer all

Q. No	Question	Marks	BL	CO	PO	PI Code
1	<p>Assume that $f(n)$ and $g(n)$ are asymptotically positive, then which of the following relation holds good?</p> <p>a. $f(n)=O(g(n))$ and $g(n) = O(h(n))$ then $f(n)=O(h(n))$</p> <p>b. $f(n)=\Omega(g(n))$ and $g(n) = O(h(n))$ then $f(n)=\Omega(h(n))$</p> <p>c. $f(n)=\Omega(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n)=\Omega(h(n))$</p> <p>d. $f(n)=O(g(n))$ and $g(n) = \Theta(h(n))$ then $f(n)=\Omega(h(n))$</p>	1	1	1	1-4,10,12	1.1, 1.3, 2.4, 12.2.2
2	<p>Consider the equality $\sum_{i=0}^n i^3 = X$ and the following choices for X</p> <p>I. $\theta(n^4)$</p> <p>II. $\theta(n^5)$</p> <p>III. $O(n^5)$</p> <p>IV. $\Omega(n^3)$</p> <p>The equality above remains correct if X is replaced by</p> <p>a. only I</p> <p>b. only II</p> <p>c. I or III or IV but not II</p> <p>d. II or III or IV but not I</p>	1	1	1	1-4,10,12	1.1, 1.3, 2.4, 12.2.2

3	<p>Consider the following segment of C code</p> <pre>int j, n; j = 1; while (j <= n) j = j*2;</pre> <p>The number of comparisons made in the execution of the loop for any $n > 0$ is</p> <p>a. $\log n$ b. $\log n * n$ c. $\log n + 1$ d. n</p>	1	1	1	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
4	<p>The Worst case occur in linear search algorithm when _____</p> <p>a. Item is somewhere in the middle of the array b. Item is not in the array at all c. Item is the last element in the array d. Item is the last element in the array or is not there at all</p>	1	1	1	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
5	<p>The worst case complexity for insertion sort is _____</p> <p>a) $O(n)$ b) $O(\log n)$ c) $O(n^2)$ d) $O(n \log n)$</p>	1	1	1	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
6	<p>If for an algorithm time complexity is given by $O(\log^2 100)$ then upper bounded complexity will be _____</p> <p>a) constant b) polynomial c) linear d) none of the mentioned</p>	1	1	2	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
7	<p>What is the result of the recurrences? Let the recurrence be given by $T(n) = aT(n/b) + f(n)$ and $f(n) = n^c$, when $f(n)$ is $O(n^{\log_b a - e})$ for some constant $e > 0$, using Master's theorem</p> <p>a) $T(n) = O(n^{\log_b a})$ b) $T(n) = O(n^c \log n)$ c) $T(n) = O(f(n))$ d) $T(n) = O(n^2)$</p>	1	1	2	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
8	<p>What is the worst case time complexity of a quick sort algorithm?</p> <p>a) $O(N)$ b) $O(N \log N)$ c) $O(N^2)$ d) $O(\log N)$</p>	1	1	2	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
9	<p>What is the optimal time required for solving the closest pair problem using divide and conquer approach?</p> <p>a) $O(N)$ b) $O(\log N)$ c) $O(N \log N)$ d) $O(N^2)$</p>	1	1	2	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2

10	<p>What is the runtime efficiency of using brute force technique for the closest pair problem?</p> <p>a) $O(N)$ b) $O(N \log N)$ c) $O(N^2)$ d) $O(N^3 \log N)$</p>	1	1	2	1-4,10,12	1.1, 1.3, 2.4, 12.2.2
<p align="center">Part – B (5 x 4 = 20 Marks)</p> <p>Instructions: Answer All the Questions</p>						
11	<p>Let $t(n)=9n^2$. Prove that this is of the order of $O(n^2)$</p> <p>Answer: It can be observed from the given relation that, $9n^2 \leq c * n^2$. This holds good for all values of c greater than 9. Also, assuming $c=9$, the given relation holds good. This is true $n \geq n_0$, where $n_0 = 1$. Therefore $t(n) \in O(n^2)$</p>	5	2	1	1-4,10,12	1.1, 1.3, 2.4, 12.2.2
12	<p>Perform the complexity analysis of the following algorithm segment.</p> <p>a) Algorithm Sample() Begin loop $i=1$ to n loop $j=1$ to m $A(i,j)=0$ End loop End loop End</p> <p>Answer : Ans: $O(n \times m)$ Here, the outer loop runs from 1 to n In the inner loop runs from 1 to m</p> <p>So, time complexity will be $O(nm)$.</p> <p>b) def fun(N): for i in range(1,N+1) $j=N$ while $j>0$: $j/=2$</p> <p>Answer: $O(N \times \log(N))$ Here, the outer loop runs from 1 to N In the inner while loop, the j gets divided by 2 every time. So the inner loop will run for $\log_2(j) = \log_2(N)$ times. As the inner loop runs for N times, our time complexity will be $N \times \log(N)$.</p>	2.5 +2.5	2	1	1-4,10,12	1.1, 1.3, 2.4, 12.2.2
13	<p>Compare and contrast Merge sort and Quick sort</p> <p>Similarities: Divide and conquer: Both algorithms divide the input list recursively into smaller sub-lists, sort the sub-lists,</p>	5	2	2	1-4,10,12	1.1, 1.3, 2.4,

	<p>and then merge them back together in the correct order.</p> <p>Average time complexity: Both have an average time complexity of $O(n \log n)$, making them efficient for large datasets.</p> <p>Differences:</p> <p>Methodology:</p> <p>Merge Sort: Divides the list exactly in half and merges the sorted halves. Requires extra space for temporary storage.</p> <p>Quick Sort: Chooses a pivot element and partitions the list into elements less than and greater than the pivot. More flexible partition ratios possible. Requires no extra space.</p> <p>Complexity:</p> <p>Worst case time complexity:</p> <p>Merge Sort: $O(n \log n)$ (guaranteed)</p> <p>Quick Sort: $O(n^2)$ (possible in certain cases where pivot selection is poor)</p>					12.2. 2
14	<p>Solve using substitution method</p> <p>$T(n)=T(n-1)+3$; $t(1)=4$.</p> <p>Answer:</p> <p>$T(n-1)= (T(n-2)+3)+3$</p> <p>$T(n-2) =((T(n-3)+3)+3)+3$</p> <p>On generalizing,</p> <p>$T(i)= T(n-i) +i*3$</p> <p>When $i=n-1$, the resulting equation would be,</p> <p>$T(i)=T(1)+(n-1)*3$</p> <p>$=4+3n -3$</p> <p>$=3n +1$</p> <p>The solution of this recurrence equation is $3n+1$</p>	5	2	2	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
<p align="center">Part – C</p> <p align="center">(2 x 10 = 20 Marks)</p> <p>Instructions: Answer All the Questions</p>						
15. A	<p>Solve the following the recurrence relations using recursive tree method:</p> <p>a. $T(n)= T(n/3) +T(2n/3)+n$; $T(1)=1$.</p> <p>b. $T(n) = 8T(n/2) +1$; $T(1)=1$.</p> 	10	3	1	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2

						
OR						
15. B	<p>Prove the following relations:</p> <p>a. $T(n)=3n^3+2n^2+3$; $g(n)=n^3$; $T(n)$ is in $O(g(n))$</p> <p>Answer :</p> <p><i>Solution</i> The definition of the big-Oh notation is that $f(n) \leq c \times g(n)$. Therefore, one has to show that $3n^3 + 2n^2 + 3 \leq cn^3$ holds good for a positive number c and for sufficiently large values of n.</p> $ \begin{aligned} f(n) &= 3n^3 + 2n^2 + 3 \\ &\leq 3n^3 + 2n^3 + 3n^3 \quad (\text{as growth of functions } n^3 \text{ and } 3 \text{ is less than } n^3) \\ &\leq 8n^3 \end{aligned} $ <p>It can be observed that $c = 3 + 2 + 3 = 8$ (one can approximate $2n^2$ to 3 to $2n^3$ and $3n^3$ respectively). This condition holds good for any values of $c \geq 8$. Assuming $c = 8$, one can say that this algorithm is of the order of $O(n^3)$.</p> <p>One can generalize the result as follows: only the largest degree of the polynomial matters.</p> <p>Let the polynomial be $f(n) = \sum_{i=0}^m a_i n^i$, whose degree is m. Then one can show that $f(n) = O(n^m)$.</p> <p>The proof is given as follows:</p> $ \begin{aligned} f(n) &\leq a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0 \\ &\leq a_m n^m + a_{m-1} n^m + \dots + a_1 n^m + a_0 n^m \quad \text{for } n \geq 1 \\ &= \left(\sum_{i=0}^m a_i \right) n^m \\ &= c \times n^m \\ &= O(n^m) \end{aligned} $ <p>b. $n \in O(n^2)$</p>	10	3	1	$\frac{1-4 \cdot 10^1}{2}$	1.1, 1.3, 2.4, 12.2. 2

	Answer: The given statement implies that $n \leq c \cdot n^2$. This is true $n \geq n_0$, where $n_0 = 0$ and $c > 0$. Therefore $n \in O(n^2)$					
16. A	<p>Design an algorithm to arrange given integers in an array with running time complexity in $O(n \log n)$ and prove the same.</p> <p>Answer: mergesort(array) If len(array) > 1 Then # This is the point where the array is divided into two subarrays halfArray = len(array) / 2</p> <p>FirstHalf = array[:halfArray] # The first half of the data set</p> <p>SecondHalf = array[halfArray:] # The second half of the data set</p> <p># Sort the two halves mergeSort(FirstHalf) mergeSort(SecondHalf) merge(FirstHalf, SecondHalf)</p> <p>k = 0</p> <p>Merge: merge(FirstHalf, SecondHalf) # Begin swapping values While i < len(FirstHalf) and j < len(SecondHalf) If FirstHalf[i] < SecondHalf[j] Then array[k] = FirstHalf[i] i += 1 Else array[k] = SecondHalf[j] j += 1 k += 1 EndIf EndWhile EndIf</p>	10	3	2	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2
OR						
16. B	<p>Justify the statement with an illustrative example “selection of pivot in Quick sort algorithm affects the running time”</p> <p>Answer: The selection of the pivot in the Quick Sort algorithm indeed has a significant impact on its running time. Quick Sort works by partitioning an array around a chosen pivot element, such that all elements smaller than the pivot are placed before it, and all elements greater than the pivot are placed after it. Then, the algorithm recursively sorts the subarrays created by the partitioning process.</p> <p>Let's illustrate this with an example:</p>	10	3	2	1- 4,10,1 2	1.1, 1.3, 2.4, 12.2. 2

	<p>Consider an array of integers: [5, 9, 3, 7, 2, 8, 6].</p> <p>Suppose we choose the first element, 5, as the pivot. In this case, after the partitioning step, the array might become [3, 2, 5, 9, 7, 8, 6], with the pivot 5 ending up in its sorted position.</p> <p>Now, if the pivot is chosen poorly (for example, always picking the first element in the array), and the array is already sorted or nearly sorted, Quick Sort degrades into its worst-case time complexity of $O(n^2)$. This happens because the partitioning step doesn't effectively split the array into two roughly equal halves, leading to one subarray significantly larger than the other, causing inefficient recursive calls.</p> <p>Example 2: Consider an unsorted array: [8, 3, 1, 4, 2, 7, 6, 5]</p> <p>Choose a bad pivot: Let's pick the largest element, which is 8 in this case.</p> <p>Partition the array: All elements are less than 8, resulting in: [1, 2, 3, 4, 5, 6, 7] 8</p> <p>Recursively sort both sub-arrays: We only have one element on the left (8) and a sub-array of size $n-1$ on the right. This creates a highly unbalanced situation where one sub-array remains unsorted in each recursive call.</p> <p>This scenario continues, leading to $n-1$ comparisons for each element in the worst case, resulting in a time complexity of $O(n^2)$.</p> <p>However, if the pivot is chosen effectively, such as selecting the median or a randomly chosen element, Quick Sort achieves its average-case time complexity of $O(n \log n)$. This is because a good pivot choice ensures that the array is split into approximately equal halves, leading to balanced partitions and efficient sorting.</p>					
--	--	--	--	--	--	--