SRM Institute of Science and Technology

College of Engineering and Technology

SET C

School of Computing

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

Academic Year: 2023-24 (EVEN)

Test: CLA-T1 **Date: 19.02.2024**

Course Code & Title: 21CSC204J Design and Analysis of Algorithms **Duration:** 1 hr 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	Number of comparisons required to search an element x = 18 in the list A= [5, 44, 89, 22, 18, 9, 3, 15,8] using linear search a) 3 b) 1 c) 4 **d)** 5 **Ans: 5**	1	2	1	1,4	4.4.2

2	What is the time complexity of following code. Assume that n > 0
	int segment(int n) {
	If(n==1)
	return 1;
	else
	return (n+ segment(n-1)); }
	a) O(n)
	b) O(log n)
	c) $O(n^2)$
	d) O(n!)
	Ans: a
		1	4	1	2,3	2.8.2

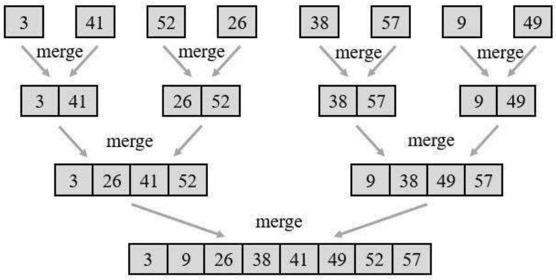
3	Which of the following functions provides the maximum asymptotic complexity? a) f1(n) = n^(3/2) b) f2(n) = n^(logn) c) f3(n) = nlogn **d)** f4(n) = 2^n. Ans: d.	1	2	1	2	2.8.2

4	Consider the following 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)
	  } }
	}
	a) $\Theta(n \sqrt{n})$
	b) $\Theta(n^2)$
	c) Θ (nlogn)
	d) $\Theta(n^2 \log n)$
	Ans: c
		1	2	1	4	4.6.2

5	Derive the recurrence relation for the following code:
	fact (int n)
	{
	  if(n= = 0)
	    return 1;
	  else
	   return fact(n-1)*n;
	}
	A. T(n) = T(n-1)+n^2
	B. T(n) = T(n)+n
	C. T(n) = T(n-1)+1
	D. T(n) = T(n-1)+n
	Ans: C
		1	1	1	1	1.2.1

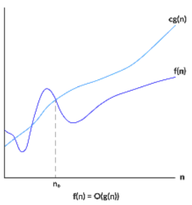
6	What is the worst case time complexity of a quick sort algorithm? a) O(N) b) O(N log N) c) $O(N^2)$ d) O(log N) Ans: C	1	1	2	2	2.8.2
7	Which of the following sorting algorithms provide the best time complexity in the worst-case scenario? a) Merge Sort b) Quick Sort c) Bubble Sort d) Selection Sort **Ans: a**	1	3	2	2,4	4.4.3
8	Solve the following recurrence using Master’s theorem. $T(n) = 4T(n/2) + n^2$ a) T(n) = O(n) b) T(n) = O(log n) c) $T(n) = O(n^2 \log n)$ d) $T(n) = O(n^2)$ Ans: c	1	2	2	1	1.2.1
9	Develop the algorithmic steps to find the maximum and minimum element in the given list.	1	3	2	2	2.5.2

	Apply Quick sort on a given sequence 7 11 14 6 9 4 3 12. What is the sequence after first phase, pivot is first element? a) 6 4 3 7 11 9 14 12 b) 6 3 4 7 9 14 11 12 c) 7 6 14 11 9 4 3 12 d) 7 6 4 3 9 14 11 12 Ans: b					
10	What is the time complexity of Largest subarray sum problem using naïve approach a) $T(n) = O(n)$ b) $T(n) = O(\log n)$ c) $T(n) = O(n^2 \log n)$ d) $T(n) = O(n^2)$ Ans: d	1	1	2	2	2.8.2
Part – B **(5 x 4 = 20 Marks)** **Instructions: Answer All the Questions**						
11	Express the function f(n) = n3/1000-100n2 - 100n + 3 in terms of Theta notation. $\mathbf{n^3/1000 - 100n^2 - 100n + 3 = \Theta(n^3)}$ Ans: For c1=1/2000, c2=1, f(n)=n3/1000 − 100n2 − 100n + 3 and g(n)= n^3) c1*g(n) ≤ f(n) ≤ c2*g(n) for large values of n (Or) The highest-order term in the function, which is n^3/1000. As n grows larger, the influence of the other terms in the function diminishes relative to n^3/1000. Therefore, for sufficiently large n, the function is dominated by the term n^3/1000. We can drop the lower-order terms and coefficients and write the function as Θ(n^3/1000).	5	2	1	1	1.2.1
12	State the objective of Strassen Matrix Multiplication and list the steps involved in the process Analyze the order of growth. (i).$F(n) = 2n^2 + 5$ and $g(n) = 7n$. Use the **Ω** (g(n)) notation Ans: f(n) = 2n2+5 and g(n) = 7n. We need to find the constant c such that f(n)≥ c∗ g(n). Let n = 0, then f(n) = 2n2+5 = 2(0)2+5 = 5 g(n) = 7(n) = 7(0) = 0 Here, f(n)>g(n)	5	3	1	1	1.2.1

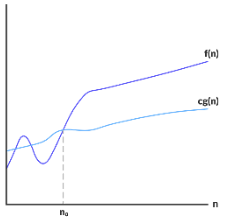
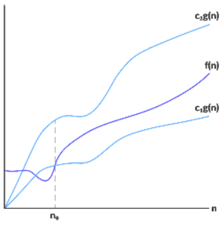
	Let n = 1, then f(n) = 2n2+5 = 2(1)2+5 = 7 g(n) = 7(n) = 7(1) = 7 Here, f(n)=g(n) Let n = 2, then f(n) = 2n2+5 = 2(2)2+5 = 13 g(n) = 7(n) = 7(2) = 14 Here, f(n)<g(n) Thus, for n=1, we get f(n) ≥ c∗ g(n). This concludes that Omega helps to determine the "lower bound" of the algorithm's run-time.					
13	Develop the algorithmic steps to find the maximum and minimum element in the given list. Illustrate the operation of merge sort on the array A = { 3 , 41, 52 , 26 , 38 , 57 , 9 , 49 } Ans: Have to explain the divide operation and then merge operation. (2 marks) Diagram (3 marks) 	5	2	2	4	4.5.1
14	i) Write the recurrence relation of Matrix Multiplication using divide and conquer approach and solve it to find time complexity (2 marks) ii) How Strassen Matrix Multiplication algorithms reduces time complexity of matrix multiplication (1 mark) iii) Write the recurrence relation of Strassen Matrix Multiplication and solve it to find its time complexity (2 marks) Ans: i) $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 . \end{cases}$ Solving above recurrence relation we get **T(n) : Θ (n^3)** **ii)** In Strassen Matrix Multiplication instead of performing eight recursive multiplications of n/2 X n/2 matrices, it performs only seven. The cost of eliminating one matrix multiplication will be several new additions of n/2 X n/2 matrices, but still only a constant number of additions. iii) $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 . \end{cases}$	5	4	2	4	4.6.2

<table>
<tr><td></td><td>Solving above recurrence relation we get

$T(n) : \Theta(n^{\lg 7})$ Since lg 7 lies between 2:80 and 2:81, Strassen's algorithm runs in $O(n^{2:81})$</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="7">Part – C
(2 x 10 = 20 Marks)
Instructions: Answer All the Questions</td></tr>
<tr><td>15. A</td><td>Find the time complexity of the below recurrence relation
i) $T(n) = \{2T(n-1)$ if $n > 0$
1 otherwise
ii) $T(n) = \{ 2T(n/2)+1$ if $n > 1$
1 otherwise
Ans:
i) $T(n) = 2T(n-1)$
$T(n) = 2[2T(n-2)] = 2^2T(n-2)$
$T(n) = 2[2^2T(n-2)] = 2^3T(n-3)$
$T(n) = 2^kT(n-k)$
$n-k = 0$, $n= k$, $T(0) = 1$
$T(n) = O(2^n)$

ii) $a = 2$, $b = 2$ and $f(n) = 1$.
So $c = \log_2 2 = 1$ and $O(n^1) > O(1)$,
which means that it fall in the third case and therefore a complexity is O(n).</td><td>10</td><td>1</td><td>1</td><td>1</td><td>1.2.1</td></tr>
<tr><td colspan="7">OR</td></tr>
<tr><td>15. B</td><td>Illustrate briefly on Big oh Notation, Omega Notation and Theta Notations. Depict the same graphically and explain
Big-O
The Asymptotic Notation O(n) represents the upper bound of an algorithm's running time
$O(g(n)) = \{f(n) \mid$ there exist positive constants c and n0, such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n0$. }


For example, if an algorithm has a time complexity of O(n), it means that the algorithm's running time will not exceed the linear growth rate, even if the input size increases
Big-Ω
The Omega Asymptotic Notation O(n) represents the lower bound of an algorithm's running time.
$\Omega(g(n)) = \{f(n) \mid$ there exist positive constants c and n0, such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n0$. }</td><td>10</td><td>1</td><td>1</td><td>1</td><td>1.6.1</td></tr>
</table>

	 f(n) = Ω(g(n)) For example, if an algorithm has a time complexity of Ω(n), it means that the algorithm's running time will not be less than the linear growth rate, even if the input size decreases **Big-Θ** The Theta Asymptotic Notation O(n) represents the both lower bound and upper bound of an algorithm's running time. $\Theta(g(n)) = \{f(n) \mid$ there exist positive constants c1, c2, and n0, such that $0 \leq c1g(n) \leq f(n) \leq c2g(n)$ for all $n \geq n0$  f(n) = Θ(g(n)) For example, if an algorithm has a time complexity of Θ(n), it means that the algorithm's running time will be proportional to the linear growth rate, even if the input size increases or decreases					
16. A	What is the closet pair problem? Explain the brute force approach to solve closest-pair with an example ii) Derive its time complexity. The closest-pair problem calls for finding the two closest points in a set of n points. It is the simplest of a variety of problems in computational geometry that deals with proximity of points in the plane or higher-dimensional spaces. We assume that the points in question are specified in a standard fashion by their (x, y) Cartesian coordinates and that the distance between two points pi(xi, yi) and pj (xj , yj) is the standard Euclidean distance $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$ The brute-force approach compute the distance between each pair of distinct points and find a pair with the smallest distance. we do not want to compute the distance between the same pair of points twice. To avoid doing so, we consider only the pairs of points (pi, pj) for which i < j . ALGORITHM BruteForceClosestPair(P) //Finds distance between two closest points in the plane by brute force	10	3	2	4	4.4.1

	//Input: A list P of n ($n \geq 2$) points p1(x1, y1), . . . , pn(xn, yn) //Output: The distance between the closest pair of points
	d ← ∞
	for i ← 1 to n − 1 do
	for j ← i + 1 to n do
	d ← min(d, sqrt((xi − xj)2 + (yi − yj)2)) //sqrt is square root
	return d
	The number of times it will be executed can be computed as follows:
	$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2 = 2 \sum_{i=1}^{n-1} (n-i)$
	$= 2[(n-1) + (n-2) + \cdots + 1] = (n-1)n \in \Theta(n^2).$					

OR						
16. B	Devise an algorithm for Quick sort and derive its time complexity. For the above algorithm find the time complexity if all the elements are arranged in ascending order. Illustrate with the help of recurrence tree. Quicksort is based on the three-step process of divide-and-conquer. • To sort the subarrayA[p . . r]: **Divide:** Partition A[p . . r], into two subarrays A[p . . q − 1] and A[q + 1 . . r], such that each element in the firstsubarray A[p . . q − 1] is ≤ A[q] and A[q] is ≤ each element in the second subarrayA[q + 1 . . r]. **Conquer:** Sort the two subarrays by recursive calls to QUICKSORT. **Combine:** No work is needed to combine the subarrays, because they are sorted in place. • Perform the divide step by a procedure PARTITION, which returns the index q that marks the position separating the subarrays. QUICKSORT (A, p, r) If p < r then q ←PARTITION(A, p, r) QUICKSORT (A, p, q − 1) QUICKSORT (A, q + 1, r) Initial call is QUICKSORT (A, 1, n) **Partitioning** Partition subarrayA [p . . . r] by the following procedure: PARTITION (A, p, r) x ← A[r] i ← p –1 for j ← p to r –1 } do if A[j] ≤ x then i ← i + 1 swap (A[i] ↔ A[j])	10	3	2	4	4.4.3

}
Swap(A[i + 1] ↔ A[r])
return i + 1

Complexity Analysis:

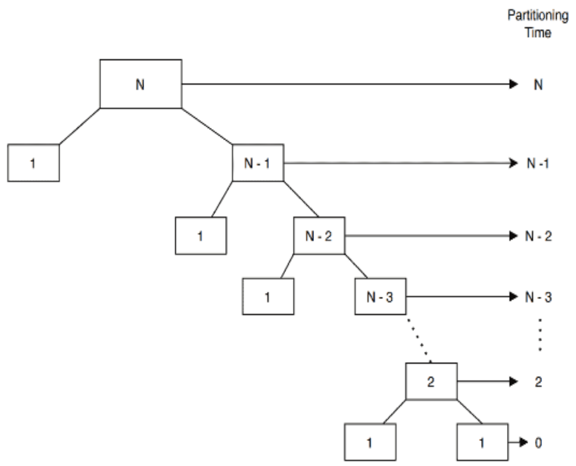
T(n) = O(nlogn)

if all the elements are arranged in ascending order Worst case occurs

If N is the length of array and having current pivot at starting position of array, pivot at last index is identified only traversing array from start to end . After fixing pivot and splitting resultant sub arrays of length are (N-1),1
Again this (N-1) sub array finds next pivot at last index resulting array partitions with lengths (N-2),1.

This process repeats until the final sub arrays lengths are both 1,1.

So, this can be represented using recursive tree as follows.



Therefore, the time complexity of the Quicksort algorithm in worst case is

$$[N + (N-1) + (N-2) + (N-3) + ... + 2] = [\frac{N(N+1)}{2} - 1] = \mathcal{O}(N^2)$$

Course Outcome (CO) and Bloom's Level (BL) Coverage in the Questions