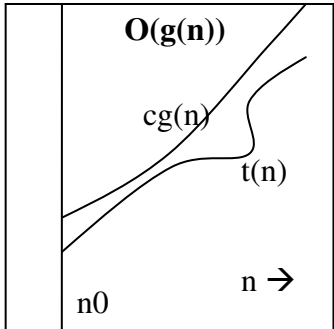
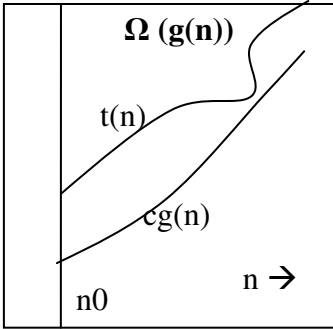
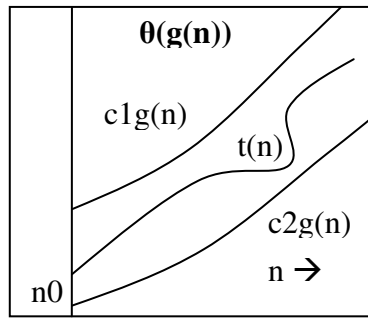


Sub :	Design and Analysis of Algorithms					Code:	15CS43		
Date :	28 / 03 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	4	Branch:	CSE/ISE
Answer Any FIVE FULL Questions									

		Mar	OBE	
		ks	CO	RB T
1	<p>(a) Explain the asymptotic notations for analysis of algorithms. Support your answer with proper graphs and examples.</p> <p>O-notation: A function $t(n)$ is said to be in $O(g(n))$, if $t(n)$ is bounded above by some positive constant multiple of $g(n)$ for all large n, i.e. $t(n) \leq c.g(n)$ for all $n \geq n_0$ e.g. $100n + 5 \in O(n^2)$</p> <p>Ω-notation: A function $t(n)$ is said to be in $\Omega(g(n))$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n, i.e. $t(n) \geq c.g(n)$ for all $n \geq n_0$ e.g. $100n^3 + 5 \in \Omega(n^2)$</p> <p>$\Theta$-notation: A function $t(n)$ is said to be in $\Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large n, i.e. $c_2g(n) \leq t(n) \leq c_1.g(n)$ for all $n \geq n_0$ e.g. $100n^2 + 5n \in \Theta(n^2)$</p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <p>$O(g(n))$</p>  </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <p>$\Omega(g(n))$</p>  </div> <div style="border: 1px solid black; padding: 5px; text-align: center;"> <p>$\theta(g(n))$</p>  </div> </div>	[5]	CO1	L1
	<p>(b) Prove that if $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then, $t_1(n) + t_2(n) \in O(\max\{O(g_1(n)), (g_2(n))\})$</p> <p>Proof: $t_1(n) \leq c_1g_1(n)$ for all $n \geq n_1$ $t_2(n) \leq c_2g_2(n)$ for all $n \geq n_2$ let $c_3 = \max(c_1, c_2)$, and consider $n \geq \max(n_1, n_2)$ Adding the above two, we get $t_1(n) + t_2(n) \leq c_1g_1(n) + c_2g_2(n)$ $t_1(n) + t_2(n) \leq c_3g_1(n) + c_3g_2(n) = c_3.2.\max(g_1(n), g_2(n))$ thus, $t_1(n) + t_2(n) \in O(\max\{O(g_1(n)), (g_2(n))\})$</p>	[5]	CO2	L3
2	<p>(a) Compare the orders of growth of the functions $(\log_2 n)$ and $(\sqrt[2]{n})$, and specify which function grows faster.</p>	[5]	CO1	L2

	$\lim_{n \rightarrow \infty} \left(\frac{\log_2 n}{\sqrt{n}} \right) = \lim_{n \rightarrow \infty} \left(\frac{(\log_2 n)'}{(\sqrt{n})'} \right) = \lim_{n \rightarrow \infty} \left(\frac{\log_2 e \frac{1}{n}}{1/2\sqrt{n}} \right)$ <p>Applying limits, we get 0. Since the limit = 0, log2 n has smaller growth than $\sqrt[2]{n}$</p>			
	<p>(b) Explain the mathematical analysis of Fibonacci recursive algorithm.</p> $\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2), n > 1$ $= 1, n=1$ $= 0, n=0$ <p>$A(n) = A(n-1) + A(n-2) + 1$, at step 1 (+1 is for the comparison to check if $n > 1$) This is re-written as, $[A(n) + 1] - [A(n-1) + 1] - [A(n-2) + 1] = 0$ Let $B(n) = A(n) + 1$ Then the above is re-written as $B(n) - B(n-1) - B(n-2) = 0$ $B(0) = 0, B(1) = 1$</p> <p>That is, $B(n) = F(n+1)$ And from the Fibonacci recurrence equation, $A(n) = B(n) - 1 = \frac{1}{\sqrt{5}} (\phi^{n+1} - \phi^{n-1}) - 1$ This is exponential in time.</p>	[5]	CO4	L4
3	<p>(a) What is an algorithm? Give the algorithm specifications. Define time complexity and space complexity.(1+2+1+1)</p> <p>An algorithm is a finite set of instructions that if followed accomplishes a particular task. Algorithm must satisfy the following criteria: Input: Zero or more inputs supplied Output: At least one quantity is produced Definiteness: Each instruction is clear and unambiguous. Finiteness: Algorithm terminates after finite number of steps. Effectiveness: Every instruction must be basic and feasible.</p> <p>Specification: Algorithm is specified as a pseudocode or program form. It can either be iteratively specified with loops, or recursively specified using recursive function calls.</p> <p>Time Complexity: Number of steps/computer time it takes to run the algorithm. The time T(P) taken by the program P is the sum of the compile time and run time.</p> <p>Space Complexity: Amount of memory/resources it needs to run the algorithm. There is a fixed part and a variable part.</p>	[5]	CO1	L1
	<p>(b) Design a recursive algorithm for solving tower of Hanoi problem and give the general plan of analyzing that algorithm. Show that the time complexity of tower of Hanoi algorithm is exponential in nature.</p> <pre> TowersOfHanoi(int n, tower x, tower y, tower z) { if (n) { TowersOfHanoi(n-1, x,z,y) Output "Move top disk from tower 'x' to top of tower 'y'" TowersOfHanoi(n-1, z, y,x) } } </pre> <p>$M(n) = 2 M(n-1) + 1, n > 1$ $M(1) = 1$ \</p> <p>Step 1</p>	[5]	CO5	L2

	$M(n) = 2 M(n-1) + 1$ <p>Step 2, $M(n) = 2^2 M(n-2) + 2 + 1$</p> <p>Step k, $M(n) = 2^k M(n-k) + 2^{k-1} + 2^{k-2} + \dots + 2^0$</p> <p>Last step is when $k = n-1$</p> $M(n) = 2^{n-1} M(1) + 2^{n-2} + 2^{n-3} + \dots + 2^0$ $= 2^n - 1, \text{ which is exponential in time.}$			
4	<p>(a) Is Quick sort a stable sort. Design the algorithm for quick Sort. Give the time complexity for best, average, and worst cases.</p> <p>No, quick sort is not a stable algorithm as it does not preserve the order of the input sequence.</p> <pre> algorithm quicksort(A, lo, hi) is if lo < hi then p := partition(A, lo, hi) quicksort(A, lo, p - 1) quicksort(A, p + 1, hi) algorithm partition(A, lo, hi) is pivot := A[hi] i := lo - 1 for j := lo to hi - 1 do if A[j] ≤ pivot then i := i + 1 swap A[i] with A[j] swap A[i+1] with A[hi] return i + 1 </pre>	[5]	CO3	L2
	<p>(b) Sort the following numbers using quick sort, 65, 70, 75, 80, 85, 60, 55, 50, 45</p> <p>Let array be called A[], and assume elements start from 0 and ends at 8.</p> <p>Choose 65, A[0] as pivot.</p> <p>i=0, j=9: 65, 70, 75, 80, 85, 60, 55, 50, 45;</p> <p>i=1, j=8: 65, 45, 75, 80, 85, 60, 55, 50, 70;</p> <p>i=2, j=7: 65, 45, 50, 80, 85, 60, 55, 75, 70;</p> <p>i=3, j=6: 65, 45, 50, 55, 85, 60, 80, 75, 70;</p> <p>i=4, j=5: 65, 45, 50, 55, 60, 85, 80, 75, 70;</p> <p>i=5, j=4: break loop</p> <p>Swap pivot with A[j]</p> <p>i=4, j=5: 60, 45, 50, 55, 65, 85, 80, 75, 70;</p> <p>A: 60, 45, 50, 55, 65, 85, 80, 75, 70;</p> <p>A11 Quicksort(60, 45, 50, 55) and A12 Quicksort(85, 80, 75, 70)</p> <p>A11: 60, 45, 50, 55</p> <p>55, 45, 50, 60 => A21 Quicksort(55, 45, 50)</p> <p>A12: 85, 80, 75, 70</p> <p>85, 80, 75, 70</p> <p>70, 80, 75, 85 => A22 Quicksort(70, 80, 75)</p> <p>A: 55, 45, 50, 60, 65, 70, 80, 75, 85</p> <p>A21: 55, 45, 50 => 50, 45, 55 => A31 Quicksort(50, 45) => 45, 50</p> <p>A22: 70, 80, 75 => A32 Quicksort(80, 75) => 75, 80</p>	[5]	CO5	L3

A: 45, 50, 55, 65, 70, 75, 80, 85

OR

(a) You are given a set of n elements. Your task is to design an algorithm to find the maximum and minimum element in the set. Use the divide & conquer approach. The algorithm should satisfy all the criteria of a good algorithm.

[10] CO4 L5

```

Void MaxMin(int i, int, j, Type& max, Type& min)
{
if (i==j) max=min=a[i];
else if (i=j-1)
    {
        if (a[i]<a[j]) {max=a[j]; min=a[i];}
        else {max=a[i]; min=a[j];}
    }
else
    {
        mid=(i+j)/2
        MaxMin(i,mid,max,min);
        MaxMin(i,mid+1,max1,min1);

        if(max<max1) max = max1;
        if(min>min1) min = min1;
    }
}

```

5 (a) Explain the concept of divide and conquer. Design the algorithm for merge Sort.

[5] CO4 L2

Given a problem of size n, the divide and conquer strategy suggests splitting the problem into k distinct subsets, $1 < k \leq n$, yielding k subproblems. These subproblems are in turn solved using further dividing them in sub-subproblems, and so on until the problem is small enough to be solved individually or in constant time. The steps are as follows:

Divide, Conquer and combine.

Merge sort:

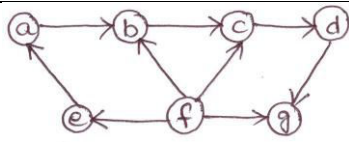
MERGE-SORT (A, p, r)

1.	IF $p < r$	//	Check	for	base	case
2.	THEN $q =$	FLOOR[(p + r)/2]	//	Divide	step	
3.	MERGE	(A, p, q)	//	Conquer	step.	
4.	MERGE	(A, q + 1, r)	//	Conquer	step.	
5.	MERGE (A, p, q, r)	//	Conquer step.			

MERGE (A, p, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays L[1 .. $n_1 + 1$] and R[1 .. $n_2 + 1$]
4. **FOR** $i \leftarrow 1$ **TO** n_1
5. **DO** L[i] \leftarrow A[p + i - 1]
6. **FOR** $j \leftarrow 1$ **TO** n_2
7. **DO** R[j] \leftarrow A[q + j]
8. L[$n_1 + 1$] \leftarrow ∞
9. R[$n_2 + 1$] \leftarrow ∞
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. **FOR** $k \leftarrow p$ **TO** r
13. **DO IF** L[i] \leq R[j]

	<p>14. THEN A[k] ← L[i] 15. i ← i + 1 16. ELSE A[k] ← R[j] 17. j ← j + 1</p>				
	<p>(b) Sort the list E, X, A, M, P, L, E in alphabetical order using merge sort.</p> <p>Divide: EXAMPLE EXAM PLE EX AM PL E E X A M P L E Conquer & Combine: EX AM LP E AEMX ELP AEELMPX</p>	[5]	CO2	L3	
6	<p>(a) Design a recursive algorithm for finding the minimum and maximum elements in a list. Give the recursive equation for time complexity.</p> <pre> Void MaxMin(int i, int, j, Type& max, Type& min) { if (i==j) max=min=a[i]; else if (i=j-1) { if (a[i]<a[j]) {max=a[j]; min=a[i];} else {max=a[i]; min=a[j];} } else { mid=(i+j)/2 MaxMin(i,mid,max,min); MaxMin(i,mid+1,max1,min1); if(max<max1) max = max1; if(min>min1) min = min1; } } </pre> <p>$T(n) = T(n/2) + T(n/2) + 2, n > 2$ $= 1, n = 2$ $= 0, n = 1$</p> <p>Solving, we get $T(n) = 3n/2 - 2$</p>	[5]	CO4	L2	
	<p>(b) Find the upper bound of recurrences given below by substitution method</p> <p>i) $T(n) = 2T(n/2) + n$ $a = 2, b = 2, f(n) = n$ $h(n) = f(n)/n^{(\log_b a)} = n/n = 1 = (\log n)^i, i = 0$ $u(n) = \theta(\log n)$ $T(n) = n^{(\log_b a)} [T(1) + \theta(\log n)] = \theta(n \log n)$</p> <p>ii) $T(n) = T(n/2) + 1$ $a = 1, b = 2, f(n) = 1$ $h(n) = f(n)/n^{(\log_b a)} = 1/1 = 1 = (\log n)^i, i = 0$ $u(n) = \theta(\log n)$ $T(n) = n^{(\log_b a)} [T(1) + \theta(\log n)] = \theta(\log n)$</p>	[5]	CO1	L2	
7	<p>(a) Obtain topological sorting for the given diagraph</p>	<p>Remove source nodes one by one.</p> <ol style="list-style-type: none"> 1. Remove f 2. Remove e 3. Remove a 4. Remove b 	[5]	CO5	L3



5. Remove c
 6. Remove d
 7. Remove g
- Topological sorted order: **f, e, a, b, c, d, g**

	<p>(b) Analyze and compare the time complexities of matrix multiplication using (i) conventional divide-and conquer method, versus (ii) Strassen's method. Which method is better? Justify your answer.</p> <p>(i) Conventional method: $T(n) = b, n \leq 2$ $= 8T(n/2) + cn^2, n > 2, b, c \text{ are constants}$ Solving, we get $h(n) = c/n$ $u(n) = O(1)$ $T(n) = O(n^3)$</p> <p>(ii) Strassen's method: $T(n) = b, n \leq 2$ $= 7T(n/2) + an^2, n > 2, b, a \text{ are constants}$ $h(n) = a/n^{(0.81)}$ $u(n) = O(1)$ $T(n) = O(n^{(2.81)})$</p> <p>Comparing the time complexities, we see that Strassen's method has a lower time complexity than the conventional method, and hence better.</p>	[5]	CO4	L4
8	<p>(a) Consider a modification of merge sort in which the array is recursively divided into two halves, up to a point till the sub-arrays reach size k (where k is a constant, and $1 < k < n$). These sub-arrays of size k are individually sorted in time $\theta(k^2)$, and then merged using the standard merging mechanism. Illustrate this modified algorithm, and analyze its time complexity.</p> <p>At the lowest level we have n/k sub-arrays each of size k. They are sorted in $\theta(k^2)$ time. These n/k sub arrays at that level are sorted in $(n/k)\theta(k^2) = \theta(nk)$ time in the worst case. They are then merged using the standard merge procedure. Thus the complexity to merge from level n/k is $\theta(n \log (n/k))$ in worst case.</p> <p>Thus the modified algorithm runs in $\theta(nk + n \log (n/k))$ in worst case.</p>	[5]	CO5	L4
	<p>(b) Solve the following recurrence relations</p> <p>i) $f(n) = f(n-1) + n, n > 0$ $= 0, n = 0$ $f(n) = f(n-1) + n = f(n-2) + n-1 + n$ step k, $f(n) = f(n-k) + kn - (1+2+\dots + k-1)$ last step, $k=n$ $f(n) = f(0) + n^2 - (1+2+\dots + n-1)$ solving, we get $f(n) = n(n+1)/2 = O(n^2)$</p> <p>ii) $x(n) = 3x(n-1)$ for $n > 1, x(1) = 4$ At step k, $x(n) = 3^k x(n-k)$ Last step, $k=n-1$ $x(n) = 3^{(n-1)} \cdot 4 = O(3^n)$</p> <p>iii) $x(n) = x(n/2) + n$, for $n > 1, x(1) = 1, n = 2^k$ Step i, $x(n) = x(n/2^i) + n/2^{(i-1)} + \dots + n/2^0$ Putting $n = 2^k$ Step i, $x(n) = x(2^{(k-i)}) + 2^{(k-(i-1))} + \dots + 2^{(k-0)}$</p> <p>Last step, $i=k-1$ $x(n) = 1 + (2^2 + 2^3 + \dots + 2^k) = 2n-1 = O(n)$</p>	[5]	CO5	L3

OBE- Outcome Based Education
 RBT- Revised Blooms Taxonomy
 CO - Course Outcome
 PO- Program Outcome
 L - Cognitive Level

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Ability to understand the fundamental principles of algorithm design to solve a computational problem.	2	1	2	1	1	1	0	0	2	0	1	1
CO2:	Ability to analyse algorithms to determine the correctness and efficiency class	1	2	2	2	0	1	0	0	2	0	1	1
CO3:	Ability to analyse algorithms at their worst-case, average-case, and best-case behaviours	1	2	2	2	0	1	0	0	2	0	1	1
CO4:	Ability to devise algorithms using different design techniques (brute-force, divide and conquer, greedy, decrease and conquer, dynamic programming and back tracking), and compare their efficiencies	1	2	2	2	0	1	0	0	2	1	2	1
CO5:	Ability to apply and implement learned algorithm design techniques and data structures to solve problems	1	2	2	2	2	1	0	0	2	0	1	1
CO6:	Ability to design and implement new algorithms to solve real world problems	1	2	2	2	2	1	0	0	2	0	1	2

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 - *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*