K.S. INSTITUTE OF TECHNOLOGY, BANGALORE - 560109

Ist SESSIONAL TEST QUESTION PAPER 2018–19 Even SEMESTER

USN										

Set A

Degree	:	**B.E**	**Semester**	:	**IV**
Branch	:	**Computer Science & Engineering**	**Subject Code**	:	**17CS43**
Subject Title	:	**Design and Analysis of Algorithms**	**Date**	:	**2019-03-12**
Duration	:	**90 Minutes**	**Max Marks**	:	**30**

Note: Answer ONE full question from each part.

Q No.	Question	Marks
1(a)	**Write** an algorithm using iteration to output all prime factors of a given positive integer N.	**5**

Sch & Ans	**Sch:** 2 marks for defining iteration loops, 3 marks for correct algo
	Ans
	for factor = 2 to sqrt(n), do
	 while remainder(n, factor) eq 0, do
	  print factor
	  replace n by n/factor
	 done //while
	done // for
	if n > 1, then
	 print n
	fi	

(b)	**Discuss** an algorithm using recursion to output all prime factors of a given positive integer N.	**5**

Sch & Ans	Sch: 2 marks for defining iteration loops, 3 marks for correct algo
	Ans
	function primefactor(n)
	 for factor=2 to n do
	  if remainder(n, factor) eq 0 then
	   print factor
	   prime(n / factor)
	   break
	  fi
	 //end for
	 return
	
	#invoke the function
	primefactor(n)	

(c)	**Evaluate** the performance of above two algorithms w.r.t. time computation and memory requirements.	**5**
Sch & Ans	Sch: 2 marks each for defining time complexity of each algo, 1 marks for comparing Ans Algo with iteration will take n computations for remainder function for a primer	

	number and thus worst case performance is O(n). Algo with recursion will again take n computations of remainder and thus its worst case performance will also be O(n). Further, 2nd algorithm will use extra stack space for non-prime numbers, and the stacks space will be equal to number of prime factors. So, in that sense algo with recursion takes more resources.	
2(a)	**Write** an algorithm using recursion to compute Binomial coefficients ^nC_k = `n!/(k!*(n-k)!)`	**5**

Sch & Ans	Sch: 2 marks for defining mathematical expression and 2 marks for defining terminating condition and 1 mark remaining algorithm
	Ans
	The mathematical expression is
	$^nC_k = {}^{n-1}C_k + {}^{n-1}C_{k-1}$
	
	`algo` binomial(n, k)
	 `if` k `eq` 0 `or` k `eq` 1 `then`
	  `return` 1
	 `fi`
	 `return` (binomial(n-1, k) + binomial(n-1, k-1))	

(b)	**Outline** an algorithm to compute a polynomial using Horner's rule $P_n(x) = a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0.$	**5**

Sch & Ans	Sch: 2 marks for defining mathematical expression and 2 marks for defining terminating condition and 1 mark remaining algorithm
	Ans
	Mathematical computation for Horner's rule to compute in O(n) time is given as follows
	$P_n(x) = a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0.$
	$= a_0+x(a_1+ x(a_2 +x(\dots. + xa_n)\dots)$
	i.e.
	$P_n(x) = a_0+x\ P_{n-1}(x)$
	Assuming that all coefficients are given in an arrays arr[] i.e. a_0=arr[0], a_1=arr[1], …, a_n=arr[n], the algorithm will be as below
	
	`Algo` horner(x, arr, index, n):
	 `if` (index == n) `then`
	  `return` arr[n]
	 `else`
	  `return` (arr[index] + x * horner(x, arr, index+1, n))
	 `fi`
	
	#Invocation of algorithm
	horner(x, arr, 0, len(arr) -1)
		x

(c)	**Construct** the recurrence equation for the computation of Q2(b) and solve the same.	**5**

Sch & Ans	Sch: 2 marks for defining recurrence relation and 3 marks for solving it
	Ans
	Recurrence equation for Horner's rule is
	T(n) = T(n-1) + 1
	    = T(n-2) + 1 + 1
	    = T(1)+ 1+ …(n-1 times) + 1
	    = O(n)	

3(a)	**Outline** an algorithm to compute sum of N numbers given in an array using divide and conquer technique by dividing the input into two (approximately) equal parts.	**5**

Sch & Ans	Sch: 2 marks for dividing the array in equal parts, and 3 marks for conquering it
	Ans
	`Algo` computesum(L, R, arr)
	  `if` (L==R) `then`
	    `return` arr[L]
	  mid = (L + R)/2
	  `return` (computesum(L, mid, arr) + computesum(mid+1, R, arr))
	#invocation
	`print` computesum(1,N, arr)	

(b)	**Show** the recurrence equation for the computation of Q3(a) and solve the same.	**5**
Sch & Ans	Sch: 2 marks for writing recurrence relation and 3 marks for solving it. Ans $T(1) = 0$ $T(n) = 2T(n/2) + 1$, and $= 2[2T(n/2^2)+1] + 1 = 2^2T(n/2^2) + 2^1 + 2^0$ $=2^kT(n/2^k) + 2^{k-1}+\ldots+ 2^1 + 2^0$ $=2^{k-1}+ \ldots+ 2^1 + 2^0$ $= 2^k-1 = n = O(n)$	
(c)	**Apply** the algorithm in Q3(a) to find the sum of following numbers `11, 27, 18, 14, 25, 31, 29, 15.` Show the results at each step of the computation.	**5**
Sch & Ans	Sch: 1 mark each for splitting and combining Ans Step 1: left arr: 11, 27, 18, 14; right array = 25, 31, 29, 15 Step 2: left arr: 11, 27; right array = 18, 14 Step 3: left arr: 11, right array = 27 Recursion terminates Step 4: Returns 11+27 = 38. Step 5: left array: 18, right array 14 Recursion terminates Step 6: returns 18+14=32 Recursion terminates Step 7: returns 38 + 32 = 70 Process continues like this.	

4(a)	**Compare** the order of growth of following functions: $f(n) = n(n+1)(2n+1)/6,\ g(n) = n^3$	**5**
Sch & Ans	Sch: 2 marks for defining the limit formula and 3 marks for computing the limits Ans $\lim_{n\to\infty} f(n)/g(n)$ $= \lim_{n\to\infty} (n(n+1)(2n+1)/6)\ /n^3$	

	$= \text{Lim}_{n\to\infty} (2n^3 + 3n^2 + n)/6n^3$ $= \text{Lim}_{n\to\infty} (1/3 + 1/2n + 1/6n^2)$ = 1/3 i.e. a constant. Thus, both f(n) and g(n) are of same order i.e. f(n) = O(g(n)) and g(n) = O(f(n))	
(b)	**Explain** Big-Oh, Big-Theta and Big-Sigma notations and provide one example of each	**5**
Sch & Ans	Sch: 2 marks for defining the notations and 3 marks for the example Ans Big-Oh defines the upper limit, and formally specified as A function t(n) is said to be in O(g(n)) if t(n) is bounded above by some +ve constant multiple of g(n) for large n, i.e. t(n) ∈ O(g(n)), if $t(n) \leq cg(n)$ for all $n \geq n_0$ Big-Sigma defines the lower limit and formally specified as A function t(n) is said to be in Ω(g(n)) if t(n) is bounded below by some +ve constant multiple of g(n) for large n, i.e. t(n) ∈ Ω(g(n)), if $t(n) \geq cg(n)$ for all $n \geq n_0$ Big-Theta defines the similar order of growth A function t(n) is said to be in Θ(g(n)) if t(n) is bounded both above and below by some +ve constant multiple of g(n) for all large n, i.e. t(n) ∈ Θ(g(n)), if $c_2g(n) \leq t(n) \leq c_1g(n)$ for all $n \geq n_0$	
(c)	**Develop** an algorithm to sort 4 numbers a, b, c, d using max of 5 comparisons.	**5**
Sch & Ans	Sch: 1 marks for showing the approaching of taking 4!=24 possibilities and 4 marks for dividing the solution at each of division Ans The 24 possible orders for 4 numbers are abcd abdc acbd acdb adbc adcb bacd badc bcad bcda bdac bdca cabd cadb cbad cbda cdab cdba dabc dacb dbac dbca dcab dcba Only one of these will be in sorted order. Choose a strategy so that each time the set is divided into half. Comparison 1 compare a with b, say a<b, (other swap a, b) then set becomes abcd abdc acbd acdb adbc adcb cabd cadb cdab dabc dacb dcab Comparison 2: Choose next comparison such that this set gets divided into half Compare c with d, assume c<d (if not swap, c and d) and the set becomes abcd acbd acdb cabd cadb cdab Comparison 3 : Choose next comparison such that set again is divided into half. If we consider b and c, and let us say b<c, then set is subdivided in size of 1 and 5 and thus this comparison is not correct. So consider a and c, let us say a <c. the set then become abcd acbd acdb	**a**

	Comparison 4: Choose a comparison such that set divided into two and one. Let us compare a with d and assume a < d, then this does not divide the set. So let us say compare b and d. Then the set become `abcd   acbd` Comparison 5: Compare b and c now thus we will get Either `abcd` or `acbd`.	

Signature of the Faculty **Signature of the Module Coordinator** **Signature of the HOD**

Signature of the Chief Academic Coordinator **Signature of the Principal**