K.S. INSTITUTE OF TECHNOLOGY, BANGALORE - 560109

Ist SESSIONAL TEST QUESTION PAPER 2019–20 Even SEMESTER

USN										

Set A

Degree	**:**	**B.E**	**Semester**	**:**	**IV**
Branch	**:**	**Computer Science & Engineering**	**Subject Code**	**:**	**18CS42**
Subject Title	**:**	**Design and Analysis of Algorithms**	**Date**	**:**	**2020-03-12**
Duration	**:**	**90 Minutes**	**Max Marks**	**:**	**30**

Note:

1. Answer ONE full question from each part.
2. This is an open book exam. You can refer to any book, notes etc.
3. Sharing of notes, printed material etc. is not permitted.
4. Use of any electronic gadget e.g. phone, calculator, tablets, laptops etc. is prohibited.

Q No.	Question	Marks	K Level	CO mapping
	PART-A			
1(a)	**Sketch** an algorithm using recursion to output all prime factors of a given positive integer `N`. For example, if `N` is `24`, it should output `2, 2, 2, 3.`	**5**	**K-2 Under-standing**	**CO1**
(b)	Let `A[1],A[2],…,A[N]` be an array of `N` distinct positive integers. A pair (`A[i],A[j]`) is said to be an inversion if these numbers are out of order i.e. `i<j` but `A[i]>A[j]`. **Design** a brute-force algorithm that output all of inversions in the array. Hint: For each pair of (`i,j`), print the inversions. For example, for the array `[1,4,3,2,5]`, the inversions are `(4,3), (4,2), (3,2).`	**5**	**K-3 Applying**	**CO1**

(c)	Consider the following code segment, where X corresponds to last digit of your USN (For example, if your USN is `1KS18CS004`, then `X=4`).
	`int p = 200 + X;`
	`int q = 50;`
	`main(){`
	`  while (p >= q){`
	`  p = p-2;`
	`  q = q+1;`
	`  }`
	`}`
	Analyze the above code, and **identify** how many times the comparison `p>=q` is performed. Explain your answer.
		5	**K-3 Applying**	**CO2**

	OR			
2(a)	**Sketch** an algorithm using recursion to convert a given positive integer `N` into its octal notation. For example, if `N` is `67`, then it should output 103.	**5**	**K-2 Under-standing**	**CO1**
(b)	We are trying to determine the worst-case time complexity of a library function that is provided to us, whose code we cannot	**5**	**K-3 Applying**	**CO1**

	read. We test the function by feeding random inputs of different sizes. We find that a) for inputs of size 100, the function always returns well within one second, for inputs of size 1000, it mostly takes up to 2 second and for inputs of size 10,000 it sometimes takes up to 3 seconds. What is a reasonable conclusion we can draw about **determining** the worst-case time complexity of the library function for the input size N? Explain your reasoning.			

(c)	Consider the following algorithm to perform a mathematical operation on two input positive integers
	`function mathfn(X, Y) {`
	`  if y equal 1 {`
	`  return X`
	`  } else {`
	`  if odd(y) {`
	`  return X + mathfn(double(X), half(Y))`
	`  } else {`
	`  return mathfn(double(X), half(Y))`
	`  }`
	`  }`
	`}`
	Identify the mathematical operation performed by `mathfn` (e.g. exponentiation, logarithm, square, square root, multiply, divide etc.) and evaluate the time complexity of this operation. The function `odd(a)` return `True` if `a` is odd number else returns `False`. The function `double(a)` return `2*a`, and function `half(a)` returns integer division of `a` by 2, e.g. `half(7)` returns `3`. Assume that all three functions `odd()`, `double()`, and `half()` take `O(1)` time.
		5	**K-3 Applying**	**CO2**

	PART-B			
3(a)	Consider a variation of Hanoi's tower problem of moving `N` discs from Tower T1 to Tower T2 using two additional towers T3 and T4. Your friend employs following algorithm. S/he takes top `N/2` discs and moves them from T1 to T3 making use of T4 using the traditional Hanoi's tower approach of working with 3 towers. S/he then moves remaining bottom half of `N/2` discs from T1 again using the traditional Hanoi's tower approach. **Develop** on it **construct** the entire algorithm that moves all the `N` discs. **Evaluate** time complexity of this approach. For simplicity, you can assume that N=2**K, where K is a positive integer.	**5**	**K-3 Applying**	**CO1**
(b)	**Solve** the following recurrence relation using backward substitution method to compute the expression for `T(n)` i. `T(n) = 3T(n/2) + O(n)` ii. `T(n) = 6T(n/2) + O(n²)`.	**5**	**K-3 Applying**	**CO1**
(c)	**Demonstrate** the use of In-place Mergesort algorithm to sort the list 'A', 'L', 'G', 'O', 'R', 'I', 'T', 'H', 'M', 'S' in alphabetical order. The In-place algorithm implies that no extra array is to be used during the Merge operation. **Illustrate** the results at each step of the In-place Mergesort algorithm	**5**	**K-2 Under-standing**	**CO2**

	OR			
4(a)	**Construct** an algorithm to compute 3**N for some positive integer N using divide and conquer approach to divide it into two sub problems of about equal size. Identify the base case to solve the small enough problem (i.e. when N=0, and N=1). Evaluate the time complexity of this algorithm. Hint: $3^n = 3^{n/2} * 3^{n/2}$ when n even, and $3^n = 3 * 3^{(n-1)/2} * 3^{(n-1)/2}$ when n odd	**5**	**K-3 Applying**	**CO1**
(b)	Consider the master theorem as given below $T(n)=aT(n/b)+\Theta(n^d)$ for $n=b^k$, $k=1,2$, $T(1) = c$, where, $a \geq 1$, $b \geq 2$, $c>0$ $T(n) \in \begin{cases} \Theta(n^d) & if\ a < b^d \\ \Theta(n^d log n) & if\ a = b^d \\ \Theta(n^{log_b a}) & if\ a > b^d \end{cases}$ **Apply** this master theorem to solve following recurrence relations i. $T(n) = 4T(n/3) + O(n^2)$ ii. $T(n) = 9T(n/3) + O(n^{1.5})$ iii. $T(n) = 4T(n/2) + O(n^2)$	**5**	**K-3 Applying**	**CO1**
(c)	Let $A[1],A[2],\ldots,A[N]$ be an array of N distinct positive integers arranged in ascending order i.e. $A[i]<A[j]$ if $i<j$. **Sketch** an algorithm using divide and conquer approach to find closest pair of integers i.e. find j such that $A[j+1]-A[j]$ is smallest for all $0<=j<=n-1$. Hint: Conquer step in the algorithm would also involve comparing $A[(n/2)+1]-A[n/2]$ with left half and right half of the sub problem	**5**	**K-2 Under-standing**	**CO2**