K.S. INSTITUTE OF TECHNOLOGY, BANGALORE - 560109

I[st] SESSIONAL TEST Sch & Ans 2019–20 Even SEMESTER

Scheme and Answers

Set A

USN										

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
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**

Sch & Ans	Sch: 3 marks for correct approach, 2 marks for algo
	
	Ans
	`primefactor(n,k)`
	`  if n == k`
	`    print(n)`
	`    Return`
	`  if n%k == 0`
	`    print(k)`
	`    return primefactor(n/k, k)`
	`  else`
	`    return primefactor(n, k+1)`
	
	`#invoke as`
	`primefactor(n,2)`	

1(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**

Sch & Ans	Sch: 2 marks for the two nested for loops, 3 marks for correct code
	
	Ans
	`For I=1 to N-1`
	`  For j = i+1 to N`
	`    If A[i] > A[j]`
	`      Print((A[i], A[j])`	

(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`).
	<pre>int p = 200 + X;
	int q = 50;
	main(){
	  while (p >= q){
	    p = p-2;
	    q = q+1;
	  }
	}</pre>**Analyze** the above code, and **identify** how many times the comparison `p>=q` is performed. Explain your answer.
		5

Sch & Ans	Sch: 2 marks for right answer, 3 marks for analysis
	
	Ans
	`Consider X=0.`
	`Thus, initially, p=200, q=50`
	`Loop succeeds first time (200>=50)`
	`It will succeed next 50 times as well (each loop gap between p and q decreases by 3).`
	`After 50 iterations, p becomes 100, and q becomes 100. The 51st iteration succeeds since p=100, q=100.`
	`After 51st iteration, p becomes 98, and q becomes 101, and the comparison fails (but comparison is done).`
	`Thus, comparison is performed 52 times. Essentially, number of comparison = (p-q)/3+2.`	

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**

Sch & Ans	Sch: 2 marks for defining termination condition of recursion, 3 marks for invoking recusion.
	
	Ans
	<pre>Octal(N):
	  If N < 8
	    Return str(N)
	  Else
	    Return octal(N/8) + str(octal % 8)</pre>	

(b)	We are trying to determine the worst-case time complexity of a library function that is provided to us, whose code we cannot 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.	**5**

Sch & Ans	`Sch: 3 marks for reasoning, 2 marks for correct answer`
	
	`Ans`
	`Size of 100 (=10`^{2}`), time: 1s (2-1)`
	`Size of 1000 (=10`^{3}`), time: 2s (3-1)`
	`Size of 10000 (=10`^{4}`), time: 3s (4-1)`
	
	`Thus, the time complexity is O(log`$_{10}$`N -1) = O(Log1`$_0$`N)`	

(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

Sch & Ans	Sch: 2 marks for analysis/reasoning and 2 marks for right answer, 1 mark for complexity analysis
	
	Ans
	`X is doubled and Y is halved. Only those values of X are added which correspond to value of 1 for corresponding bit value of Y in its binary representation. Thus, the above achieves Multiply operation. (The algorithm is known as La-Russe algorithm)`
	
	`Time Complexity: O(log Y)`	

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**

Sch & Ans	Sch: 2 marks for writing the proper recurrence equation, and 3 marks for solving it. Ans $T_4(n) = T_3(n/2) + T_3(n/2) + T_3(n/2) = 3T_3(n/2)$ Where $T_4(n)$ represents Hanoi's tower using 4 towers, and $T_3(n)$ represents Hanoi's tower using 3 towers. The solution of $T_3(n)=2^n-1$. Thus, the answer to above problem is $3(2^{n/2}-1)$.	
(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. $T(n) = 6T(n/2) + O(n^2)$.	**5**
Sch & Ans	Sch: 2.5 marks for solving each. Ans i. $T(n) = 3T(n/2) + O(n)$ $=3[3T(n/2^2)+n/2)+n=3^2T(n/2^2)+3(n/2)+n$ … $=3^kT(n/2^k)+3^{k-1}(n/2^{k-1})+3^{k-2}(n/2^{k-2})+\ldots+ 3^1(n/2^1)+3^0(n/2^0)$ $= 3^kT(1)+n[(3/2)^{k-1}+[(3/2)^{k-1}++[(3/2)^{k-2}+\ldots]$ $= O(3^k) = O(3^{\log_2 n}) = O(n^{\log_2 3})$ Same way, $T(n) = 6T(n/2) + O(n^2)$. Would give the answer $O(n^{\log_2 6})$	
(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**
Sch & Ans	Sch: 3 marks for right technique, 2 marks for correct answer Ans ALGORITHMS ALGOR ITHMS AL GOR IT HMS A L G OR I T H MS(split till get to size 1) AL G O R IT H M S(merge arrays of size 1) AL G OR IT H MS AL GOR IT HMS AG LOR HT IMS (in place for I) AGLOR HIMST (in place for T) AGHOR ILMST (In place L) AGHIR LMOST (In place for O) AGHIL MORST (In place for R) AGHILMORST (everything in place)	
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	**5**

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

Sch & Ans

Sch: 2 marks for terminating recursion condition, 2 marks for recursion invocation using divide and conquer, 1 mark for complexity analysis

Ans

```
power(3,n)
  if n==0
    return 1
  elif n==1
    return 3
  elif n is even
    X = power(3, n/2)
    return X*X #also return power(3,n/2)*power(3,n/2)
  else
    X = power(3, (n-1)/2)
    return 3*X*X #return 3*power(3,(n-1)/2)*power(3,(n-1)/2)
Complexity: T(n) = T(n/2)+2 = O(log n)
For alternate case (given as comment)
   T(n) = 2T(n/2)+O(1) = O(n)
```

T(

(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, $a\geq 1$, $b\geq 2$, $c>0$

$$T(n) \in \begin{cases} \Theta(n^d) & if\ a < b^d \\ \Theta(n^d logn) & if\ a = b^d \\ \Theta(n^{log_b a}) & if\ a > b^d \end{cases}$$

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

Sch & Ans

Sch: 2 marks each for first 2 and 1 mark for 3rd equation

Ans

Eqn 1: a=4, b=3, n=2, thus $a<b^2$, thus 1st part of theorem

Ans : $\Theta(n^d) = \Theta(n^d) = \Theta(n^2)$

Eqn 2: a=9,b=3,n=1.5, thus $a>b^{1.5}$, use 3rd part of theorem

Ans : $\Theta(n^{log_a b}?) = \Theta(n^{1log_3 9}) = \Theta(n^2)$

Eqn 3: a=4, b=2, n=2, thus $a=b^2$, thus 2nd part of theorem

Ans : $\Theta(n^d log\ n) = \Theta(n^2 log\ n)$

(c)

Let A[1],A[2],…,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

Sch & Ans	Sch: 1 mark for dividing, 3 mark conquering, and 1 marks for solving smallest size
	
	Ans
	`closest(A,low,high)`
	`  # solving the smallest size problem`
	`  if high=low+1`
	`    return low`
	`  elif high = low + 2`
	`    X=Abs(A[low+1]-A[low])`
	`    Y=Abs(A[high] - A[low+1])`
	`    if X<Y`
	`      return low`
	`    else`
	`      return low+1`
	`   # dividing and combing part`
	`   X = closest(A, low, (low+high)/2)`
	`   Y = closest(A, (low+high)/2 +1, high)`
	`   Z = Abs(A[(low+high)/2+1] - A[(low+high)/2])`
	`   if A[X] < Z`
	`     if A[X] < A[Y]`
	`       return X`
	`     else`
	`       return Y`
	`   else`
	`     if Z < A[Y]`
	`       return (low+high)/2`
	`     else`
	`       return Y`
	`#invocation`
	` X=closest(A,1,N)`
	` print(A[X], A[X+1])#  the closest pairs are A[X], A[X+1]`	