![PES University logo]	**PES University, Bengaluru** (Established under Karnataka Act No. 16 of 2013)	**UE19CS251**

MARCH 2021: IN SEMESTER ASSESSMENT B Tech IV SEMESTER
TEST – 1

UE19CS251 – Design and Analysis of Algorithms
Scheme and Solutions

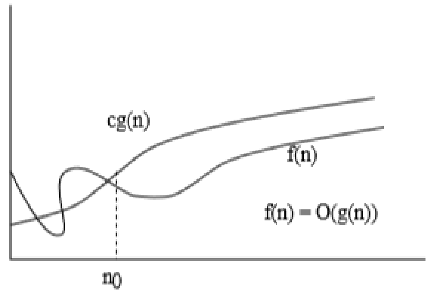
Time: 2 Hrs	Answer All Questions	Max Marks: 60

1. a) Provide formal definition with suitable graphs for Big-O and Big-Theta notations — **4**

Solution: 2M 2M

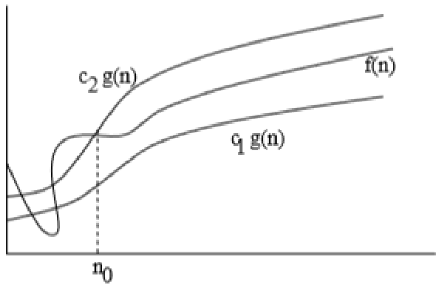
Big-O Notation

- **Definition:** $f(n) = O(g(n))$ iff there are two positive constants c and n_0 such that
  $|f(n)| \leq c\,|g(n)|$ for all $n \geq n_0$
- If $f(n)$ is nonnegative, we can simplify the last condition to
  $0 \leq f(n) \leq c\,g(n)$ for all $n \geq n_0$
- We say that "$f(n)$ is big-O of $g(n)$."
- As n increases, $f(n)$ grows no faster than $g(n)$. In other words, $g(n)$ is an *asymptotic upper bound* on $f(n)$.



Θ notation

- **Definition:** $f(n) = \Theta(g(n))$ iff there are three positive constants c_1, c_2 and n_0 such that
  $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$ for all $n \geq n_0$
- If $f(n)$ is nonnegative, we can simplify the last condition to
  $0 \leq c_1\,g(n) \leq f(n) \leq c_2\,g(n)$ for all $n \geq n_0$
- We say that "$f(n)$ is theta of $g(n)$."
- As n increases, $f(n)$ grows at the same rate as $g(n)$. In other words, $g(n)$ is an *asymptotically tight bound* on $f(n)$.



b) Using limits, compare the order of the growth of $\mathbf{(\log_2 n)^2}$ and $\mathbf{\log_3 n^2}$ — **4**

Solution:

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n), \\ c & \text{implies that } t(n) \text{ has the same order of growth as } g(n), \\ \infty & \text{implies that } t(n) \text{ has a larger order of growth than } g(n). \end{cases}$$ 1M

$\lim_{n\to\infty} \mathbf{(\log_2 n)^2 / \log_3 n^2} = \infty$ 2M

$(\log_2 n)^2 = \Omega(\mathbf{\log_3 n^2})$
$(\log_2 n)^2$ has faster order of growth than $(\mathbf{\log_3 n^2})$ 1M

c) Sort the given functions in increasing order of time complexity(Big O) — **2**

t1(n) = $\mathbf{n^{0.98787} \log n}$
t2(n) = **100n**
t3(n) = $\mathbf{1.01^n}$
t4(n) = $\mathbf{n^2}$

		Solution: t1(n),t2(n), t4(n), t3(n) 0.5M each t1(n) = n $^{0.98787}$ *log n = O(n $^{0.98787}$ *n $^{0.01213}$) = O(n) = O(t2(n))	

2.	a)	Algorithm Mystery(A[0….n-1,0…..n-1])
		//Input: A matrix A[0….n-1,0…..n-1] of real numbers
		for i ← 0 to n-1 do
		   for j ← i+1 to n-1 do
		      if(A[i,j]!=0)
		         return false
		return true
		(i) What does this algorithm compute?
		(ii) What is the basic operation?
		(iii) How many times the basic operation is executed for the best and worst case?
			4

		Solution: Algorithm checks if the matrix is lower triangular or not 1M Basic operation: if(A[i,j]!=0) 1M Basic operation executes once in Best case and n(n-1)/2 in the worst case 1+1M (Best case is when the first element A[0,1] it checks is non zero) so best case it runs once***	
	b)	Solve the recurrence using substitution method **$T(n)$=2$T(n^{1/2})$+log$_2 n$ T(1)=1**	4
		Solution:	

		$T(n) = 2T(n^{1/2}) + \log_2 n$ $\log_2 n = m \quad n = 2^m$ $T(2^m) = 2T(2^{m/2}) + m$ $S(m) = 2S(m/2) + m$ — 2m $S(m) = 2\left[2S(m/2^2) + m/2\right] + m$ $= 2^2 S(m/2^2) + 2m$ $\vdots$ $= 2^i S(m/2^i) + i \cdot m$ $m/2^i = 1 \Rightarrow i = \log_2 m$ $= m\, S(1) + m \log_2 m$ $S(1) = T(2) \quad T(2) \approx 2$ $= 2m + m \log_2 m$ $S(m) = O(m \log m)$ $\boxed{T(n) = O(\log n \log \log n)}$ — 2m	
	c)	Algorithms A1 has time complexity O(nlogn). During a test, this algorithm spends 10 seconds to process 100 data items. Derive the time algorithm A1 should spend to process 10,000 data items.	2
		Solution: T(A1)=C(nlogn) T(100)=C(100log100) C=10/(100log100) T(10000)=(10/(100log100))*10000*log(10000)=2000 sec	
3.	a)	Write a recursive sorting algorithm that uses divide and conquer technique which divides problem size by considering values in the list.	4
		Solution:	

		ALGORITHM *Quicksort*$(A[l..r])$
		//Sorts a subarray by quicksort
		//Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right indices
		// l and r
		//Output: Subarray $A[l..r]$ sorted in nondecreasing order
		if $l < r$
		$s \leftarrow Partition(A[l..r])$ //s is a split position
		$Quicksort(A[l..s-1])$
		$Quicksort(A[s+1..r])$
		Partition position is determined based on value of Pivot element	

	b)	Derive worst case time complexity for algorithm in Q.3a	4
		Solution: Assuming algorithm sorts the list in ascending order Worst case is when input list is already sorted in ascending order 1M All the splits will happen at extreme left leaving list of size one element less to be sorted further at each call, 1M so the number of comparisons is given by the recurrence $C_{worst}(n) = (n + 1) + n + . . . + 3 = (n + 1)(n + 2)/2 -3$ 1M $\in O(n^2)$. 1M	
	c)	What is upper bound on number of swaps selection sort performs?	2
		Solution: Number of swaps is n-1 for selection sort. So upper bound on number of swaps is Theta(n)	

4.	a)	Consider the string matching algorithm
		Algorithm stringmatch(P[0….m-1],S[0……n-1])
		// P-Pattern S-Text
		for i ← 0 to n-m do
		j ← 0
		while j < m and P[j] == S[i+j] do
		j ← j + 1
		if j == m
		return i
		return -1
		Given a text consisting of string of length 1000 with all characters as ‘a’ (aaaaaaaaaaa……..), how many comparisons, successful and unsuccessful, will this algorithm make in searching for each of these patterns?
		i) baaaa
		ii) ababa
			4

		Solution i) 996 2M ii) 1992(996 successful +996 unsuccessful) 2M	
	b)	Design an algorithm using Brute force approach to determine number of inversions present in an array. An inversion of an array A[1..n] of n distinct integer elements is a pair such that i < j and A[i] > A[j]. For example{5,9,10,4,8,7,3,6} has a total of 18 inversions	4

Solution:

```
Algorithm count_inversion_BruteForce (A[], n)
{
count ← 0
 for i ← 0 to n-1 do
   for j ← i+1 to n-1 do
          if (A[i] > A[j])
               count  ← count+1
return count.
}
```

c) State Master Theorem to solve the recurrences — 2

Solution:

$T(n) = aT(n/b) + f(n)$ where $f(n) \in \Theta(n^d)$, $d \geq 0$

<u>Master Theorem</u>: If $a < b^d$, $T(n) \in \Theta(n^d)$

If $a = b^d$, $T(n) \in \Theta(n^d \log n)$

If $a > b^d$, $T(n) \in \Theta(n^{\log_b a})$

---

5. a) Explain source removal method to determine topologically sorted ordering of vertices. What is algorithm design strategy used? — 4

Solution:

Algorithm SourceRemoval_Toposort(V, E) 3M

```
L ← Empty list that will contain the sorted vertices
S ← Set of all vertices with no incoming edges
while S is non-empty do
   remove a vertex v from S
   add v to tail of L
   for each vertex m with an edge e from v to m do
      remove edge e from the graph
      if m has no other incoming edges then
         insert m into S
if graph has edges then
   return error (not a DAG)
else return L (a topologically sorted order)
```

Design strategy: Decrease and conquer (Decrease by one approach) 1M

b) Write Decrease by a factor of 3 algorithm to solve fake coin puzzle. How faster it is as compared to decrease by a factor of 2 approach, in which we split the coins into two piles? — 4

Solution:

Algorithm find_fake_coin(coins) 3M

if n = 1 then

the coin is fake
else
divide the coins into piles of A = ceiling(n/3), B = ceiling(n/3),
and C = n-2*ceiling(n/3)
weigh A and B
if the scale balances then
iterate with C
else
iterate with the lighter of A and B

Decrease by factor of 3 algorithm runs $\log_2 3$(1.6) times faster than Decrease by a factor of 2 algorithm 1M

c) Starting with initial sequence 2

← ← ← ←
A C E R

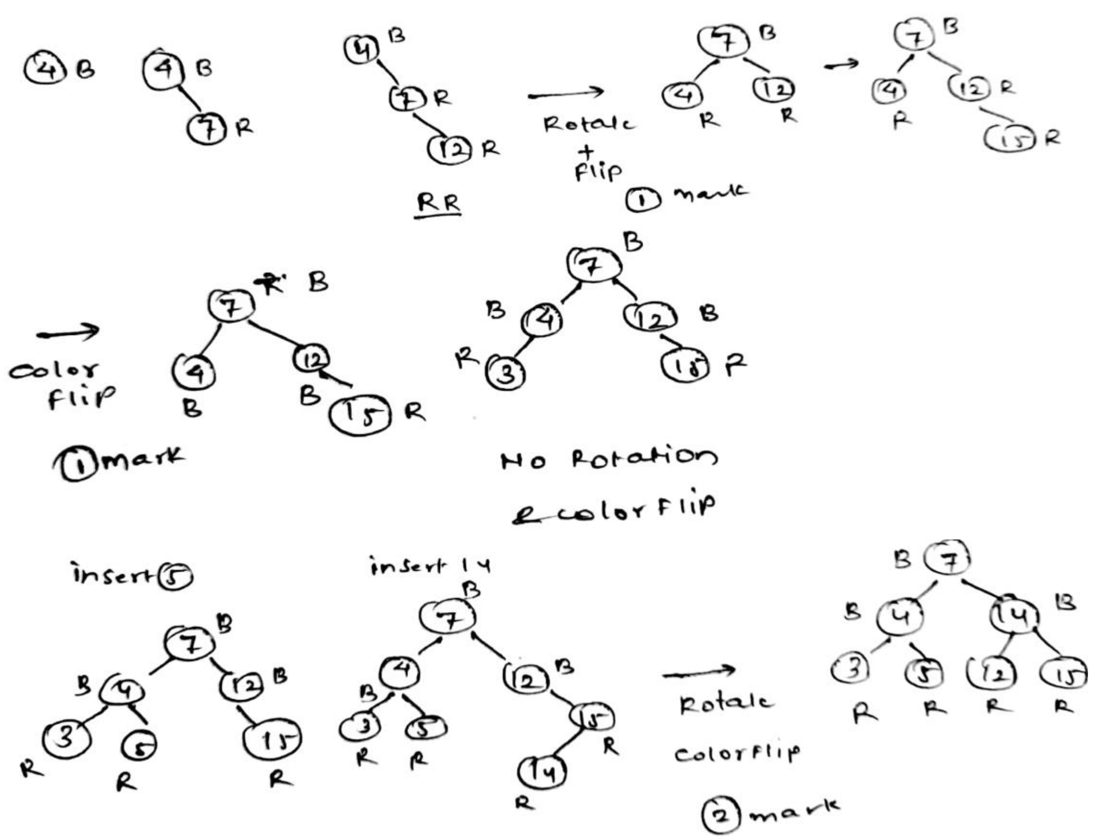
Generate next four permutations using Johnson Trotter method

Solution

A C R E 0.5M each
A R C E
R A C E
R A E C

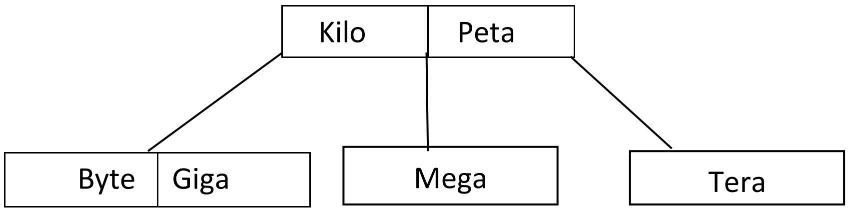
6 a) Construct a Red Black Tree for the key values 4, 7, 12, 15, 3, 5, 14. Show all the steps. Use R to denote Red and B to denote Black color. 4

Solution:



b) What is relation between a B Tree and 2-3 Tree?

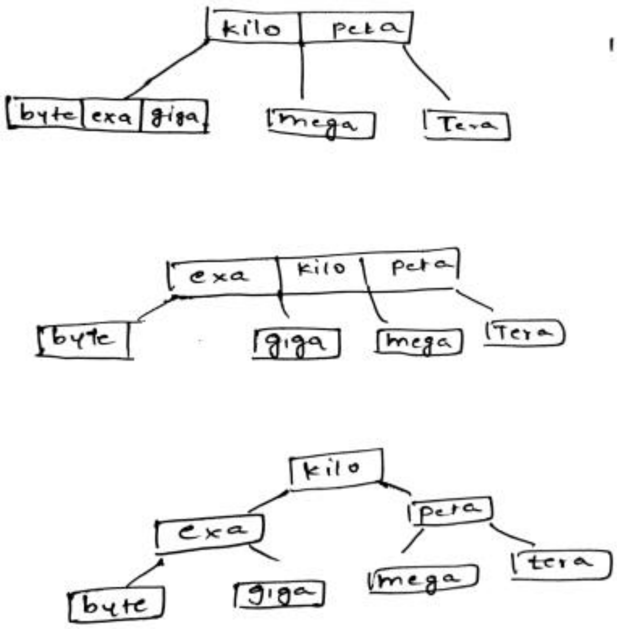
Insert the string ‘Exa’ in the given 2-3 Tree. Show the resultant tree after insertion. — 4



Solution:

2-3 Tree is a B Tree of order 3 1M

Tree construction 1M each step



c) What is a Heap? What is time complexity of constructing a Heap using bottom up Technique? — 2

Solution:

Definition 1M Time complexity 1M

<u>Definition</u> A *heap* is a binary tree with keys at its nodes (one key per node) such that:

- It is essentially complete, i.e., all its levels are full except possibly the last level, where only some rightmost keys may be missing
- The key at each node is ≥ (<=)keys at its children in case of max (min)heap

Time complexity of constructing a Heap using bottom up Technique $\Theta(n)$