SRN

PES UNIVERSITY

PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

UE14CS251

MAY 2016: END SEMESTER ASSESSMENT (ESA) B.TECH. IV SEMESTER

UE14CS251- Design and Analysis of Algorithms

Time: 3 Hrs | Answer All Questions | Max Marks: 100

1. a) Derive an asymptotic running time complexity of the adjacent algorithm in terms of θ. Assume n is a power of 2. — 6

```
Algorithm Foo()
Sum ← 0, i ← 1
while (i ≤ n)
        for j ← i to n
                Sum ← Sum + i + j
        i ← 2 * i
return Sum
```

b) For the adjacent algorithm, considering "Moving a disk" as a basic operation, derive the asymptotic running time complexity of the algorithm. — 6

```
Algorithm Hanoi(n, Src, Aux, Dst)
//Input: n (∈ Z⁺) disks and three pegs.
        if (n = 0)
                RETURN
        Hanoi(n-1, Src, Dst, Aux)
        Move disk n from Src to Dst
        Hanoi(n-1, Aux, Src, Dst)
End
```

c) Define "Big Oh" O(g(n)) and "Big Theta" Θ(g(n)), and show that $n(n-1)/2 \in O(n^3)$ and $n(n-1)/2 \in \Theta(n^2)$. — 8

2. a) Design a brute-force string matching algorithm to match a pattern of length m in a text of length n characters. Derive worst-time asymptotic running time efficiency of the algorithm. — 6

b) Let us define a binary tree is weight-balanced if the difference of number of nodes in the left and the right sub-trees is not more than one. Design an O(n) algorithm to validate if the given binary tree is weight-balanced, where n is the number of nodes in the tree. — 6

c) — 8

$$\left[\begin{array}{c|c} C_{00} & C_{01} \\ \hline C_{10} & C_{11} \end{array}\right] = \left[\begin{array}{c|c} A_{00} & A_{01} \\ \hline A_{10} & A_{11} \end{array}\right] + \left[\begin{array}{c|c} B_{00} & B_{01} \\ \hline B_{10} & B_{11} \end{array}\right]$$

Two matrices of order n x n can be added by splitting each input matrix into four n/2 x n/2 matrices and adding them recursively. Using this concept, design a divide-and-conquer algorithm to add two matrices of order n x n. Derive worst-time asymptotic running time efficiency of the algorithm.

3 a) Derive θ(n) as the worst-case time efficiency of the adjacent algorithm with "**k** ← j" as the basic operation. — 6

```
Algorithm HeapBottomUp(H[1..n])
//An array H[1..n] of orderable items
for i ← ⌊n/2⌋ downto 1 do
        k ← i
        while (2*k ≤ n) do
                j ← 2*k
                if (j < n)
                        if (H[j+1] > H[j]) j ← j+1
                if (H[k] ≥ H[j])
                        break out of the while loop
                Swap (H[j], H[k])
                k ← j
return H
```

P.T.O

SRN

b) Design a presorting-based algorithm to find the smallest possible mean of k elements in an array of n elements. **6**

```
Algorithm SmallestKMean(A[1..n], k)
```

c) Consider the problem of searching for genes in DNA sequences using Boyer-Moore string matching algorithm. A DNA sequence is represented by a text on the alphabet {A, C, G, T}, and the gene or gene segment is the pattern. **8**

Construct the bad-symbol shift table and good-suffix shift table for the following gene segment: TAATAA.

Apply Boyer-Moore algorithm to locate first occurrence of the above pattern in the following DNA sequence: TAATCAGGAAAGCGTAATAATAATA.

4. a) Binomial Coefficient C(n, k) can be defined by the recurrence: **6**

$C(n, k) = C(n-1, k-1) + C(n-1, k)$ for $n > k > 0$, and
$C(n, 0) = C(n, n) = 1$ for $n \geq 0$.

The recurrence is implemented in the following algorithm:

```
Algorithm BinomialCoeff (n, k)
if (k = 0) or (k = n) return 1
else return BinomialCoeff(n-1, k-1) + BinomialCoeff(n-1, k)
```

What is the glaring issue the above algorithm has, which can be solved by dynamic programming?

Design an algorithm using dynamic programming to find the binomial coefficient defined in the above recurrence, which has significantly lower running-time efficiency than the above algorithm.

b) Construct a Huffman tree for the following data and obtain its Huffman code. **6**

Character:	A	B	C	D	E	F
Frequency:	0.5	0.35	0.5	0.1	0.4	0.2

c) **8**

```
ALGORITHM Floyd(W[1..n, 1..n])
    //Implements Floyd's algorithm for the all-pairs shortest-paths problem
    //Input: The weight matrix W of a graph with no negative-length cycle
    //Output: The distance matrix of the shortest paths' lengths
    D ← W //is not necessary if W can be overwritten
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
    return D
```

Above algorithm is the well-known Floyd's algorithm to find shortest distance between every pair of vertices in a graph. Enhance the algorithm to find shortest paths themselves. (Hint: Along with D, you may want to generate another data structure, which holds some kind of information about the shortest paths. You may want to write another procedure, which takes i, j and the data structure, and prints shortest path from vertex i to vertex j.)

5. a) When do we call a Lower Bound of a problem is "tight"? **6**

Mention four known methods for establishing lower bounds.

b) Define P, NP and NP-Complete Problems. **6**

c) Explain Backtracking method for solving n-Queens problem (using state-space trees). **8**