**Birla Institute of Technology and Science Pilani**
**Dept. of Computer Science & Information Systems**
**Design and Analysis of Algorithms (CS 364)**

8 – 11 AM
Friday May 12, 2017

Time: 3 hour | *Comprehensive Exam (Closed Book)* | Maximum Marks: 80

**INSTRUCTIONS:**

1. There are **8** questions spread over **3** pages, make sure you have all of them. Attempt **ALL** questions
2. State clearly, any assumption that you make while solving the problem
3. Calculator is allowed in this exam

1. Victor was new to programming when he was asked to write a code for arranging integer numbers in decreasing order. He has implemented some logic and then submitted his code for testing. Test engineer Bob found the code very similar to the Bubble sort in terms of efficiency. His notion of efficiency was related to the time taken by the code. Bob had tested the code on three randomly generated data sets of integers having 220, 350 and 400 items and reported that the code took 195166, 492476 and 642826 mili seconds. Can you determine the time taken by Victor's algorithm for 150 items. [8]

2. `Horner's Rule` to evaluate a polynomial of the form $a_0 + a_1x + a_2x^2 + ... + a_nx^n$ devices a strategy that computes the polynomial in a way similar to $a_0+x(a_1+x(a_2+...+n(a_{n-1}+xa_n)...)$. Write a *pseudocode* to implement Horner's rule and also give proof of its correctness by clearly providing loop invariant. [3+5]

3. Tony wants to send a secret message to Holmes who is a secret agent and would use the message to unlock a really important security chest. The message is a pin number. Tony sends some cards to Holmes, each having written a number and a function on them. If Holmes could arrange all the functions in increasing order of their complexity he can get the pin by just looking on each card in order.

   (a) Device an algorithm for Holmes to decode the message. [3]

   (b) Determine the secret pin by considering the cards send by Tony as below [8]

   | | | |
   |---|---|---|
   | [3] $2^{\sqrt{2\log n}}$ | [2] $7^{\log\log n}$ | [5] $2^{2^{n+1}}$ |
   | [1] $\log(n!)$ | [9] $\log(\log^* n)$ | [7] $(\log n)^{\log n}$ |

   You must justify your answer using appropriate complexity measures

   (c) What is the complexity of algorithm you proposed? Provide derivation. [2]

4. Prove that maximum degree $D(n)$ of any node in an $n$-node Fibonacci Heap is $O(\log n)$. [11]

5. Write an algorithm for computing $a^b$ where $a$ and $b$ are positive integers using the repeated squaring method. Assuming that (1) $a = O(n)$, and $b = O(n)$; (2) you can multiply two $m$ bit numbers in $O(m^2)$ time; and (3) you can add two $m$ bit numbers in $O(m)$ time; find the best possible upper bound on worst case time complexity in $O()$ notation as a function of $n$. [5 + 5]

6. Prove that `RP` $\subseteq$ `NP` [10]

7. Prove that the following language is NP-complete: [10]
   `TMSAT` $= \{(\alpha, x, 1^n, 1^t) \mid \exists u \in \{0,1\}^n$such that $M_\alpha$ outputs 1 on input $(x, u)$ within $t$ steps$\}$
   where $M_\alpha$ denotes the *Deterministic Turing Machine* represented by the string $\alpha$.

8. We formulate the *Load Balancing Problem* as follows. We are given a set of $m$ machines $M_1, ..., M_m$ and a set of $n$ jobs; each job $j$ has a processing time $t_j$. We seek to assign each [10]

job to one of the machines so that the loads placed on all machines are as "balanced" as possible. More concretely, in any assignment of jobs to machines, we can let $A(i)$ denote the set of jobs assigned to machine $M_i$; under this assignment, machine $M_i$ needs to work for a total time of $T_i = \sum_{j \in A(i)} t_j$, and we declare this to be the load on machine $M_i$. We seek to minimize a quantity known as the makespan; it is simply the maximum load on any machine, $T = \max_i T_i$. Prove that the following algorithm is a $2-$approximation algorithm for the *Load Balancing Problem*:

```
Greedy-Balance:
Start with no jobs assigned
Set T_i = 0 and A(i) = φ for all machines M_i
For j = 1, ..., n
  Let M_i be a machine that achieves the minimum min_k T_k
  Assign job j to machine M_i
  Set A(i) ← A(i) ∪ {j}
  Set T_i ← T_i + t_j
EndFor
```

---