Programming and Data Structures in Python

Duration: 2.5 Hours
Maximum Marks: 35

You may use any algorithm described in class as is without writing it down (except binary search in problem 2, for obvious reasons). You may submit handwritten answers or email your code. Send your code by email to kumar@cmi.ac.in from your CMI email id.

1. Write a python function `findNumber` whose aim is to determine the value of a number `N` (you may assume `N` ≥ 0) as follows: `findNumber` can call a function `compare(x)` with any value `x` and this compare function will return `True` if $x \leq N$ and `False` otherwise. `findNumber` should print out the value of `N` once it determines the value of `N`. How many calls of `compare` does your `findNumber` make (as a function of the value of `N`) ?

   (5 marks)

2. You are given a list of numbers $l = a_0, a_1, \ldots, a_n$. Your aim is to compute the number of pairs (a_i, a_j), $i \neq j$ such that $a_i < a_j$. Write a function `opairs` which achieves this. What is the complexity of your algorithm? [Can you solve this problem in $O(nlogn)$? How?] *(7 marks)*

3. Why did the function `permute`, to enumerate permutations iteratively, written in class (Lecture 09) report an incorrect answer (the code is listed below for those of you who don't have a computer or cannot connect to the network)?

```
def permute(n):
   ans = []
   ls = [0]*n
   pos = 0
   while (pos >= 0):
     if pos < n:
       ls[pos] = ls[pos] + 1
       if ls[pos] > n:
         ls[pos] = 0
         pos = pos - 1
       else:
          pos = pos + 1
     elif len(ls) == len(list(set(ls))):
       ans.append(ls)
       pos = pos - 1
     else:
       pos = pos - 1
   return(ans)
```

On input N it returns a list with $N!$ copies of the list $[0, 0, ..., 0]$. Explain why this is the case and suggest a way to fix this bug. *(5 marks)*

4. Consider the following two programs:

```
Prog 1:

x = 1

def g():
  x = x
  y = x
  print(y)

g()
```

```
Prog 2:

x = 1

def g()
  x = 1
  y = x
  print(x)

g()
```

   Program 1 generates an error ("UnboundLocalError: local variable 'x' referenced before assignment") while Program 2 runs as expected. Explain why. *(3 marks)*

5. We can define an ordering on the set of permutations of $1..n$, by comparing them lexicographically. In other words, for permutations σ and π ($\sigma \neq \pi$), let i_0 be the first position where they differ. Then $\sigma < \pi$ if $\sigma[i_0] < \pi[i_0]$ and $\sigma > \pi$ otherwise.

   Given a permutation σ, the previous permutation is the largest permutation among all permutations smaller than σ.

   Write a Python function `prevPermutation` which takes a list of integers as argument and returns the previous permutation of the given permutation, or an empty list if it does not exist. You may assume that the given list is permutation of $1..n$.

```
prevPermutation([1,2,3]) = []
prevPermutation([3,2,1]) = [3,1,2]
prevPermutation([1,2,8,7,6,3,4,5])  = [1,2,8,7,5,6,4,3]
```

   What is the complexity of your function? Why ? *(8 marks)*

6. Write a Python function `incSeq(n,k)` which produces all strictly increasing sequences (as lists) of length k from among $1, 2, \ldots, n$. For example

```
incSeq(2,3) = []
incSeq(3,3) = [[1,2,3]]
incSeq(4,3) = [[1,2,3],[1,2,4],[1,3,4],[2,3,4]]
```

   The order in which the lists appear in the answer is not important. *(7 marks)*
