Programming Contest (Part I)

Data Structures Lab (CS 210)

2012

Q1. **(10 marks)** An intelligent robot is taught the definition of "sorting distinct integers in an ascending order", but not taught any sorting algorithm to achieve the task. One day, the robot is instructed to sort n distinct integers from $\{1, 2, \ldots, n\}$ in an ascending order and it devises the following algorithm itself.

Generate a uniformly-random permutation of the given integers, and check whether this permutation is sorted in an ascending order. If it is not, then repeat the step above. Otherwise, stop with the sorted output.

The robot can achieve the task above as it is capable of generating a sequence of uniformly-random integers in any given range, and verifing whether a given sequence of distinct integers is sorted in an ascending order. However, for very large n, the robot runs out of memory space as it didn't implement it's algorithm in an in-place manner.

Your job is to implement the algorithm used by the robot in an in-place manner using C/C++. You are allowed to store the input elements in an array of n integers. Only a constant number of extra variables can be used in addition to this array. Assume (even though it's only an approximately correct assumption) that the pseudo-random number generator *rand*() function in your program can generate a sequence of uniformly-random integers in any given range of integers, if you can use it properly. It's your responsibility to use the *rand*() function properly.

Input: A sequence of n distinct integers in the range $[1, n]$, manually typed on the terminal, separated by space and terminated by "ENTER". The value of n would be at most 10000, and it needs to be determined after "ENTER" is pressed.

Output: The integers in an increasing order, separated by space on the terminal.

[Termination of your code need not be guaranteed, but the algorithm must be correct within the constraints given in the problem statement.]

Q2. **(10 marks)** Write the C/C++ code of an $O(n^{\frac{1}{2}} \log n)$-time algorithm to identify all prime numbers of the form $2^k + 1$, which are less than or equal to a large number n. Note that k can only be a positive integer and 1 is not a prime number.

Input: An integer n typed on the terminal followed by "ENTER".

Output: All prime numbers of the form $2^k + 1$, which are less than or equal to n, in an increasing order. The numbers should be separated by space on the terminal.

[5 marks would be awarded if your program works for any $n \leq 10^4$, and full 10 marks would be awarded if it works for any $n \leq 10^8$ within the constrants given the problem statement. You are not allowed to use any prime number table that directly gives such primes. Your algorithm should be able to identify the primes without any prior knowledge about the sequence of prime numbers.]