CSL 101 Introduction to Computers and Programming
Minor 2, Sem I 2008-09, Max 40, Time 1 hr

Name ______________________ **Entry No.** ______________________ **Group**

Note (i) This question paper has 3 pages
(ii) Write only in the space provided below each question including back of the sheets. Use the extra sheets for rough work.
(iii) Write your answers neatly and precisely. You won't get a second chance to explain what you have written.
(iv) Note that the blank spaces do not carry equal marks. Marks will be awarded on the basis of overall correctness.

1. The following higher order function represents the n-th composition of a function f with itself. **(2 + 8 marks)**

```
let rec selfcomp f n = if n = 0 then (function x -> x) else
                   function x -> f ( selfcomp f (n-1) x) ;;
```

(i) What is the type of `selfcomp`?

$$(\alpha \rightarrow \alpha) \rightarrow (\texttt{int} \rightarrow (\alpha \rightarrow \alpha)))$$

which is displayed as `('a -> 'a) -> int -> 'a -> 'a .`
(ii) How would you use `selfcomp` (i.e., specify the appropriate parameters) to generate the 10th term of the arithmetic sequence

$$1, 4, 7, 10 \ldots 1 + 3(i - 1)$$

```
selfcomp (function x -> x+3) 9 1
```

starting with 1, apply the "add 3" function 9 (=10-1) times.

and the 5th term of the geometric series

$$2, 6, 18, 54 \ldots 2 \cdot 3^{i-1}$$

```
selfcomp (function x -> x*3) 4 2
```

starting with 2 apply the "multiply by 3" function 4 (=5-1) times.

2. A point (x, y) is *dominated* by (x', y') if $x \leq x'$ and $y \leq y'$. Given a set S of n points (x_i, y_i), a point $p \in S$ is *maximal* if it is **not** dominated by any point in S. For example in the set $\{(3, 4), (2, 1), (4, 0)\}$, points $(3, 4)$ and $(4, 0)$ are maximal. Complete the program below that computes all the maximal points of a given list `l` of points. The input list is given in non-increasing order of x coordinates. The algorithm scans the points of `l` and determines if the current point is maximal based on the y coordinates of all the points scanned before. The parameter `max` keeps track of the largest y coordinate and `last` keeps track of the x-coordinate of the previous point. Note that the point with the largest x coordinate is always maximal. For simplicity you may assume that all points are distinct. **(10+5 marks)**

```
let maximal l = let rec prefixmax lst last max = (*l is reverse sorted on 1st
                                                               component *)
    match lst with
   [] -> [] |
```

```
(a,b)::tail -> if a > last then raise Not_sorted

    else if b > max then____(a,b)::( prefixmax tail a b)________

          else prefixmax____tail a max


         in match l with [] -> [] |
          (a,b)::tail ->   (a,b)::(prefixmax tail a b ) ;;
                            ------------------------------------------
```

Rewrite the function `prefixmax` as a tail recursive function with minimal modifications.

```
let rec prefixmax  lst last max output =
      match lst with
  [] -> output |
   (a,b)::tail -> if a > last then raise Not_sorted
       else if b > max then     prefixmax tail a b  output@[(a,b)]
             else prefixmax   tail a max output ;;
```

The most common mistake was failure to append (a,b) when it was a maximal point, i.e., it won't produce any output in part 1.
In part 2, you needed an extra parameter to create an output list - without that it won't be tail recursive.

3. To compute all the prime numbers in $[2, n]$, the *sieve of Eratosthenes* proceeds by eliminating all factors of 2, then all factors of 3 etc. Initially an array of size n+1 is initialised to 0. In the i-th iteration, all the multiples of p_i (the i-th prime where $p_1 = 2$) are marked as 1 (not prime). In the end, i is prime if `a[i] == 0` (== is the Java comparator operator for equality testing). **(8+4+3 marks)**
(i) Complete the following program.

```
          for (i = 2 ;  i <= n ; i++ ) /* 2 is prime */
               {
            if (a[i] == 0)
                 {                         \\*** part (ii)
                    j = _2__;

                   while  ( j <= _(n/i)__)
                     {
                       a[__ i*j___] = 1 ;
                        j++ ; // Invariant of while: _____________
                               a[2i], a[3i] ....a[ji] is set to 1
                          (a[i] is 0 since i is prime !)

                          } //end while
                      } //end of if

                } //end of for
   for ( i= 2; i <= n ; i++) { if (a[i] == 0) System.out.println( i); }
 }
}
```

Note that starting from j = i is also correct but j = i+1 is not correct since $i \times i$ is not prime and must be marked as 1. Since i is the smallest prime factor it will not be marked earlier. With $j = i$, the proof of part (b) must address it. (ii) In the line marked `***`, *what property* can you claim about `i` and *why* ?
The number i is prime. Inductively assume that all prime numbers have been correctly identified (before i). Clearly i is not a multiple of any **prime** number j, $j < i$. Therefore i must be prime. Note that we are only marking the multiple of prime numbers as 1 and not the multiple of all numbers.
(iii) What is the number of basic operations as a function of n ? Each non-prime number i is crossed out at most the number of prime factors. The right expression is $n/2 + n/3 + n/5 + ...n/p_i \ldots$ where p_i is the i-th prime. This is bounded by $\sum_{i=2}^{n}(n/i) \leq n(1/2 + 1/3 + \ldots 1/n) \leq n \cdot H_n$ where H_n is the n-th Harmonic and is roughly $\log_e n$.
1 mark was deducted if closed form solution was not given. A tighter answer (hard to prove) is that the sum is bounded by $n \log \log n$.