CSL 101 Introduction to Computers and Programming

Major Sem I 2008-09, Max 70, Time 2 hr

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

Note (i) Write only in the space provided below each question. You can use the backside for roughwork.
(ii) Write yor answers neatly and precisely. You won't get a second chance to explain what you have written.

1.
```
let rec remdup lst = match lst with
          [] -> []
         | a::b -> let rec remlist x y = match y with
           [] -> []
         | hd::tail -> if x = hd then (remlist x tail) else
            hd:: (remlist  x tail)
                       in
               a::(remdup (remlist a b)) ;;
```

1 mark for including `a` (by cons-ing it in front); 2 marks for recursive call to `remdup` to remove *other* duplicates from a list not containing `a`, which is the result of `remlist a b`). 0.5 marks less for the less efficient `a:: (remlist a (remdup b))`.

The type is $\alpha \rightarrow (\alpha \text{ list} \rightarrow \alpha \text{ list})$. 1 mark deducted for the over-specific instance where α is replaced by `int`. Marks deducted for $\alpha * \alpha$ list $\rightarrow$ α list).

`remlist x y` returns a list consisting of elements of list `y` **excluding** each occurrence of element `x`.

2.
```
let rec f n = if n = 1 then 1
         else if n = 2 then 2
        else let rec ftr(n,i,a,b,c) =
                    if i=n then a else ftr(n,i+1, 3*a+c, a, b)

             in  ftr(n,3,3,2,1);;
```

Variations on this pattern are possible. But at least 3 *consecutive* values of the function have to be "carried". 1 mark for keeping the outer function the same type; 1 mark for the base cases; 1 mark for the correct number of parameters of the tail recursive helper function; 1 mark for the correct termination condition; 2 marks for the tail recursive call; 2 marks for the correct initialization of the tail recursive function. For the version that "counts down", 2 marks were allotted for the correct termination, i.e., at $n = 3$, and 1 for the initialization.

3. Professor **OverConfident** claims that a circular list can be designed without using recursive structure or references. He says that arrays suffice - can you justify his claim ? Show how to represent the circular list $a_3 \Rightarrow a_2 \Rightarrow a_4 \Rightarrow a_1 \Rightarrow a_5 \Rightarrow a_3$ where the data fields are 5.4 , 2.0, 0.4 , 23.5 and 9.0 respectively. You don't have to write Ocaml or Java syntax but draw a schematic diagram using array(s) for this example. **(5 marks)**
You can use an array of pairs, (x_i, y_i) where x_i contains the value of a_i and y_i contains the position of the array where the successor of a_i is stored, or alternately two arrays for x_i and y_i. So one solution is (23.5 , 5) (2.0, 4) (5.4, 2) (0.4, 1) (9.0, 3) using array index 1,2,..5.

Most solutions used circular arrray with no separate field for the address which is not acceptable since any change in pointers will lead to changing the positions of elements themselves.

4. (a) [(10,0); (-1,1); (3,2)]
   and
   []
   Note that in the example, terms with 0 as a coefficient are omitted. This keeps the representation efficient in space. So in particular the trivial polynomial that is the additive identity is represented as the empty list. This choice maps nicely to the base case for lists, and is unambiguous and efficient.The lists can be in ascending order of exponents or descending order of exponents. neither is inherently better, but keeping them sorted is key for efficiency.

   (b) Three crucial properties (once we have specified in the question that coefficient and exponent are both integers): (a) For each (a, i) in the list then $i \geq 0, a \neq 0$; (b) If (a, i) in the list, then there does not exist any other element (b, i) in the list for any b; (c) The list is sorted in strictly ascending (or strictly descending) order of the second components of the pairs (namely the exponents).

   Various other inessential properties can be stated. Note that (c) implies (b).

   (c)
```
(* assuming lists are sorted in ascending order of second component of the
pairs, ie. on exponent of the terms and that no coefficient is zero.
We must ensure that the result respects these constraints *)

let rec addpoly p1 p2 = match (p1,p2) with
                          ([],l2) -> l2
                        | (l1,[]) -> l1
                        | ((a,b)::xs, (c,d)::ys) ->
                              if b=d then if (a+c)=0 then addpoly xs ys
                                          else (a+c,b)::(addpoly xs ys)

                                    else (* b > d *) (c,d)::( addpoly p1 ys);;
```
   2 marks for base cases in pattern matching – note that pattern `([],[])` is subsumed; 2 marks for adding the coefficients of equal exponent terms, 1 for taking care to omit the term is the resultant coefficient is zero, 1 mark each for the other two recursive calls.

   (d) Induction Hypothesis: (Induction is on the sum of the length of the lists, not on the degree of the polynomials. Also, the Induction Hypothesis must apply to the recursive calls).

   Assume that for all polynomials p_1, p_2 whose list representations `l1,l2` are of length m, n such that $m + n \leq k$, `addpoly l1 l2` returns the correct representation of the polynomial $p = p_1 + p_2$.

   Some credit for any kind of induction hypothesis. More for stating it in terms of lists; more for identifying a list or some operation e.g., minimum of the lengths, etc.

5. Consider a special case of sorting where numbers are integers in the range 0..k (k is known in advance). We will use the following idea to sort. By scanning the input array, we will keep a count on the number of elements of a particular value (0 , 1 ..k). The array cnt[] is such that cnt[i] is the number of elements with value i ($0 \leq i \leq k$). Then the sorted sequence is cnt[0] 0's followed by cnt[1] 1's ...cnt[k] k's (some of the values may be absent). For example, if n = 5, k = 4 and input is [4, 1, 2, 0, 1], then cnt[0]= 1, cnt[1] = 2, cnt[2] = 1, cnt[3] = 0 and cnt[4] = 1. So, the sorted output is [0, 1, 1, 2, 4], i.e., (one 0) , (two 1's) , (one 2), (zero 3) (one 4).

Complete the program below that implements this approach . Note that while writing the output, the starting location of is depends on cnt[j] $j \leq i$. **(12+3 marks)**

```
 import java.lang.Math ;

 class intsort{

 public static void main (  String argv [] )

     throws java.io.IOException
{
          int k ; //range
          int n ;// number of elements to be sorted
          int i, j //no more variables to be used

            int[] cnt = new int[k+1] ;
            int[] A = new int[n]; //assume that user has supplied n,k

          for (i =0 ; i <=k ; i++ ) {cnt[i] = 0;}

          for (i = 0; i <= n-1; i++){

                cnt[_A[i]__] = _cnt[A[i]] + 1__ ;

Loop invariant: The number of j, (0 <= j <= k) in A[0] ..A[i] is stored in
                                cnt[j]       }//end for


            j = 0;
              for (_i =0__ ; _i <= k_ ; __i++__) {
              while( cnt[i] > 0)  {

                 A[j] = _  i __ ; j++ ;

                 _cnt[i]__ = __cnt[i]__ - 1 ;
                   }// end while

      Loop invariant ___The array A contains all 0s, all 1's ..all i's
            in sorted order.                                 }//end for


            System.out.println("The sorted sequence is ") ;
           for (i=0 ; i<= n-1 ; i++) {
               System.out.println( _A[i]__) ;
                 }
         } //end main
 } //end class
```

(ii) What is total number of operations in terms of the relevant parameters ?
Initialising array cnt[] takes k operations. Reading, counting and writing the output in A takes O(n), so total O(n+k).

1 mark given for O(n) and 0 for answers like $O(n^2)$, $O(n \cdot k)$ etc.

6. The following program generates all the k-wise combinations of distinct objects in an array using the following classic definition. The k-wise combination is the union of all (k-1) combinations that contains a specific object X and all k-combinations that do not contain X.
The following program assumes that the objects are given in an array `A` and we generate a combination by filling up the entries of an array `B` of size k. The array `B` is printed using a predefined function `printarray` when we have chosen a subset of k objects. In the recursive function `recurcomb`, the parameter `i` keeps track of which object is being considered in `A` and `j` indicates that `j-1` objects have been chosen (locations 0 .. (j-1) positions are filled in `B`). The main program calls `recurcomb` with appropriate parameters. Complete the program that prints every combination exactly once. **(12+5+3 marks)**

```
// combin.java
 import java.io.* ;
 import java.lang.Math;

 class combin {

  public static void main (String argv [] )

   throws java.io.IOException
       {
    int [] A = new int [n] ;// Initially array A contains [0,1,2,3,...(n-1)]
    int [] B = new int [k] ; //The k chosen output will be stored in B

  System.out.println ("The choices are");
         recurcomb(A , B, 0 , 0 );
        }// end main

  static void recurcomb (int [] A , int [] B, int i, int j)
//chooses (B.length - j) objects from A[i] ..A[n-1]

    {
           if ( (A.length - i) >= (B.length - j))     {

            if (B.length == j) {Array.printarray(B); //k objects chosen
                           System.out.println();}

              else { recurcomb ( A, B, i+1 , j  ) ;//not choose A[i]
                                        -----------------


                          B[_j__] = __A[i]__ ; //choose A[i]
```

```
                    recurcomb(A, B, i+1, j+1))        ; //choose A[i]
                       ------------------------------------------
                  }//end else
           }// end if
       } // end recurcomb
  } // end class
```

(ii) Write an appropriate induction hypothesis to prove the correctness of the program (no proof required).
Induction Hypothesis $\forall n[\forall k$ `recurcomb (A, B, n, k)` prints each distinct subset of k elements exactly once].

Proof (not required): It is induction on n since we are using the identity

$$\binom{n}{k} = \binom{n-1}{k} \bigcup \binom{n-1}{k-1}$$

This identity **is not be proved** - it is already given. You must prove the correctness of `recurcomb`. When $k > n$, it doesn't print anything and for $k = n$, it prints all elements, otherwise it uses the above identity to invoke I.H. Note that k does not decrease in the first term so, doing induction on k is not obvious.
State the base case(s). n = 1. We must check that it holds for all k. For $k > 1$, nothing is printed. For $k = 1$, we print the one choice. You can verify that the function is never called with $k = 0$, since it terminates when k==j.

Most answers didn't have an I.H. and sometimes had an I.H. which was the above identity. Induction on k is incorrect the way the program is written and base case with I.H. is not acceptable.
(ii) How much space (as a function of n) is used by the program ?
The space used are the two arrays A and B and the local variables i and j. Since A and B are passed by reference (and not copied), the total space is $O(n + k)$ (including the copying space of i and j).

The number of choices is not space in the program but is printed and should not be counted in space (it is counted in time).