Problem 1

Here is a sample solution:

```
fun is_older (date1 : int * int * int, date2 : int * int * int) =
    let
        val y1 = #1 date1
        val m1 = #2 date1
        val d1 = #3 date1
        val y2 = #1 date2
        val m2 = #2 date2
        val d2 = #3 date2
    in
        y1 < y2 orelse (y1=y2 andalso m1 < m2)
                orelse (y1=y2 andalso m1=m2 andalso d1 < d2)
    end
```

- Be lenient on how let-expressions are used. It is okay if there are no local val bindings. It is also okay if there are more (e.g., to avoid repeating the expression `y1=y2`).
- For the logic expression, it is okay to use `if ... then ... else ...` instead of `orelse` and `andalso`, but the logic should still be clear: starting by comparing the year, then the month, then the day. If the logic is hard to follow, give a 4 or 3.

Give a 3 for this sort of more imperative looking code:

```
fun is_older (date1 : int * int * int, date2 : int * int * int) =
    let val y1 = #1 date1
        val m1 = #2 date1
        val d1 = #3 date1
        val y2 = #1 date2
        val m2 = #2 date2
        val d2 = #3 date2
    in
        let val b1 = y1 < y2
        in
           if b1
           then true
           else let val b2 = y1 > y2
                in
                  if b2
                  then false
                  else ...
           ...
    end

```

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 2

Here is a sample solution:

```
fun number_in_month (dates : (int * int * int) list, month : int) =
    if null dates
    then 0
    else if #2 (hd dates) = month
    then 1 + number_in_month(tl dates, month)
    else number_in_month(tl dates, month)

```

Make sure the solution has clear recursive calls and clearly evaluates to `0` if `dates` is `null`. The solution does not have to be exactly like the sample above. For example, this solution also deserves a 5:

```
fun number_in_month (dates : (int * int * int) list, month : int) =
    if null dates
    then 0
    else (if #2 (hd dates) = month then 1 else 0)
         + number_in_month(tl dates, month)

```

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 3

Here is a sample solution:

```
fun number_in_months(dates : (int * int * int) list, months : int list) =
    if null months
    then 0
    else number_in_month(dates, hd months) + number_in_months(dates, tl months)

```

- Give a 3 if the solution does not use **`number_in_month`** as a helper function or if it is substantially longer than a single if-then-else expression.
- Give a 4 if it uses a let expression for not much reason (for a short expression that is used only once). Do this for all the remaining problems (we won't repeat this instruction for each problem).

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 4

Here is a sample solution:

```
fun dates_in_month (dates : (int * int * int) list, month : int) =
    if null dates
    then []
    else if #2 (hd dates) = month
    then (hd dates)::dates_in_month(tl dates, month)
    else dates_in_month(tl dates, month)

```

Give at most a 4 for any solution that uses ML's append operator (the **`@`** character). Otherwise follow similar instructions as for earlier problems.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 5

Here is a sample solution:

```
fun dates_in_months(dates : (int * int * int) list, months : int list) =
    if null months
    then []
    else dates_in_month(dates, hd months) @ dates_in_months(dates, tl months)

```

Give a 3 if the solution does not use **`date_in_month`** as a helper function or if it is substantially longer than a single if-then-else expression.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 6

Here is a sample solution:

```
fun get_nth (lst : string list, n : int) =
    if n=1
    then hd lst
    else get_nth(tl lst, n-1)
```

Give at most a 3 if the solution uses an algorithm much more complicated than the code above.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 7

Here is a sample solution:

```
fun date_to_string (date : int * int * int) =
    let
        val names = ["January", "February", "March", "April", "May", "June",
                     "July", "August", "September", "October", "November",
                     "December"]
    in
      get_nth(names,#2 date) ^ " " ^ Int.toString(#3 date)
      ^ ", " ^ Int.toString(#1 date)
    end
```

Give at most a 2 if the solution does not use a list of month names in some way. However, you can give a 5 for a solution that puts the list of month names outside the function. Give at most a 4 if the solution does not use **`get_nth`** with the list of month names as an argument.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 8

Here is a sample solution:

```
fun number_before_reaching_sum (sum : int, lst : int list) =
    if sum <= hd lst
    then 0
    else 1 + number_before_reaching_sum(sum - hd lst, tl lst)
```

Any nicely formatted solution of roughly this length is probably good style, but look for the logic of a recursive call with argument **`sum - hd lst`**, giving at most a 4 if it is difficult to find.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 9

Here is a sample solution:

```
fun what_month (day_of_year : int) =
    let
       val month_lengths = [31,28,31,30,31,30,31,31,30,31,30,31]
    in
       1 + number_before_reaching_sum(day_of_year, month_lengths)
    end
```

Give at most a 2 if the solution does not use a list of month lengths and the **`number_before_reaching_sum`** function in some way. However, you can give a 5 for a solution that puts the list of month lengths outside the function.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 10

Here is a sample solution:

```
fun month_range (day1 : int, day2 : int) =
    if day1 > day2
    then []
    else what_month day1 :: month_range(day1 + 1, day2)
```

Give at most a 3 for a solution that uses ML's append operator (the `@` character). Give at most a 4 for a solution that has more than a single if-then-else expression.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 11

Here are two sample solutions:

```
fun oldest (dates : (int * int * int) list) =
    if null dates
    then NONE
    else let
             val ans = oldest(tl dates)
   in
       if isSome ans andalso is_older(valOf ans, hd dates)
       then ans
       else SOME (hd dates)
   end

(* arguably better alternate solution avoiding isSome / valOf *)
fun oldest (dates : (int * int * int) list) =
    if null dates
    then NONE
    else let
             fun f dates =
                 if null (tl dates)
                 then hd dates
                 else let
                          val ans = f (tl dates)
                      in
                          if is_older(ans, hd dates)
                          then ans
                          else hd dates
                      end
         in
             SOME(f dates)
         end
```

Give at most a 3 if oldest could be called recursively twice with the same list (probably **`tl dates`**). Give at most a 4 if **`is_older`** is not used.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problems 12 and 13

You do not need to provide feedback on problems 12 and 13 (the challenge problems), but you are welcome to give text feedback on these problems if you wish. Here are sample solutions for the challenge problems although there are other equally good if not better approaches you could take:

```
(* quadratic algorithm rather than sorting which is nlog n *)
fun mem(x : int, xs : int list) =
    not (null xs) andalso (x = hd xs orelse mem(x, tl xs))
fun remove_duplicates(xs : int list) =
    if null xs
    then []
    else
        let
            val tl_ans = remove_duplicates (tl xs)
        in
            if mem(hd xs, tl_ans)
            then tl_ans
            else (hd xs)::tl_ans
        end

fun number_in_months_challenge(dates : (int * int * int) list, months : int list) =
    number_in_months(dates, remove_duplicates months)

fun dates_in_months_challenge (dates : (int * int * int) list, months : int list) =
    dates_in_months(dates, remove_duplicates months)

fun reasonable_date (date : int * int * int) =
    let
        fun get_nth (lst : int list, n : int) =
        if n=1
        then hd lst
        else get_nth(tl lst, n-1)
        val year  = #1 date
        val month = #2 date
        val day   = #3 date
        val leap  = year mod 400 = 0 orelse (year mod 4 = 0 andalso year mod 100 <> 0)
        val feb_len = if leap then 29 else 28
        val lengths = [31,feb_len,31,30,31,30,31,31,30,31,30,31]
    in
        year > 0 andalso month >= 1 andalso month <= 12
        andalso day >= 1 andalso day <= get_nth(lengths,month)
    end
```

✓ Complete