COL100 Minor Exam

Date: Monday, 28 December, 2020

Instructions:

1. The duration of the exam is 1 hour. The total marks are 25.
2. This is a closed book exam. You cannot look at your notes or browse the internet.
3. Attempt each question in a new page.
4. Almost all sub-parts of all questions can be attempted individually, i.e., while attempting a sub-part, correct answers to previous sub-parts may not be needed.
5. For fill-in-the-blanks questions please only mention what the blanks must get filled with.

1 Short Answer Type

1.1 Sort these in ascending order according to big O notation: n^{100}, 2^n, n^n, $n!$. [2 marks]

Give a brief justification, i.e. if you wrote f, g, h, k, write one sentence each on why $f = O(g)$, $g = O(h)$, and $h = O(k)$.

1.2 Consider the following code: [2 marks]

```
fun foo [] = 0
  | foo (x::xs) =
      if x mod 2 = 0
      then x + (foo xs)
      else (foo xs);
```

(a) Give the output for `foo` $[72, 8, 1, 53, 2]$.

(b) What does the function `foo` do for any list L?

1.3 Consider the following definition of a function $f : \mathbb{N} \to \mathbb{N}$: [2 marks]

$$f_{helper}(i, n) = \begin{cases} g(i) & \text{if } i^2 \geq n, \\ g(i) \times f_{helper}(i + 1, n) & \text{otherwise.} \end{cases}$$

$$f(n) = f_{helper}(0, n) \times h(n).$$

Assume the time complexity of evaluating $g(n)$ is $O(n^2)$ and that of $h(n)$ is $O(n)$. Give the time complexity of $f(n)$ in big O notation. Justify your answer in brief.

2 Tail Recursion, Invariants and Time Complexity

Consider the following two programs:

```
fun bar1 [] = []
| bar1 (x::xs) = (bar1 xs) @ [x];

fun bar2 [] L = L
| bar2 (x::xs) L = bar2 xs (x::L);
```

2.1 What do the functions `bar1` and `bar2` do? [2 marks]

2.2 Which of these is tail-recursive? Why? [1 marks]

2.3 Give an appropriate invariant for the tail recursive function among these. Use the invariant to prove the function's correctness. [3 marks]

2.4 Let us consider only the first function, `bar1`. Let $T(n)$ denote the time taken by `bar1` on input lists of size n. Write down the recurrence relation for $T(n)$, and find the time complexity in big O notation. [2 marks]

Assume that list concatenation (@) takes $O(n + m)$ time, where n and m are the sizes of the input lists.

3 Selection Sort

3.1 The following function `smallest` takes as input a list and returns a tuple containing the minimum value and the corresponding index for the value. For example, given a list $[8, 3, 5]$, the function should return $(3, 1)$ (note that list indices start from 0). Fill in the four blanks (a), (b), (c), (d) to make the code work. [2 marks]

```
fun smallest L =
  let
    fun helper [] i = raise Empty
      | helper [x] i = (x,i)
      | helper (x::xs) i =
          let val (y_val,y_ind) = helper __(a)__ __(b)__
          in
            if x < y_val
            then __(c)__
            else __(d)__
          end
  in
    helper L 0
  end;
```

3.2 The following function `del` takes as input a list `L` and an integer `ind`, and returns the list with the value at index `ind` removed. For example, the output for inputs `L` = $[4, 5, 52, 42]$ and `ind` = 2 will be $[4, 5, 42]$. Fill in the four blanks to make the code work. [2 marks]

```
fun del L ind =
  let
    fun helper (x::xs) i n =
          if i = n
          then __(a)__
          else x::(helper __(b)__ __(c)__ __(d)__)
      | helper [] i n = raise Empty
  in
    helper L 0 ind
  end;
```

3.3 Now we shall use the two functions we have defined in the previous parts to code a new sorting algorithm, called selection sort. The following is the algorithm for selection sort:

- If the list is empty, then return empty list
- If the list is nonempty,
    - (i) Find the smallest number in list
    - (ii) Delete this number from the list
    - (iii) Sort the remaining list
    - (iv) Cons the smallest number from (i) to the front of the sorted list obtained in (iii).

Complete the SML code: [2 marks]

```
fun selectionSort [] = []
  | selectionSort L =
      let
        val (k, ind) = __(a)__
      in
        __(b)__
      end;
```

4 Recursive Relations

Given a stick of length n and a function $p : \mathbb{N} \to \mathbb{N}$ which gives prices of any stick of length i (where $1 \leq i \leq n$), find the optimal way to cut the stick into smaller sticks in order to sell them at maximum overall profit.

For example, suppose the stick length is $n = 4$, and the prices are given as

$$p(1) = 1, \quad p(2) = 5, \quad p(3) = 8, \quad p(4) = 9,$$

i.e. the price of a stick of length 1 is 1, the price of a stick of length 2 is 5, and so on. Then the best solution is 10: cut the stick into two pieces of length 2 each, to earn a profit of $p(2) + p(2) = 10$.

Cut	Profit
4	9
1, 3	1 + 8 = 9
2, 2	**5 + 5 = 10**
3, 1	8 + 1 = 9
1, 1, 2	1 + 1 + 5 = 7
1, 2, 1	1 + 5 + 1 = 7
2, 1, 1	5 + 1 + 1 = 7
1, 1, 1, 1	1 + 1 + 1 + 1 = 4

Let $stickCut(n, p)$ denote the function which gives the maximum overall profit given a stick of length n and prices p.

4.1 Find the recursive relation for $stickCut(n, p)$ (in mathematical form). [2 marks]

4.2 Prove its correctness. [3 marks]

Hint: Think about how you can get the best profit for a stick if you know the best profit for sticks of smaller lengths.