Homework 12: Tail Recursion and Analysis
Due: 11:59 PM, Dec 4, 2018

Contents

1 Tail Recursion
   1.1 Harmonic Numbers .................................................. 2
   1.2 Sublist ................................................................. 2

2 Writing Recurrence Relations .......................................... 3

3 Big-O and Big-Omega ...................................................... 4

Objectives

By the end of this homework you will:

1. written tail recursive procedures
2. have written recurrence relations for various procedures
3. determine big-O class based on recurrence relations

How to Hand In

For this (and all) homework assignments, you should hand in answers for all the non-practice questions. For this homework specifically, this entails answering the Tail Recursion, Writing Recurrence Relations, and Big-O and Big-Omega questions.

In order to hand in your solutions to these problems, they must be stored in appropriately-named files. In particular, each should be named for the corresponding problem:

- harmonic.ml
- sublist.ml
- writing-recurrence-relations.txt
- big-o.txt
- big-omega.txt

For this assignment, all files you turn in that contain code must be OCaml files, so they must end with extension .ml. For all coding problems in this homework, you should follow the OCaml design recipe outlined previously.
For this and every future assignment, you should include the `CS17setup.ml` file with your hand-in. You should also have a `README.txt` file whose first line contains only your Banner ID, and optionally with a message to the person grading explaining anything peculiar about the handin. For example:

```
README.txt:
klein@brown.edu
There's nothing to say except that I'm turning in these files plus this README
the way the instructions say that I should.
```

To hand in your solutions to these problems, you must upload them to Gradescope. Do not zip or compress them. If you re-submit your homework, you must re-submit all files. If you choose to also store these files on department machines, all your solution files should reside in your `~/course/cs0170/homeworks/hw12` directory.

### Problems

#### 1 Tail Recursion

A procedure is **tail recursive** if the output to the original input is the same as the output to the recursive call. Said another way, the term tail recursion refers to the concept that the output of a procedure is just the recursive call, without any additional computation.

Consider the following recursive procedure:

```ocaml
let rec factorial : int -> int = function
  | 0 -> 1
  | n -> n * factorial (n-1);;
```

This procedure is **not** tail recursive because in the recursive case, we multiply the recursive call by `n`. However, we can re-write this procedure to be tail recursive.

```ocaml
let rec factorial_helper : int * int -> int = function
  | 0, acc -> acc
  | n, acc -> (factorial_helper ((n-1), (acc * n)));;

let rec factorial : int -> int = function
  factorial_helper (n, 1);;
```

Notice now, the final output of our program is identical to the recursive output, with no additional computation done. To accomplish this, we kept an argument that accumulated the answer - we call this an accumulator.

#### 1.1 Harmonic Numbers

**Task:** Write a procedure `harmonic` that takes in a number `n` and produces the `n`\textsuperscript{th} harmonic number. To earn credit, you must write this procedure using tail recursion.
Hint: Let $H(n)$ be a function that returns the $n^{th}$ harmonic number. Recall that $H(1) = 1$, $H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$.

1.2 Sublist

Task: Write a procedure `sublist` that takes in a list of any type, an integer $i$, and an integer $j$ and returns the list of elements within range $[i, j)$. To earn credit, you must write this procedure using tail recursion.

Note: `sublist` should be 0-indexed. That is, the 0$^{th}$ elements refers to the first element in the list.

Here are some examples:

```plaintext
sublist (["a" ; "b" ; "c" ; "d" ; "e"], 0, 3) => ["a" ; "b" ; "c"]
sublist (["a" ; "b" ; "c" ; "d" ; "e"], 2, 4) => ["c" ; "d"]
sublist (["a" ; "b" ; "c" ; "d" ; "e"], 2, 5) => ["c" ; "d", "e"]
```

2 Writing Recurrence Relations

Task: In a text file calling `writing-recurrence-relations.txt`, write a recurrence relation for each of the following procedures. You may use variables to represent constant number of operations.

Here’s an example to get you started.

```plaintext
let rec containsP : 'a list * 'a -> bool = function
    [], _ -> false
  | hd :: tl, item -> hd = item || (containsP (tl, item));;
```

Solution: Let $f(n)$ be the maximum number of elementary operations involved in evaluating `containsP` (alod, item) where $n$ is the number of items in the list.

\[
\begin{align*}
    f(0) &= a \quad \text{(1)} \\
    f(n) &\leq b + f(n-1) \quad \text{for } n > n_0 \quad \text{(2)}
\end{align*}
\]

1. Write a recurrence relation for `fun_math` which takes in three integers, $x$, $y$, and $z$ and produces $(x + y + z)^3$.

```plaintext
let fun_math : (int * int * int) -> int = function
    x, y, z -> (x + y + z) * (x + y + z) * (x + y + z);;
```
2. Write a recurrence relation for `make_pairs` which takes in a list of \( n \) items and produces all possible pairs of items.

```ocaml
let rec make_pairs : 'a list -> ('a * 'a) list = function
  | [] -> []
  | hd :: tl -> (List.map (function x -> (x, hd)) tl) @ (make_pairs tl);
```

3. Write a recurrence relation for `fill_bag` which takes in a tuple of two things:
   - a list of size \( n \) of tuples where the first item in the tuple represents a weight, and the second item represents a value.
   - a maximum weight a bag can hold

and returns the maximum value achieved by putting items in the bag without exceeding the given maximum weight a bag can hold.

```ocaml
let rec fill_bag : ((int * int) list) * int -> int = function
  | [], _ -> 0
  | _, 0 -> 0
  | (w, v) :: tl, max_weight ->
    if w > max_weight
    then fill_bag (tl, max_weight)
    else max (fill_bag (tl, max_weight)) (v + (fill_bag (tl, max_weight - w)));
```

3 Big-O and Big-Omega

**Task:** Copy and paste the following table into a text file called `big-o.txt`. Each row in the table represents one of the respective problems below. For each row, identify if the problem is in the respective Big-O class by replacing the “.” with “Y” indicating the recurrence is in the Big-O class, or with “N” indicating the recurrence is not in the Big-O class.

<table>
<thead>
<tr>
<th></th>
<th>( O(1) )</th>
<th>( O(\log n) )</th>
<th>( O(n) )</th>
<th>( O(n \log n) )</th>
<th>( O(n^2) )</th>
<th>( O(2^n) )</th>
<th>( O(3^n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#3</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#4</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#5</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#6</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

1. \( T(n) = 2T(n - 1) + a \), where \( T(0) = b \).
2. \( T(n) = T(n - 2) + an \), where \( T(0) = b \).
3. \( T(n) = a + b + c \), where \( T(0) = d \).
4. \( T(n) = T\left(\frac{n}{2}\right) + An \), where \( T(0) = b \).

5. \( T(n) = 2T(n-1) + T(n-2) + a \), where \( T(0) = b \).

6. \( T(n) = T\left(\frac{n}{4}\right) + A \), where \( T(0) = b \).

**Task:** Repeat the above task, but instead, for each row, identify if the problem is in the respective Big-Omega class by replacing the “.” with “Y” indicating the recurrence is in the Big-Omega class, or with “N” indicating the recurrence is not in the Big-Omega class.

We’ve provided the table with Big-Omega below. Put this table in a file called `big-omega.txt`.

<table>
<thead>
<tr>
<th></th>
<th>( \Omega(1) )</th>
<th>( \Omega(\log n) )</th>
<th>( \Omega(n) )</th>
<th>( \Omega(n \log n) )</th>
<th>( \Omega(n^2) )</th>
<th>( \Omega(2^n) )</th>
<th>( \Omega(3^n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#3</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#4</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#5</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>#6</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

**Note:** We’ve also provided a copy of the table located in `/course/cs0170/src/hw12/big-o.txt` and `/course/cs0170/src/hw12/big-omega.txt`.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback).