You're a lumberjack, and it's OK.

Lucas Larsson

Spring Term 2022

The recursive solution

Below is the code completed from the skeleton code in the instructions, not much to explain the code does exactly what is supposed to do from the explanation in the instructions. see example below.

```
def split(seq) do split(seq, 0, [], []) end
def split([], l, left, right) do
    [{left, right, l}]
    end
def split([s|rest], l, left, right) do
    split(rest, l+s, [s|left], right) ++ split(rest, l+s, left, [s|right])
end
```

Output from calling the `split/1` function, with the list `[1,2,3]`, notice that the generated output is 2^n tuples, where `n` is the length of the list.

```
iex(1)> Lumberjack.split([1,2,3])
   [{[3, 2, 1], [], 6}, {[2, 1], [3], 6}, {[3, 1], [2], 6},
    {[1], [3, 2], 6}, {[3, 2], [1], 6}, {[2], [3, 1], 6},
    {[3], [2, 1], 6}, {[], [3, 2, 1], 6}]
```

Next step was the cost function, i applied the changes needed in order for it to calculate the cost, i am not going to include the code, because we basically got the solution in the instructions.

I run `Elixir` in `Visual Studio Code` so i don't need to recompile every time i modify the program, I can simply press one button, this is relevant because after running the cost function with the lists from the instructions in the `VS Code` debugger i got the following error. see below

```
Lumberjack.cost([1,2,3,4,5,6,7,8,9,10])
:elixir_ls_expression_timeout
```

The debugger took so much time that the editor would shut the process down because it thought it was stuck in an infinite loop.

The solution

I used the mentioned optimization mentioned to get a better run time, such as sorting the list before sending it to the cost function, this showed a bit of better run time, but nothing consistent, maybe only because i am running it from a laptop, and only for small n to get a definitive answer i needed to run it on a grater value, and that was still impossible, and just not realistic. Another optimization i did was to turn the list to binary before sending it from the memory module, using Erlang's library `:binary.list_to_bin`

after doing the above step I realized that the whole assignment including the bench marks are included in the instructions. So I am not completely sure about what to write here I am not complaining, just mentioning it.

After running the bench mark I got $52 mins$ on the 20 so pretty good estimate there as well.

Reflection

I followed the instructions and do feel like I have learned a lot about dynamic programming power, but another thing that i picked up is the importance of the algorithm used, thinking about a simpler solution before committing to a solution and optimizing it.

A simpler solution in my opinion for this problem is to cut the lumber in the biggest wanted piece first and continue on in descending order. For an example if you want to cut a lumber into pieces of `[3,2,1,4]` just rearing the list in descending order `[4,3,2,1]` will give the least expansive cost. please let me know if I am wrong.