Task 0

Eduardas Acuta

October 2024

1. let denote complexity as T, we have following recursive relation

$$T(0) = 1$$

$$T(n) = T(n-1) + T(n-1) = 2T(n-1)$$

But this is just definition of exponent base 2, so

$$T(n) = 2^n = O(2^n)$$

2.we could do it better. $f(n) = f(n-1) + f(n-1) = 2f(n-1)$ And the recursive relation of complexity becomes

$$T(0) = 1$$

$$T(n) = 1 + T(n-1)$$

so what we can easily get from induction is following

$$T(n) = n + 1 = O(n)$$

. But we can do better, because this is not tail recursive. If we would rewrite as cycle in java it would be

```java
int f(int n) {
    int i = 0;
    int res = 1
    while(i != n) {
        res = res + res
        i = i + 1
    }
    return res
}
```

since every while loop can be turned into tail recurtion we will do that part in braces we could introduce as function *step*:(Int, Int) $\rightarrow$ (Int, Int). $step(x, y) = (x+x, y+1)$, and then we recursively call while, so in the end we get. $while$:(Int, Int) $\rightarrow$ (Int, Int). $while(res, n) = res$, and $while(res, i) = step(res, i)$. Combing previous results we get that we can optimize f in following manner. $f'(n, res, n) = res$, or $f'(n, res, i) = f'(n, res + res, i + 2)$, and then $f(n) = f'(n, 1, 0)$

```
def fhelper(n : Int, res : Int, i : Int) = i match {
    case n => res
    case i => fhelper(n, res + res, i+1)
}
def f(n) = fhelper(n, 1, 0)
```

and it is tail recursive