Trabajo Práctico 2

Gabriel Eberlein, Juan Segundo Valero

b)

Operación	W	S
mapS f s	$O(\sum_{i=0}^{\lvert s\rvert-1} W(f\ i))$	$O(\max_{i=0}^{\lvert s\rvert-1} S(f\ i))$
reduceS $\oplus$ e s	$O(\lvert s\rvert + \sum_{(x,y)\in O_r(\oplus,e,s)} x \oplus y)$	$O(lg\lvert s\rvert \max_{(x,y)\in O_r(\oplus,e,s)} x \oplus y)$
scanS $\oplus$ e s	$O(\lvert s\rvert + \sum_{(x,y)\in O_r(\oplus,e,s)} x \oplus y)$	$O(lg\lvert s\rvert \max_{(x,y)\in O_r(\oplus,e,s)} x \oplus y)$

d)

Esta función hace una simple operación con f en dos elementos contiguos del array entonces,

$$S(contraerAux\ \oplus\ i\ s) = S(s[2i]\ \oplus\ s[2i+1])$$

```
contraerAux :: (a -> a -> a) -> Int -> A.Arr a -> a
contraerAux f i xs | i == (lengthS xs - 1) = (xs A.! i)
                   | otherwise = (f (xs A.! i) (xs A.! (i + 1)))
```

Esta función hace un tabulate con la función contraerAux sobre un array, reduciendo su tamaño a la mitad (ceiling de esto). Siguiendo entonces la profundidad del tabulate tenemos,

$$S(contraer\ \oplus\ s) = O(\max_{i=0}^{\lfloor \lvert s\rvert/2\rfloor -1} S(s[2i]\ \oplus\ s[2i+1]))$$

```
contraer :: (a -> a -> a) -> A.Arr a -> A.Arr a
contraer f xs =
    tabulateS (\x -> contraerAux f (2*x) xs) (ceiling(fromIntegral (A.length xs) / 2))
```

Finalmente reduceArr toma un array, lo contrae en cada recursión y recursa sobre si misma con el array contraído hasta tener un solo elemento. Al reducirse a la mitad el array cada vez que se recursa, va a haber log(n) recursiones. Queda entonces la profundidad,

$$S(reduceArr\ \oplus\ e\ s) = O(lg\lvert s\rvert \max_{(x,y)\in O_r(\oplus,e,s)} S(x\ \oplus y))$$

Acotamos por arriba la complejidad poniendo la máxima profundidad posible de la operación.

```
reduceArr :: (a -> a -> a) -> a -> A.Arr a -> a
reduceArr f b xs | (lengthS xs) == 0 = b
         | (lengthS xs) == 1 = f b (nthS xs 0)
         | otherwise = let ys = (contraer f xs) in reduceArr f b ys
```