COMP50001: Algorithm Design & Analysis

Sheet 4 (Week 5)

Exercise 4.1

```
data Deque a = Deque [a] [a]

instance List Deque where
  toList :: Deque a → [a]
  toList (Deque xs sy) = xs ++ reverse sy

  fromList xs = Deque ys (reverse zs)
    where (ys, zs) = splitAt (length xs `div` 2) xs

  tail :: Deque a → Deque a
  tail (Deque []     []) = error "tail: empty list"
  tail (Deque []     sy) = empty
  tail (Deque [x]    sy) = fromList (reverse sy)
  tail (Deque (x:xs) sy) = Deque xs sy

  cons :: a → Deque a → Deque a
  cons x (Deque xs []) = Deque [x] xs
  cons x (Deque xs sy) = Deque (x : xs) sy

  snoc :: Deque a → a → Deque a
  snoc (Deque [] sv) x = Deque sv [x]
  snoc (Deque us sv) x = Deque us (x : sv)
```

1. Give the time complexity of the following *reverse* in terms of the length of a deque:

$$\mathit{reverse} :: \mathit{Deque}\ a \to \mathit{Deque}\ a$$
$$\mathit{reverse} = \mathit{fromList} \circ \mathit{reverse} \circ \mathit{toList}$$

2. Implement a *reverse* for deques that runs in $O(1)$ time.

Exercise 4.2

Supposing $xs_0 :: \mathit{Deque}\ a$ is the empty deque, show that the amortised complexity of each operation in the following sequence is $O(1)$:

$$xs_0 \overset{op_0}{\leadsto} xs_1 \overset{op_1}{\leadsto} xs_2 \overset{op_2}{\leadsto} \ldots \overset{op_{n-1}}{\leadsto} xs_n$$

where each $op_i \in \{\mathit{tail},\ \mathit{snoc},\ \mathit{cons}\}$.

Exercise 4.3

```
snoc' :: Deque a → a → Deque a
snoc' (Deque us sv) v = Deque us (v : sv)

tail' :: Deque a → Deque a
tail' (Deque [] []) = error "tail: empty list"
tail' (Deque [] sv) = Deque (tail (reverse sv)) []
tail' (Deque us sv) = Deque (tail us) sv

init' :: Deque a → Deque a
init' (Deque [] []) = error "init: empty list"
init' (Deque us []) = Deque [] (tail (reverse us))
init' (Deque us sv) = Deque us (tail sv)
```

Suppose no invariants are imposed on *Deque* and *snoc* and *tail* are alternatively defined as *snoc′* and *tail′*.

1. Prove that the amortised complexity of each operation in a sequence of *snoc′* and *tail′* is still $O(1)$.

2. Suppose that the sequence additionally contains operation *init′*. Determine whether the amortised complexity is still $O(1)$.

Exercise 4.4

Consider the following alternative representation of *Deque*:

$$\textbf{data}\ \mathit{Deque}'\ a = \mathit{Deque}'\ \mathit{Int}\ [a]\ [a]$$

with an invariant $n = \mathit{length}\ us + \mathit{length}\ sv$ for any $\mathit{Deque}'\ n\ us\ sv$. Define *cons* and $+\!\!+$ for this representation as follows:

```
instance List Deque' where
  toList :: Deque' a → [a]
  toList (Deque' n us sv) = us ++ reverse sv

  fromList :: [a] → Deque' a
  fromList xs = Deque' n us sv
    where n        = length xs
          (us, vs) = splitAt (n `div` 2) xs
          sv       = reverse vs
```

```
cons :: a → Deque' a → Deque' a
cons u (Deque' n us sv) = Deque' (n + 1) (u : us) sv

(++) :: Deque' a → Deque' a → Deque' a
Deque' n us sv ++ Deque' n' us' sv'
  | n < n'    = Deque' (n + n') (us ++ reverse sv ++ us') sv'
  | otherwise = Deque' (n + n') us (sv' ++ reverse us' ++ sv)
```

1. Give the worst-case complexity of $xs +\!\!+ ys$ in terms of *length xs* and *length ys*.

2. Consider a sequence of operations creating and manipulating multiple deques

$$D_0 \overset{op_0}{\rightsquigarrow} D_1 \overset{op_1}{\rightsquigarrow} D_2 \overset{op_2}{\rightsquigarrow} \ldots D_{n-1} \overset{op_{n-1}}{\rightsquigarrow} D_n$$

where each D_i is a *multiset* of deques and $D_0 = \emptyset$. Each op_i is only one of the following forms:

(a) $xs_i = empty$, and in this case

$$D_{i+1} = D_i \cup \{xs_i\},$$

(b) $xs_i = cons\ x\ xs_j$ where $xs_j \in D_i$, and in this case

$$D_{i+1} = (D_i \setminus \{xs_j\}) \cup \{xs_i\},$$

(c) $xs_i = xs_j +\!\!+\ xs_k$ where $xs_j, xs_k \in D_i$ and $j \neq k$, and in this case

$$D_{i+1} = (D_i \setminus \{xs_j,\ xs_k\}) \cup \{xs_i\}.$$

For case (c) $xs_i = xs_j +\!\!+\ xs_k$, if xs_{small} is the one in xs_j and xs_k with the smaller length, the elements in xs_{small} is said to be *merged into a larger deque*. Explain why every element can only be merged into a larger deque at most $\lceil \log_2 n \rceil$ times, where n is the length of the sequence of operations.

3. Prove that each operation has amortised complexity $O(\log_2 n)$ with the following size function:

$$S(D) = sum\ [length\ xs \times log_2\ (n\ /\ (length\ xs)) \mid xs \leftarrow D, length\ xs > 0]$$

and explain the intuition of this size function. (The cost incurred by operations $\cup$ and $\setminus$ on the multiset does not need to be considered in the analysis.)

Exercise 4.5

Define $dec :: Binary \rightarrow Binary$ that decrements a binary number discussed in the lecture and show that the amortised complexity of repeated applications of dec is $O(1)$. Determine if the amortised complexity of each operation in a sequence op_i where $0 \leqslant i < n$ and $op_i \in \{inc,\ dec\}$ is still $O(1)$.

```
type Binary = [Digit]
data Digit = O | I

inc :: Binary -> Binary
inc []       = [I]
inc (O : bs) = I : bs
inc (I : bs) = O : (inc bs)
```

Exercise 4.6

Given is the data type *Tree* which is an instance of *List*. The tree $t :: Tree\ a$ is *balanced* iff $t = Tip$, $t = Leaf\ a$, or $t = Node\ n\ l\ r$ with balanced subtrees l and r such that $size\ l = size\ r$.

Show that $(!!) :: Tree\ a \rightarrow Int \rightarrow a$ takes $O(\log_2 n)$ time for balanced binary trees using a recurrence relation, where n is the number of elements in the tree.

```
data Tree a = Tip
            | Leaf a
            | Node Int (Tree a) (Tree a)

size (Tip)        = 0
size (Leaf _)     = 1
size (Node n _ _) = n
```

Exercise 4.7

Consider the definition of *RAList a*.

1. Letting $head = (!!0)$, give the best-case and worst-case time complexities of *head xs* where $xs :: RAList\ a$.

2. Implement $tail :: RAList\ a \to RAList\ a$ such that the amortised complexity of a sequence of *tail* is $O(1)$.

3. Determine if the amortised complexity of a sequence of operations where each operation is either *tail* or *cons* is still $O(1)$. (Hint: compare with *inc* and *dec* for binary numbers.)

```
newtype RAList a = RAList [Tree a]
instance List RAList where
  fromList :: [a] → RAList a
  fromList xs = foldr cons empty xs
  toList :: RAList a → [a]
  toList (RAList ts) = (concat ∘ map toList) ts
  (!!) :: RAList a → Int → a
  RAList (t : ts) !! k
    | k < size t = t !! k
    | otherwise = RAList ts !! (k − size t)
```

Solutions to the Exercises

Solution 4.1

1. The functions *fromList*, *toList* and *reverse* for $[a]$ all run in time proportional to the length of the input, so this *reverse* for deques also runs in $O(\mathit{length}\ xs)$ time where $xs :: \mathit{Deque}\ a$ is the input.
2. Because *Deque* is symmetric, reversing it can simply done by

   $$\begin{aligned}&\mathit{reverse} :: \mathit{Deque}\ a \to \mathit{Deque}\ a\\ &\mathit{reverse}\ (\mathit{Deque}\ xs\ ys) = \mathit{Deque}\ ys\ xs\end{aligned}$$

   This clearly only needs $O(1)$ time.

Solution 4.2

We set $A_{op}(xs) = 2$ for each *op* and use the same cost function C and size function S as in the lecture:

$$\begin{aligned}C_{cons}(xs) = 1 \qquad C_{snoc}(xs) = 1\\ C_{tail}(\mathit{Deque}\ xs\ sy) = \mathbf{if}\ \mathit{length}\ xs > 1\ \mathbf{then}\ 1\ \mathbf{else}\ \mathit{length}\ sy\\ S(\mathit{Deque}\ xs\ sy) = |\mathit{length}\ xs - \mathit{length}\ sy|\end{aligned}$$

so it remains to show

$$C_{op_i}(xs_i) \leqslant A_{op_i}(xs_i) + S(xs_i) - S(xs_{i+1})$$

for $op_i \in \{cons,\ snoc,\ tail\}$. The case for *tail* can be found in the lecture notes. For $op_i = snoc$, we have the definition

$$\begin{aligned}&\mathit{snoc} :: \mathit{Deque}\ a \to a \to \mathit{Deque}\ a\\ &\mathit{snoc}\ (\mathit{Deque}\ [\,]\ sv)\ x\ = \mathit{Deque}\ sv\ [x]\\ &\mathit{snoc}\ (\mathit{Deque}\ us\ sv)\ x = \mathit{Deque}\ us\ (x : sv)\end{aligned}$$

and $C_{snoc}(xs) = 1$ and $A_{snoc}(xs) = 2$. If xs_i matches the first pattern $\mathit{Deque}\ [\,]\ sv$ of *snoc*,

$$\begin{aligned}&A_{snoc}(xs_i) + S(xs_i) - S(xs_{i+1})\\ =\ & 2 - S(\mathit{Deque}\ sv\ [x]) + S(\mathit{Deque}\ [\,]\ sv)\\ =\ & 2 - |\mathit{length}\ sv - 1| + |\mathit{length}\ sv|\\ \geqslant\ & \quad \{|a-b| \leqslant |a| + |b|\}\\ & 2 - 1 - |\mathit{length}\ sv| + |\mathit{length}\ sv|\\ =\ & 2 - 1 = 1 = C_{snoc}(xs_i)\end{aligned}$$

If xs_i matches the second pattern $\mathit{Deque}\ us\ sv$ of *snoc*,

$$\begin{aligned}&A_{snoc}(xs_i) + S(xs_i) - S(xs_{i+1})\\ =\ & 2 - S(\mathit{Deque}\ us\ (x : sv)) + S(\mathit{Deque}\ us\ sv)\\ =\ & 2 - |\mathit{length}\ sv + 1 - \mathit{length}\ us| + |\mathit{length}\ sv - \mathit{length}\ us|\\ \geqslant\ & \quad \{|a+b| \leqslant |a| + |b|\}\\ & 2 - 1 - |\mathit{length}\ sv - \mathit{length}\ us| + |\mathit{length}\ sv - \mathit{length}\ us|\\ =\ & 2 - 1 = 1 = C_{snoc}(xs_i)\end{aligned}$$

The case for *cons* is completely symmetric.

Solution 4.3

1. The cost function in this situation is

   $C_{tail'}(Deque\ us\ sv) = \textbf{if}\ null\ us\ \textbf{then}\ length\ sv\ \textbf{else}\ 1 \qquad C_{snoc'}(xs) = 1$

   Define function $A_{op}(xs) = 2$ for any op and define size function $S(Deque\ us\ sv) = length\ sv$. To show that A_{op} is the amortised cost of each operation, it is sufficient to show

   $$C_{op}(xs) \leqslant A_{op}(xs) + S(xs) - S(op\ xs)$$

   for any $op \in \{tail',\ snoc'\}$ and $xs :: Deque\ a$.

   (a) If $op = tail'$ and $xs = Deque\ [\,]\ sv$, then

   $$\begin{aligned} C_{tail'}(xs) = length\ sv &\leqslant 2 + length\ sv \\ &= A_{tail'}(xs) + S(xs) - S(tail'\ xs) \end{aligned}$$

   (b) If $op = tail'$ and $xs = Deque\ (u : us)\ sv$, then

   $$\begin{aligned} C_{tail'}(xs) = 1 &\leqslant 2 + length\ sv - length\ sv \\ &= A_{tail'}(xs) + S(xs) - S(tail'\ xs) \end{aligned}$$

   (c) If $op = snoc'$ and $xs = Deque\ us\ sv$, then

   $$\begin{aligned} C_{snoc'}(xs) = 1 &\leqslant 2 + length\ sv - (lengthsv + 1) \\ &= A_{snoc'}(xs) + S(xs) - S(snoc'\ xs) \end{aligned}$$

2. No, the amortised complexity is no longer $O(1)$ in this case. Consider the following sequence of operations of length $4n$:

   $$snoc\ 1,\ snoc\ 2,\ \ldots, snoc\ (2*n), \overbrace{tail',\ init',\ tail',\ init',\ \ldots}^{2n \text{ operations}}$$

   After the first $2n$ $snoc'$ operations, the deque is $Deque\ \ [\,]\ \ [2*n\,..\,1]$, and the $(2n+k)$-th operation $(1 \leqslant k \leqslant 2n)$, which is either a $tail'$ or an $init'$, triggers a complete *reverse* of cost $2n - k + 1$. Therefore the total cost of the sequence is $2n^2 + 3n\ \in\ \Theta(n^2)$ and the amortised complexity cannot be $O(1)$, because it would imply that the total cost is in $O(n)$.

   Note that the *tail* and *init* discussed in the lecture do not suffer from this problem. Why?

Solution 4.4

1. Because $reverse\ \ us$ and $us +\!\!+\ sv$ take $O(length\ \ us)$ time for any $us, sv :: [a]$, $xs +\!\!+\ ys$ for $xs, ys :: Deque'\ a$ runs in time

   $$O(length\ xs \sqcap length\ ys)$$

2. When an element x in xs_{small} is merged into a larger deque, it belongs to a deque at least *twice* as large as xs_{small}. Thus if an element is merged into a larger deque more than $\lceil \log_2 n \rceil$ times, it must be in a deque containing more than n elements, which is impossible because there are only n operations in the process, and each operation can create at most one element in the collection of deques.

3. The intuition is that because this version of $xs +\!\!+\, ys$ for deques takes $O(\mathit{length}\ xs \sqcap \mathit{length}\ ys)$ time, we can think that each element in the smaller deque

$$xs_{small} = \textbf{if } \mathit{length}\ xs < \mathit{length}\ ys \textbf{ then } xs \textbf{ else } ys$$

is response for $O(1)$ cost for this $+\!\!+$ operation. Furthermore, an element pays this cost at most $\log_2 n$ times because it can only be merged into a larger deque at most $\log_2 n$ times. As there can only be at most n elements in the whole collection of deques, the total cost of all operations is $O(n \log_2 n)$ and amortised $O(\log_2 n)$ for each operation.

This argument can be formally proved by defining

$$C_{empty}(D) = 1 \qquad C_{cons\ a\ xs_j}(D) = 1$$

$$C_{xs_j +\!\!+ xs_k}(D) = \mathit{length}\ xs_j \sqcap \mathit{length}\ xs_k$$

and

$$A_{empty}(D) = 1 \qquad A_{cons\ a\ xs_j}(D) = \log_2 n + 1$$

$$C_{xs_j +\!\!+ xs_k}(D) = \log_2 n$$

and for any set D of deques,

$$S(D) = \mathit{sum}\ [\mathit{length}\ xs \times \mathit{log}_2\ (n\ /\ (\mathit{length}\ xs)) \mid xs \leftarrow D, \mathit{length}\ xs > 0]$$

It remains to show

$$C_{op_i}(D_i) \leqslant A_{op_i}(D_i) + S(D_i) - S(D_{i+1}) \tag{1}$$

Logarithm satisfies the following identities:

$$\log(a \times b) = \log a + \log b$$
$$\log(\frac{a}{b}) = \log a - \log b$$
$$a \log b = \log(b^a)$$

(a) If $op_i = \mathit{empty}$, Equation 1 clearly holds because $S(D_{i+1}) = S(D_i)$.

(b) If $op_i = \mathit{cons}\ a\ xs_j$, denoting $|xs| = \mathit{length}\ xs$ for any $xs :: \mathit{Deque}\ a$, if xs_j is not an empty deque,

$$\begin{aligned}
& A_{cons\ a\ xs_j}(D_i) + S(D_i) - S(D_{i+1}) \\
= \ & \log_2 n + 1 + |xs_j| \log_2 \frac{n}{|xs_j|} - (|xs_j| + 1) \log_2 \frac{n}{|xs_j| + 1} \\
\geqslant \ & \log_2 n + 1 - \log_2 \frac{n}{|xs_j| + 1} \\
= \ & \log_2(|xs_j| + 1) + 1 \\
\geqslant \ & 1 = C_{cons\ a\ xs_j}(D_i)
\end{aligned}$$

If xs_j is an empty deque,

$$\begin{aligned}
& A_{cons\ a\ xs_j}(D_i) + S(D_i) - S(D_{i+1}) \\
= \ & \log_2 n + 1 - \log_2 n \geqslant 1 = C_{cons\ a\ xs_j}(D_i)
\end{aligned}$$

(c) If $op_i = xs_j +\!\!+ xs_k$, assuming $0 < |xs_j| < |xs_k|$,

$$\begin{aligned}
& A_{xs_j +\!+ xs_k}(D_i) + S(D_i) - S(D_{i+1}) \\
= \; & \log_2 n + |xs_j| \log_2 \frac{n}{|xs_j|} + |xs_k| \log_2 \frac{n}{|xs_k|} - (|xs_j| + |xs_k|) \log_2 \frac{n}{|xs_j| + |xs_k|} \\
= \; & \log_2 n - |xs_j| \log_2(|xs_j|) - |xs_k| \log_2(|xs_k|) + (|xs_j| + |xs_k|) \log_2(|xs_j| + |xs_k|) \\
\geqslant \; & \log_2 n + |xs_j|(\log_2(|xs_j| + |xs_k|) - \log_2(|xs_j|)) \\
\geqslant \; & \quad \{|xs_j| < |xs_k|\} \\
& \log_2 n + |xs_j|(\log_2(2|xs_j|) - \log_2(|xs_j|)) \\
= \; & \log_2 n + xs_j \geqslant xs_j = C_{xs_j +\!+ xs_k}(D_i)
\end{aligned}$$

The case $0 < |xs_k| \leqslant |xs_j|$ is symmetric, and the case

$$min\ |xs_j|\ |xs_k| = 0$$

is also straightforward.

The intuition for the size function is that it measures how many times each element can be merged into a larger deque in the rest of the process.

Solution 4.5

The function *dec* is symmetric to *inc* discussed in the lecture:

$$\begin{aligned}
& inc :: Binary \to Binary \\
& inc\ [\,] \qquad = [I] \\
& inc\ (O : bs) = I\ : bs \\
& inc\ (I : bs) \ \ = O : (inc\ bs)
\end{aligned}$$

$$\begin{aligned}
& dec :: Binary \to Binary \\
& dec\ [I] \qquad = [\,] \\
& dec\ (I : bs) \ \ = O : bs \\
& dec\ (O : bs) = I : dec\ bs
\end{aligned}$$

1. Similarly to the analysis of *inc* in the lecture, we define

$$C_{dec}(bs) = t + 1 \ \textbf{where}\ t = length\ (takeWhile\ (\equiv O)\ bs)$$

and for the amortised cost, we define

$$A_{dec}(bs) = 2$$

The size function is then

$$S_{dec}(bs) = b \ \textbf{where}\ b = length\ (filter\ (\equiv O)\ bs)$$

then for any $bs :: Binary$ and $bs' = dec\ bs$, the following holds:

$$\begin{aligned}
& C_{dec}(bs) \leqslant A_{dec}(bs) + S_{dec}(bs) - S_{dec}(bs') \\
\iff & \\
& t + 1 \leqslant 2 + b - b' \ \textbf{where}\ b' = b - t + 1 \\
\iff & \\
& t + 1 \leqslant 2 + b - (b - t + 1) \\
\iff & \\
& t + 1 \leqslant t + 1
\end{aligned}$$

2. When both *inc* and *dec* are allowed in the sequence, the amortised operation is no longer $O(1)$. Consider $bs = replicate\ \ n\ \ I$, which is the binary number contains n I's and the sequence of operations

$$inc,\ dec,\ inc,\ dec,\ inc,\ dec,\ \ldots$$

In this case, every *inc* and *dec* takes time proportional to the length of *bs*, so the amortised time complexity cannot be $O(1)$.

Solution 4.6

The time complexity of !! can be approximated by

$$T(n) = \begin{cases} 1 & n \leqslant 1 \\ 1 + T(n/2) & \text{otherwise} \end{cases}$$

To solve the recurrence relation, we calculate

$$\begin{aligned} T(n) &= 1 + T(\frac{n}{2}) \\ &= 1 + 1 + T(\frac{n}{2^2}) \\ &= 1 + 1 + 1 + T(\frac{n}{2^3}) \\ &= \ldots \\ &= k + T(\frac{n}{2^k}) \end{aligned}$$

Since $\frac{n}{2^k} \leqslant 1$ iff $k \geqslant \log n$, let $k = \lceil (log_2\ n) \rceil$, and

$$T(n) = \lceil (log_2\ n) \rceil + 1 \in \Theta(\log_2 n).$$

Obviously it helps to have a smart constructor for *Node*:

```
node :: Tree a → Tree a → Tree a
node lt rt = Node (size lt + size rt) lt rt

instance List Tree where
  toList :: Tree a → [a]
  toList (Tip)          = []
  toList (Leaf x)       = [x]
  toList (Node n lt rt) = toList lt ++ toList rt

  fromList :: [a] → Tree a
  fromList []  = Tip
  fromList [x] = Leaf x
  fromList xs  = node (fromList us) (fromList vs)
    where (us, vs) = splitAt (length xs `div` 2) xs

  Leaf x        !! _ = x
  Node n lt rt !! i
    | i < n `div` 2 = lt !! i
    | otherwise     = rt !! (i − n `div` 2)
```

Solution 4.7

1. The best case is when the first element of $xs :: RAList\ \ a$ is $Just\ \ (Leaf\ \ a)$, so $(!!0)$ is done in $O(1)$ time. The worst case is when *xs* contains exactly 2^k elements for some k, and thus every tree except the last one in *xs* is *Nothing*. In this case, $(!!0)$ takes $O(log_2(length\ xs))$ time.

2. The function *tail* for $RAList\ \ a$ corresponds to *dec* for binary numbers, and it can be defined as

```
tail :: RAList a → RAList a
tail = snd ∘ split where
  split :: RAList a → (Tree a, RAList a)
  split (RAList [t])        = (t, RAList [])
  split (RAList (Tip : ts)) = (t, RAList (t′ : ts′))
    where (Node _ t t′, RAList ts′) = split (RAList ts)
  split (RAList (t : ts))   = (t, RAList (Tip : ts))
```

It is easy to verify $split\ \ (cons_T\ \ t\ \ ts) = (t, ts)$ for the $cons_T$ discussed in the lecture. The amortised complexity of a sequence of *tail* is $O(1)$, which can be proved essentially in the same way as Exercise 4.5.

3. Similar to *dec* and *inc* of binary numbers, the amortised complexity is no longer $O(1)$ when *tail* and *cons* are used together. In particular, when $xs :: RAList\ a$ contains exactly $2^k - 1$ elements, the following sequence

$$cons\ a,\ tail,\ cons\ a,\ tail,\ \ldots$$

starting from *xs* costs $\Theta(\log_2(length\ xs))$ time per operation.