# *COMP50001: Algorithm Design & Analysis*

*Sheet 6 (Week 7)*

## *Exercise 6.1*

Notice that the definition of *montePi* only generates random $x$ and $y$ values between 0 and 1, thus only hitting points in one quadrant of the square. Explain why this measure approximates $\pi/4$.

$$
\begin{array}{l}
\mathit{montePi} :: \mathit{Double} \\
\mathit{montePi} = \mathit{loop}\ (\mathit{mkStdGen}\ 42)\ 0\ 0\ \mathbf{where} \\
\quad \mathit{loop}\ \mathit{seed}\ m\ n \\
\qquad \mid n \equiv 100000 = \\
\qquad\quad 4 * \mathit{fromIntegral}\ m\ /\ \mathit{fromIntegral}\ n \\
\qquad \mid \mathit{otherwise} \quad = \mathit{loop}\ \mathit{seed}''\ m'\ n' \\
\qquad \mathbf{where}\ n' \ = n + 1 \\
\qquad\qquad m' = \mathbf{if}\ \mathit{inside}\ (x, y)\ \mathbf{then}\ m + 1\ \mathbf{else}\ m \\
\qquad\qquad (x, \mathit{seed}')\ = \mathit{randomR}\ (0, 1)\ \mathit{seed} \\
\qquad\qquad (y, \mathit{seed}'') = \mathit{randomR}\ (0, 1)\ \mathit{seed}'
\end{array}
$$

$$
\begin{array}{l}
\mathit{inside} :: (\mathit{Double}, \mathit{Double}) \to \mathit{Bool} \\
\mathit{inside}\ (x, y) = x * x + y * y \leqslant 1
\end{array}
$$

## *Exercise 6.2*

Devise a randomized Monte Carlo algorithm to find the value of $\sqrt{2}$. It need not be efficient. *Hint: Consider the ratio of values between* 0 *and* 2 *that are less than* $\sqrt{2}$.

## *Exercise 6.3*

The $\mathit{insertBTree}'$ function does not produce correct results when the same element is inserted more than once, since it always increments the size of the tree even when no new elements are added. Discuss how the implementation can be changed without affecting asymptotic complexity.

$$
\begin{array}{l}
\mathit{insertBTree}' :: \mathit{Ord}\ a \Rightarrow a \to \mathit{RBTree}\ a \to \mathit{RBTree}\ a \\
\mathit{insertBTree}'\ x\ (\mathit{RBTree}\ \mathit{seed}\ n\ t) \\
\quad \mid p \equiv 0 \qquad = \mathit{RBTree}\ \mathit{seed}'\ (n + 1) \\
\qquad\qquad\qquad\qquad (\mathit{insertRoot}\ x\ t) \\
\quad \mid \mathit{otherwise} = \mathit{RBTree}\ \mathit{seed}'\ (n + 1) \\
\qquad\qquad\qquad\qquad (\mathit{insertBTree}\ x\ t) \\
\mathbf{where} \\
\quad (p, \mathit{seed}') = \mathit{randomR}\ (0, n)\ \mathit{seed}
\end{array}
$$

## *Exercise 6.4*

Prove that for a set of integers, each paired with a priority

$$S = \{(x_i, p_i) \mid 1 \leqslant i \leqslant n\}$$

where $x_i, p_i :: \mathit{Int}$, if $x_i \neq x_j$ and $p_i \neq p_j$ for any $i \neq j$, then there is a unique $t :: \mathit{Treap}\ \mathit{Int}$ such that $t$ satisfies the invariant of treaps and $\mathit{nodes}\ t$ contains the same set of elements as $S$, where

$$
\begin{array}{l}
\mathit{nodes} :: \mathit{Treap}\ a \to [(a, \mathit{Int})] \\
\mathit{nodes}\ \mathit{Empty} \qquad\qquad = [\,] \\
\mathit{nodes}\ (\mathit{Node}\ \mathit{lt}\ x\ p\ \mathit{rt}) = (x, p) : \mathit{nodes}\ \mathit{lt} +\!\!+ \mathit{nodes}\ \mathit{rt}
\end{array}
$$

Argue that as a consequence inserting the elements $\{(x_i, p_i)\}$ into a treap in different orders gives rise to the same treap, assuming all elements and priorities are distinct.

$$
\mathbf{data}\ \mathit{Treap}\ a = \mathit{Empty} \mid \mathit{Node}\ (\mathit{Treap}\ a)\ a\ \mathit{Int}\ (\mathit{Treap}\ a)
$$

$$
\begin{array}{l}
\mathit{insert} :: \mathit{Ord}\ a \Rightarrow a \to \mathit{Int} \to \mathit{Treap}\ a \to \mathit{Treap}\ a \\
\mathit{insert}\ x\ p\ \mathit{Empty} = \mathit{Node}\ \mathit{Empty}\ x\ p\ \mathit{Empty} \\
\mathit{insert}\ x\ p\ (\mathit{Node}\ \mathit{lt}\ y\ q\ \mathit{rt}) \\
\quad \mid x < y \ = \mathit{lnode}\ (\mathit{insert}\ x\ p\ \mathit{lt})\ y\ q\ \mathit{rt} \\
\quad \mid x \equiv y = \mathit{Node}\ \mathit{lt}\ y\ q\ \mathit{rt} \\
\quad \mid x > y \ = \mathit{rnode}\ \mathit{lt}\ y\ q\ (\mathit{insert}\ x\ p\ \mathit{rt})
\end{array}
$$

$$
\begin{array}{l}
\mathit{lnode} :: \mathit{Treap}\ a \to a \to \mathit{Int} \to \mathit{Treap}\ a \to \mathit{Treap}\ a \\
\mathit{lnode}\ \mathit{Empty}\ y\ q\ \mathit{rt} = \mathit{Node}\ \mathit{Empty}\ y\ q\ \mathit{rt} \\
\mathit{lnode}\ \mathit{lt}@(\mathit{Node}\ \mathit{llt}\ x\ p\ \mathit{lrt})\ y\ q\ \mathit{rt} \\
\quad \mid q \leqslant p \qquad = \mathit{Node}\ \mathit{lt}\ y\ q\ \mathit{rt} \\
\quad \mid \mathit{otherwise} = \mathit{Node}\ \mathit{llt}\ x\ p\ (\mathit{Node}\ \mathit{lrt}\ y\ q\ \mathit{rt})
\end{array}
$$

$$
\begin{array}{l}
\mathit{rnode} :: \mathit{Treap}\ a \to a \to \mathit{Int} \to \mathit{Treap}\ a \to \mathit{Treap}\ a \\
\mathit{rnode}\ \mathit{lt}\ x\ p\ \mathit{Empty} = \mathit{Node}\ \mathit{lt}\ x\ p\ \mathit{Empty} \\
\mathit{rnode}\ \mathit{lt}\ x\ p\ \mathit{rt}@(\mathit{Node}\ \mathit{rlt}\ y\ q\ \mathit{rrt}) \\
\quad \mid p \leqslant q \qquad = \mathit{Node}\ \mathit{lt}\ x\ p\ \mathit{rt} \\
\quad \mid \mathit{otherwise} = \mathit{Node}\ (\mathit{Node}\ \mathit{lt}\ x\ p\ \mathit{rlt})\ y\ q\ \mathit{rrt}
\end{array}
$$

## *Exercise 6.5*

Given $\mathit{lt}, \mathit{rt} :: \mathit{Treap}\ a$ such that $x \leqslant y$ for any $(x, \_)$ in $\mathit{nodes}\ \mathit{lt}$ and any $(y, \_)$ in $\mathit{nodes}\ \mathit{rt}$, prove that $\mathit{merge}\ \mathit{lt}\ \mathit{rt}$ satisfies the invariants of treaps.

$$
\begin{array}{l}
\mathit{merge} :: \mathit{Treap}\ a \to \mathit{Treap}\ a \to \mathit{Treap}\ a \\
\mathit{merge}\ \mathit{Empty}\ \mathit{rt} = \mathit{rt} \\
\mathit{merge}\ \mathit{lt}\ \mathit{Empty} = \mathit{lt} \\
\mathit{merge}\ \mathit{lt}@(\mathit{Node}\ \mathit{llt}\ x\ p\ \mathit{lrt})\ \mathit{rt}@(\mathit{Node}\ \mathit{rlt}\ y\ q\ \mathit{rrt}) \\
\quad \mid p < q \qquad = \mathit{Node}\ \mathit{llt}\ x\ p\ (\mathit{merge}\ \mathit{lrt}\ \mathit{rt}) \\
\quad \mid \mathit{otherwise} = \mathit{Node}\ (\mathit{merge}\ \mathit{lt}\ \mathit{rlt})\ y\ q\ \mathit{rrt}
\end{array}
$$

## *Exercise 6.6*

Implement a $\mathit{split} :: \mathit{Ord}\ a \Rightarrow \mathit{Treap}\ a \to a \to (\mathit{Treap}\ a, \mathit{Treap}\ a)$ such that $\mathit{split}\ t\ x$ computes $(\mathit{lt}, \mathit{rt})$ where $\mathit{lt}$ contains exactly

$$\mathit{filter}\ (\lambda(y, \_) \to y < x)\ (\mathit{nodes}\ t)$$

as nodes and $rt$ contains exactly

$$filter\ (\lambda(y, \_) \to y \geqslant x)\ (nodes\ t)$$

It should run in $O(depth\ t)$ time.

*merge* is a right-inverse of *split*:

$$uncurry\ merge\ (split\ t\ x) = t$$

$depth :: Treap\ a \to Int$
$depth\ Empty = 0$
$depth\ (Node\ lt\ x\ p\ rt) = 1 + max\ (depth\ lt)\ (depth\ rt)$

### *Exercise 6.7*

Implement insertion and deletion for treaps using only *merge* and *split*

$insert' :: Ord\ a \Rightarrow a \to Int \to Treap\ a \to Treap\ a$
$delete' :: Ord\ a \Rightarrow a \to a \to Treap\ a \to Treap\ a$

such that $delete'\ x\ y\ t$ removes all elements in interval $[x, y)$ from $t$. In this exercise, we allow duplicate elements in a treap.

### *Exercise 6.8*

Unordered lists, i.e. the *List* type class, can be efficiently implemented by a variant of treaps in which the *indices* of a list $0, \ldots,$ $length\ xs - 1$ are used as the ordered keys (the $x$ in $Node\ lt\ x\ p\ rt$) of a treap and the list elements are stored as *payloads* in treap nodes. Define

These following functions are useful in this exercise:

$length :: TList\ a \to Int$
$length\ EmptyT = 0$
$length\ (NodeT\ \_\ s\ \_\ \_\ \_) = s$

$payload :: TList\ a \to a$
$payload\ EmptyT = error$ `"empty"`
$payload\ (NodeT\ \_\ \_\ x\ \_\ \_) = x$

$priority :: TList\ a \to Int$
$priority\ EmptyT = error$ `"empty"`
$priority\ (NodeT\ \_\ \_\ \_\ p\ \_) = p$

**data** $TList\ a = EmptyT \mid NodeT\ (TList\ a)\ Int\ a\ Int\ (TList\ a)$

with invariants that any $NodeT\ lt\ s\ x\ p\ rt$ satisfies

$$s = length\ lt + length\ rt + 1$$

and the heap invariant that $p \leqslant priority\ lt$ and $p \leqslant priority\ rt$ whenever $lt$ or $rt$ is not empty. Note that the indices are *not* stored in the nodes. The list represented by a $TList\ a$ is

$toList :: TList\ a \to [a]$
$toList\ EmptyT = [\,]$
$toList\ (NodeT\ lt\ \_\ x\ \_\ rt) = toList\ lt +\!\!+ [x] +\!\!+ toList\ rt$

1. Implement $single :: MonadRandom\ m \Rightarrow a \to m\ (TList\ a)$ that creates a singleton list from an element with a random priority generated using the *MonadRandom* interface.

2. Implement $(!!) :: TList\ a \to Int \to a$ such that $xs\ !!\ n = toList\ xs\ !!\ n$ for any $xs :: TList\ a$ and $n$. The function should run in $O(depth\ t)$ time.

Same as treaps, the expected depth of $t :: TList\ a$ is $log\ (length\ t)$ when the priorities are random.

$depth :: TList\ a \to Int$
$depth\ EmptyT = 0$
$depth\ (NodeT\ l\ \_\ \_\ \_\ r) = 1 + max\ (depth\ l)\ (depth\ r)$

3. Implement $(+\!\!+) :: TList\ a \to TList\ a \to TList\ a$ in a way similar to $merge :: Treap\ a \to Treap\ a \to Treap\ a$ (see Exercise 6.5). The time complexity of $xs +\!\!+ ys$ should be in $O(depth\ xs + depth\ ys)$. Explain why the indices of the elements are not stored in *TList* nodes.

Both *tail* and *init* are special cases of *splitAt*.

4. Implement $splitAt :: Int \to TList\ a \to (TList\ a, TList\ a)$ in a way similar to *split* in Exercise 6.6 such that if $(xs, ys) = split\ n\ zs$ then $xs +\!\!+ ys = zs$ and $length\ xs = n$. The time complexity of $split\ n\ xs$ should be in $O(depth\ xs)$.