CS 315 Programming Languages

Homework 3

Yasin Balcancı / 21501109 / Section: 1

1) *Scheme*

Extra Functions:

```
(define (contains? list item)
   (cond
      ( (null? list) #f)
      ( (equal? item (car list) ) #t )
      (#t (contains? (cdr list) item) )
   )
)
(define (setifyhelp list1 out)
    (if (null? list1)
      out
      (begin
        (if (not (contains? out (car list1)))
          (setifyhelp (cdr list1) (append out (list (car list1))))
             (setifyhelp (cdr list1) out)))))
```

contains?: Takes a list and an atom as parameters; returns #t if the atom is in the list, #f otherwise.

setifyhelp: A recursive function that takes two lists as parameter, the first one is the actual list that will be provided to *setify* and the second one will be given as an empty list that will be filled with the unique atoms from the first list. Returns the unique atoms in list1 in the given order.

Actual Function:

```
(define (setify list1) (setifyhelp list1 '()))
```

Calls setifyhelp with the list given to this and an empty list, returns unique atoms of the list in the given order.

Sample Executions:

```
> (setify '())
()
> (setify '(1 2 3 1 2 3 4 5 5 6))
(1 2 3 4 5 6)
> (setify '(10 9 9 8 8 7 8 9 9 10))
(10 9 8 7)
```

2) *Join*

Actual Function:

```
(define (join list1 list2)
   (if (null? list1)
   list2
      (if (null? list2)
      list1
         (if (<= (car list1) (car list2))
            (setify (append (list (car list1)) (join (cdr list1) list2)))
               (setify (append (list (car list2)) (join list1 (cdr list2)))))
)))
```

Takes two lists as parameter, works recursively. If one of the lists is empty, returns the other one, otherwise takes the smaller one of the first elements of the lists and and unites it with the recursive call of itself. Setify is also called in order to avoid repetition.

Sample Executions:

```
> (join '() '())
()
> (join '() '(1 2 3))
(1 2 3)
> (join '(1 2 3) '())
(1 2 3)
> (join '(1 2 3) '(2 3 4 5))
(1 2 3 4 5)
```

3) BST Insert

Actual Function:

```
(define (insert-bst item list1)
   (if (null? list1)
       (list item '() '())
       (if (<= item (car list1))
       (list (car list1) (insert-bst item (car (cdr list1))) (car (cdr (cdr list1))))
       (list (car list1) (car (cdr list1)) (insert-bst item (car (cdr (cdr list1)))))
)))
```

Takes an atom and a list as parameters. If the list is empty, returns a list the first element of which is the given atom and the 2nd and 3rd elements are empty lists (left subtree and right subtree). If the list -the tree- is not empty, compares the given atom to the elements of the tree and places it so that the tree is a valid binary-search tree.

Sample Executions:

```
> (insert-bst 3 '(4 () (8 () ())))

(4 (3 () ()) (8 () ()))
> (insert-bst 8 '())

(8 () ())

> (insert-bst 5 '(8 () ()))

(8 (5 () ()) ())

> (insert-bst 9 '(8 (5 () ()) ()))

(8 (5 () ()) (9 () ()))

> (insert-bst 7 (insert-bst 4 (insert-bst 5 (insert-bst 8'())))))

(8 (5 (4 () ()) (7 () ())) ())
```