1 Set

1.1 Specification

Set: A collection of elements (without duplicates)

size() : Returns the size of the set
isEmpty() : Tests if the set is empty
elements() : Returns the collection of elements of the set
isMember(element) : Tests if element is in the set
insertElement(element) : Adds element to the set
union(other-set) : Returns the union of the set and other-set
intersect(other-set) : Returns the intersection of the set and other-set
subtract(other-set) : Returns the set minus other-set
isEqual(other-set) : Tests if the set is the same as other-set

1.2 Implementation

Variables

set: array of elements
size: int
inv: the array set is sorted
an element occurs at most once in the array set
the array set contains the elements of the set
size is the size of the set

Initialization

size = 0

Algorithms

size()
out: size of the set
return size

isEmpty()
out: the set is empty?
return (size = 0)

elements()
out: collection of elements of the set
let col be an empty collection
for $i = 0, \ldots,$ size-1
loop-inv: col contains the elements set[0], ..., set[i-1]
add set[i] to col
return col

binarySearch(element, low, high)
pre: $0 \leq$ low and high $<$ size
in: [low ... high] is the interval of the array set to be searched for element

```
out: is element contained in interval [low..high] of the array set
if the interval [low ... high] is empty
    return false
else
    set middle to be the middle of the interval [low ... high]
    if set[middle] = element
        return true
    else if set[middle] > element
        return binarySearch(element, low, middle-1)
    else
        return binarySearch(element, middle+1, high)
```

```
isMember(element)
in: element to be searched for
out: element in the set?
return binarySearch(element, 0, size-1)
```

```
insertElement(element)
in: element to be added to the set
post: element has been addded to the set
if element is not a member of the set
    i = size-1
    while set[i] > element and i ≥ 0
        loop-inv: set[i + 1], ... set[size-1] are moved one position to the right in the array set
        move set[i] one position to the right
        decrement i
    set[i + 1] = element
    increment size
```

```
union(other-set)
in: set to be added
out: union of the set and other-set
let temp be an array of elements
h = 0
i = 0
j = 0
while i < size and j < size of other-set
    loop-inv: {temp[0], ..., temp[h − 1]} = {set[0], ..., set[i − 1]} ∪ {other-set[0], ..., other-set[j − 1]}
    if set[i] = other-set[j]
        temp[h] = set[i]
        increment h, i and j
    else if set[i] < other-set[j]
        temp[h] = set[i]
        increment h and i
    else
        temp[h] = other-set[j]
        increment h and j
while i < size
    loop-inv: {temp[0], ..., temp[h − 1]} = {set[0], ..., set[i − 1]} ∪ {other-set[0], ..., other-set[j − 1]}
```

temp$[h]$ = set$[i]$
increment h and i
while $j <$ size of other-size
loop-inv: {temp$[0], \ldots,$ temp$[h-1]$} = {set$[0], \ldots,$ set[size $- 1$]}$\cup$
{other-set$[0], \ldots,$ other-set$[j-1]$}
temp$[h]$ = other-set$[j]$
increment h and j
return (temp, h)

intersect(other-set)
in: set to be intersected with
out: intersection of the set and other-set
let temp be an array of elements
$h = 0$
$i = 0$
$j = 0$
while $i <$ size and $j <$ size of other-set
loop-inv: {temp$[0], \ldots,$ temp$[h-1]$} = {set$[0], \ldots,$ set$[i-1]$}$\cap$ {other-set$[0], \ldots,$ other-set$[j-1]$}
if set$[i]$ = other-set$[j]$
temp$[h]$ = set$[i]$
increment h, i and j
else if set$[i]$ < other-set$[j]$
increment i
else
increment j
return (temp, h)

subtract(other-set)
in: set to be subtracted
out: subtraction of other-set from the set
let temp be an array of elements
$h = 0$
$i = 0$
$j = 0$
while $i <$ size and $j <$ size of other-set
loop-inv: {temp$[0], \ldots,$ temp$[h-1]$} = {set$[0], \ldots,$ set$[i-1]$}$\setminus$ {other-set$[0], \ldots,$ other-set$[j-1]$}
if set$[i]$ = other-set$[j]$
increment i and j
else if set$[i]$ < other-set$[j]$
temp$[h]$ = set$[i]$
increment h and i
else
increment j
while $i <$ size
loop-inv: {temp$[0], \ldots,$ temp$[h-1]$} = {set$[0], \ldots,$ set$[i-1]$}$\setminus$ {other-set$[0], \ldots,$ other-set$[j-1]$}
temp$[h]$ = set$[i]$
increment h and i
return (temp, h)

```
isEqual(other-set)
in: set to be checked to be equal to the set
out: set is equal to other-set?
i = 0
equal = (size = size of other-set)
while i < size and equal
    loop-inv: equal = (size = size of other-set and {set[0], ..., set[i]} = {other-set[0], ..., other-set[i]})
    if set[i] ≠ other-set[i]
        equal = false
    else
        increment i
return equal
```