DSA – Seminar 4
Sorted MultiMap (SMM)

```
//auxiliary function that will help us with the other operations (private function, it is not part of the interface).
//pre: smm is  SMM, k is a Tkey
//post: kNode is a ↑Node, prevNode is a ↑Node. If there is a node with k as key, kNode will be that node and prevNode will be the previous node. If there is no node with k as key, kNode will be NIL and prevNode will be the node after which the key k should be.

For the previous example (the one with the words and translations):
searchNode for „book” -> kNode the node with book, prevNode the node with blood
searchNode for „blood” -> kNode the node with blood, prevNode will be NIL
searchNode for „day” -> kNode will be NIL, prevNode the node with book
searchNode for „air” -> kNode will be NIL, prevNode will be NIL

subalgorithm searchNode(smm, k, kNode, prevNode) is:
    aux ← smm.head
    prev ← NIL
    found ← false
    while aux ≠ NIL and smm.R([aux].info.k, k) and  not found execute
        if [aux].info.k = k then
            found ← true
        else
            prev ← aux
            aux ← [aux].next
        end-if
    end-while
    if found then
        kNode ← aux
        prevNode ← prev
    else
        kNode ← NIL
        prevNode ← prev
    end-if
end-subalgorithm
Complexity: O(n)


subalgorithm search(smm, k, list) is:
    searchNode (smm, k, kNode, prevNode)
    if kNode = NIL then
        init(list) // return an empty list
    else
        list ← [aux].info.vl
    end-if
end-subalgorithm
Complexity: O(n)
```

```
subalgorithm add(smm, k, v) is:
    searchNode(smm, k, kNode, prevNode)
    if kNode = NIL then
        addANewKey (smm, k, v, prevNode)
    else
        addEnd([kNode].info.vl, v)
    end-if
end-subalgorithm
Complexity:
//searchNode is O(n)
//addANewKey is Θ(1) operation (we will use the prevNode)
//instead of addEnd another add function can be used (so it can have Θ(1) complexity)
If addEnd (or whatever function is used for values) is Θ(1) => O(n)
If addEnd (or whatever function is used for values) is Θ(length of the list) =>
O(smm)

//auxiliary operation (not part of interface)
//pre: smm is a SMM, k is a TKey, v is a TElem/ TValue,  prevNode is a ↑Node (the
node after which the new node should be added)
//post: a new node with key k and value v is added to the smm. The order of the keys
will respect the relation.
subalgorithm addANewKey (smm, k, v, prevNode) is:
    allocate(newNode)
    [newNode].info.k ← k
    init ([newNode].info.vl)
    addEnd([newNode].info.vl, v)
    if prevNode = NIL then
        [newNode].next ← smm.head
        smm.head ← newNode
    else
        [newNode].next ← [prevNode].next
        [prevNode].next ← newNode
    end-if
end-subalgorithm
Complexity: Θ (1) //supposing addToEnd it Θ(1) – which is true since in this
situation we will always add an element into an empty list


subalgorithm remove(smm, k, v) is:
    searchNode(smm, k, kNode, prevNode)
    if kNode ≠ NIL then
        pos ← indexOf([kNode].info.vl, v)
        if pos ≠ -1 then
            remove([kNode].info.vl, pos, e)
        end-if
        if isEmpty([kNode].info.vl) then
            removeKey(smm, k, prevNode)
        end-if
    end-if
end-subalgorithm
Complexity: O(smm)



//auxiliary operation (not part of the interface)
```

```
//pre: smm is a SMM, k is a TKey, prevNode is a ↑Node, smm contains a node with key k
after the node prevNode (if prevNode is NIL, then the first node if smm contains the
key k). The value list of the node with key k is empty.
//post: the node containing key k is removed from smm
subalgorithm removeKey(smm, k, prevNode) is:
      if prevNode = NIL then
            deleted ← smm.head
            smm.head ← [smm.head].next
            destroy([deleted].info.vl)
            free(deleted)
      else
            deleted ← [prevNode].next
            [prevNode].next ← [[prevNode].next].next
            destroy([deleted].info.vl)
            free(deleted)
      end-if
end-subalgorithm
```

Complexity: Θ(1)

Destroy will destroy an empty list => Θ(1)