Vlad Bogdan-Tudor, 317
Vlad

11)

1) As a data structure for the ADT SpecialContainer, I would use a Binary Search Tree with the relation "≤" (although it doesn't really matter). This way, I can get to minimum/maximum in $\Theta(\log_2 k)$ on average. In the representation of each node I would add a counter, so that we have in the tree as many nodes as there are unique values.

2) • init - I would just set the field that holds the number of elements to 0 and set the capacity to N

• push - I would traverse the tree, comparing the new priority with the current node's priority. After the traversal, I would just add this new node and set the link of the parent. If now nr Elems > capacity, then keep going right (to get to the max) and then remove that node. Traversing a BST takes, on average, $\Theta(\log_2 k)$.

Vlad Bogdan-Trichi, 917
Vlad

• pop - keep going left (to get to the minimum), comparing by priority. Remove that node. Traversing and removing this minimum takes on average, $O(\log_2 k)$

3.. Special Container:

root: ↑Node
cap: Integer
size: Integer

Node:

val: Integer
P: Integer
left: ↑Node
right: ↑Node

Vlad Bogdan-Tudor, 917
VladB

```
subalgorithm push(sc, v, p) is
    newNode ← @ allocate memory of one Node
    [newNode].val ← v
    [newNode].p ← p
    [newNode].left ← NIL
    [newNode].right ← NIL
    if sc.root = NIL then
        sc.root ← newNode
    else
        temp ← sc.root
        parent ← NIL
        while temp ≠ NIL execute
            if p < [temp].p then
                parent ← temp
                temp ← [temp].left
            else
                parent ← temp
                temp ← [temp].right
            end-if
        end-while
```

Vlad Bogdan-Tudor, 917
Vlad

```
temp ← newNode
if [parent].p ≥ [temp]. p then
        [parent]. left ← temp
else
        [parent]. right ← temp
end-if
sc.size ← sc.size + 1
if sc.size > capacity then
        parent ← NIL
        temp ← sc. root
        sc.size ← sc.size - 1
        while [temp]. right ≠ NIL execute
                parent ← temp
                temp ← temp. right
        end-while
        if temp = sc. root then
                sc. root ← [temp]. left
        else
                @deallocate temp
                [parent]. right ← NIL
        end-if
    end-if
end-if
end-subalgorithm
```

Vlad Bogdan-Tudor, 917
VlaB

4) I would use an AVL tree instead, and implement all the rotations needed. This ensures $\Theta(\log_2 k)$ for push/pop since the tree is balanced.