```
Heap:
    cap: Integer
    len: Integer
    elems: TElem[]
```

- Depending on the problem, you might need to have a *relation* as well, as part of the heap.

```
subalgorithm add(heap, e) is:
//heap - a heap
//e - the element to be added
    if heap.len = heap.cap then
        @ resize
    end-if
    heap.elems[heap.len+1] ← e
    heap.len ← heap.len + 1
    bubble-up(heap, heap.len)
end-subalgorithm
```

```
subalgorithm bubble-up (heap, p) is:
//heap - a heap
//p - position from which we bubble the new node up
    poz ← p
    elem ← heap.elems[p]
    parent ← p / 2
    while poz > 1 and elem > heap.elems[parent] execute
        //move parent down
        heap.elems[poz] ← heap.elems[parent]
        poz ← parent
        parent ← poz / 2
    end-while
    heap.elems[poz] ← elem
end-subalgorithm
```

- Complexity: $O(log_2 n)$

```
function remove(heap) is:
//heap - is a heap
   if heap.len = 0 then
      @ error - empty heap
    end-if
   deletedElem ← heap.elems[1]
   heap.elems[1] ← heap.elems[heap.len]
   heap.len ← heap.len - 1
   bubble-down(heap, 1)
   remove ← deletedElem
end-function
```

```
subalgorithm bubble-down(heap, p) is:
//heap - is a heap
//p - position from which we move down the element
   poz ← p
   elem ← heap.elems[p]
   while poz < heap.len execute
      maxChild ← -1
      if poz * 2 ≤ heap.len then
      //it has a left child, assume it is the maximum
         maxChild ← poz*2
      end-if
      if poz*2+1 ≤ heap.len and heap.elems[2*poz+1]>heap.elems[2*poz] th
      //it has two children and the right is greater
         maxChild ← poz*2 + 1
      end-if
//continued on the next slide...
```

```
      if maxChild ≠ -1 and heap.elems[maxChild] > elem then
         tmp ← heap.elems[poz]
         heap.elems[poz] ← heap.elems[maxChild]
         heap.elems[maxChild] ← tmp
         poz ← maxChild
      else
         poz ← heap.len + 1
         //to stop the while loop
      end-if
   end-while
end-subalgorithm
```

- Complexity: $O(log_2 n)$