```
algorithm MaxHeapify($A$, $i$)
    $l$ = left($i$)
    $r$ = right($i$)
    if $l \leq$ heap_größe[$A$] and $A[l] > A[i]$ then
        $max = l$
    else
        $max = i$
    end if
    if $r \leq$ heap_größe[$A$] and $A[r] > A[max]$ then
        $max = r$
    end if
    if $max \neq i$ then
        swap($A[i]$, $A[max]$)
        MaxHeapify($A$, $max$)
    end if
end algorithm
```

```
algorithm MaxHeapIncreaseKey($A, i, key$)
    if $key < A[i]$ then
        Error
    end if
    $A[i] = key$
    while $i > 1$ and $A[\text{Vater}(i)] < A[i]$ do
        swap($A[i], A[\text{Vater}(i)]$)
        $i = \text{Vater}(i)$
    end while
end algorithm
```

```
algorithm MaxHeapInsert($A$, $key$)
    heap_größe[$A$] = heap_größe[$A$] + 1
    $A$[heap_größe[$A$]] = $-\infty$
    MaxHeapIncreaseKey($A$, heap_größe[$A$], $key$)
end algorithm
```

```
algorithm MaxHeapExtractMax($A$)
    if heap_größe[$A$] $< 1$ then
        Error
    end if
    $max = A[1]$
    $A[1] = A[$heap_größe[$A$]$]$
    heap_größe[$A$] = heap_größe[$A$] $- 1$
    MaxHeapify($A, 1$)
    return $max$
end algorithm
```

```
algorithm BuildMaxHeap($A$)
    heap_größe[$A$] = länge[$A$]
    for $i = \left\lfloor \frac{\text{länge}[A]}{2} \right\rfloor$ down to $1$ do
        MaxHeapify($A, i$)
    end for
end algorithm
```

```
algorithm Heapsort($A$)
    BuildMaxHeap($A$)
    for $i$ = länge[$A$] down to 2 do
        swap($A[1], A[i]$)
        heap_größe[$A$] = heap_größe[$A$] $- 1$
        MaxHeapify($A, 1$)
    end for
end algorithm
```