```
SiftUp(node):
  if (node does not have a parent) done!!
  if (node.parent.value>node.value)
  | swap(node.value,node.parent.value);
  | SiftUp(node.parent);
```

```
Heapify(node):
  // pre: node.left, node.right are roots
  //      of valid heaps or null
  // post: node is a root of a valid heap
  if not(node.left) and not(node.right) done!!
  else
  | if node.right and node.right.value<node.left.value
  | | k:=node.right
  | else
  | | k:=node.left
  |
  | if node.value<=k.value done!!
  | else
  | | swap(node.value,k.value);
  | | Heapify(k);
```

```
SiftUp(node):
  while (node>1 and A[node]<A[node/2]) {
  | swap(A[node],A[node/2]);
  | node:=node/2

Heapify(node):
  // for simplicity assume unused part of A filled with infty
  while (A[node]>A[2*node] or A[node]>A[2*node+1]) {
  | if A[2*node]<A[2*node+1]
  | | k:=2*node
  | else
  | | k:=2*node+1
  | swap(A[node],A[k])
  | node:=k
```

```
BuildHeap(n):
  // build heap from values stored in A[1..n]
  for i:=n/2 downto 1
    // inv: elements i+1,...,n are roots of valid heaps
    Heapify(i);
    // inv: elements i,i+1,...,n are roots of valid heaps

Heapify(node):
  // pre: node.left (2*node), node.right (2*node+1) are roots
  //      of valid heaps or out of bounds
  // post: node is a root of a valid heap
```

Prioritné fronty

Implementácia	Insert	ExtractMin	Pozn.
Neutriedené pole	$\Theta(1)$	$\Theta(n)$	
Utriedené pole	$\Theta(n)$	$\Theta(1)$	
Počítadlá	$\Theta(1)$	$\Theta(1)$	malá doména (napr. $[1, 1000]$)
Heap	$\Theta(\log n)$	$\Theta(\log n)$	