Heap Sort

Eingabe: Array A mit n Elementen

Ausgabe: sortiertes Array A

HeapSort(A)

(1) $H :=$ CreateHeap(A);

(2) **for** $i := 1$ **to** n **do**

(3) **output**(DeleteMax(H))

(4) **end**

Eingabe: fast vollständiger binärer Baum T, in dem alle Knoten, bis auf evtl. die Wurzel w, die Heap-Eigenschaft erfüllen

Ausgabe: Heap T

REHEAP(T)

(1) $v := w$;

(2) **while** Heap-Eigenschaft in v nicht erfüllt **do**

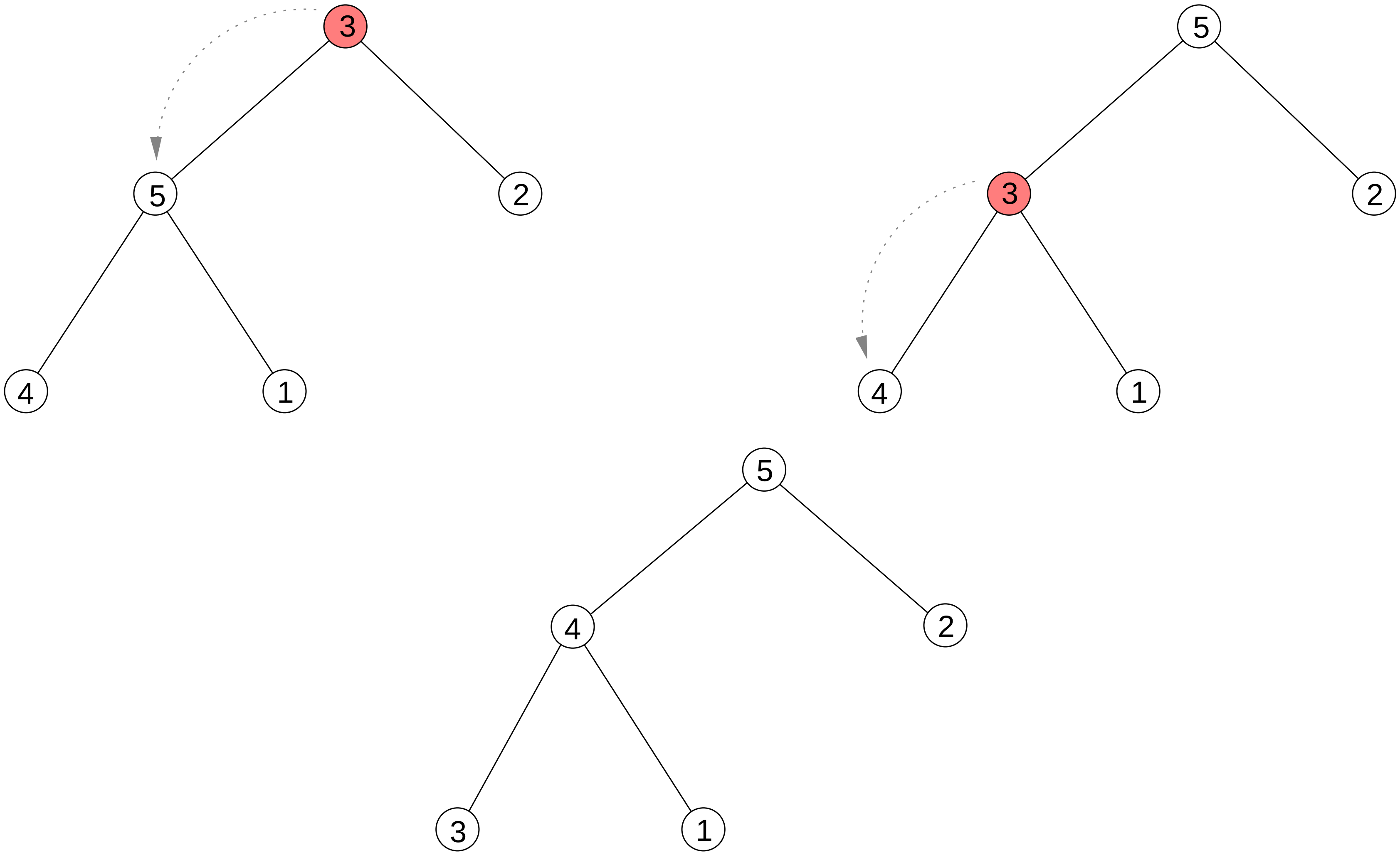
(3) Sei v^* dasjenige Kind von v, das den größeren Schlüssel speichert;

(4) vertausche die Schlüssel in v und v^*;

(5) $v := v^*$;

(6) **end**

Beispiel 1.



Eingabe: Heap H

Ausgabe: das größte Element k von H und ein Heap H', wobei H' durch Entfernen von k aus H entsteht

DELETEMAX(H)

(1) Sei w die Wurzel von H;

(2) Sei k der in w gespeicherte Schlüssel;

(3) Sei l das rechteste Blatt in der tiefsten Ebene;

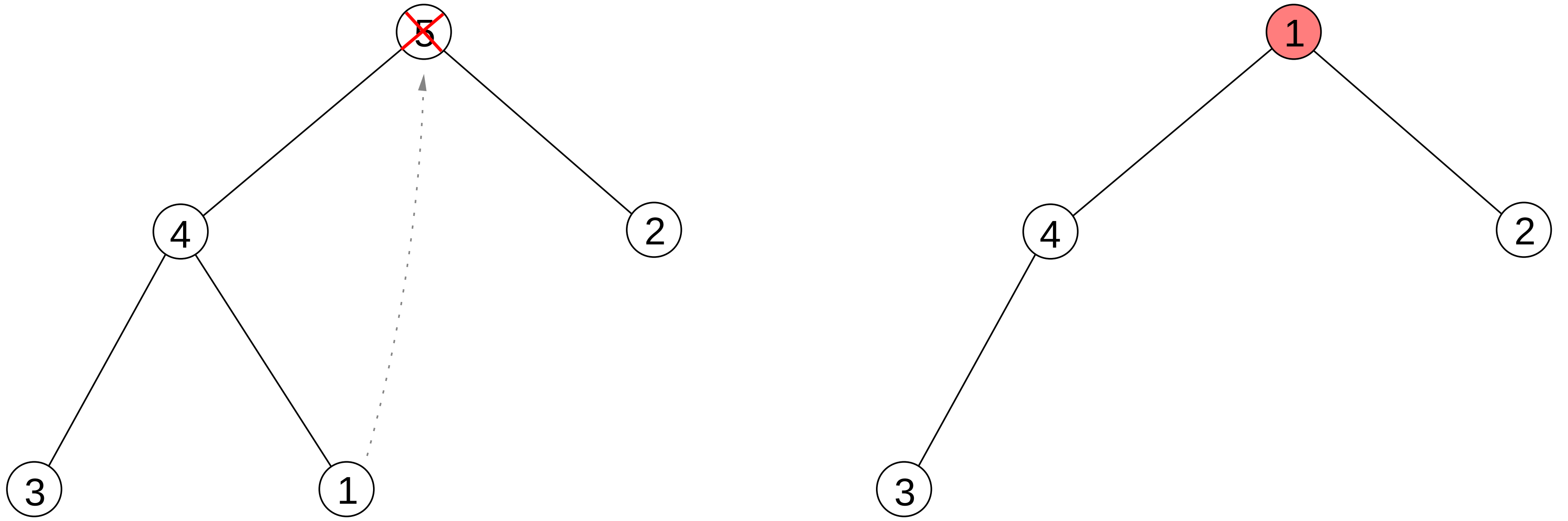
(4) Kopiere den Schlüssel in l nach w;

(5) Lösche das Blatt l in H;

(6) **if** SIZE$(H) > 0$ **then** REHEAP(H);

(7) **return** k;

Beispiel 2.



Eingabe: Array A mit n Elementen

Ausgabe: Heap H

CREATEHEAP(A)

(1) Konstruiere einen fast vollständigen binären Baum H mit n Knoten;
(2) Fülle jeden Knoten mit einem Schlüssel aus A;
(3) **for** $l :=$ height(H) **to** 0 **by** -1 **do**
(4)   **foreach** v auf Ebene l **do**
(5)   /* $H(v)$ ist ein Unterbaum von H mit Wurzel v */
(6)   REHEAP($H(v)$);
(7)   **end**
(8) **end**
(9) **return** H;