OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

INFORMATIK UND
MATHEMATIK

Algorithmen und Datenstrukturen

- Sortieralgorithmen -

Prof. Dr. Klaus Volbert

Wintersemester 2018/19
Regensburg, 25. Oktober 2018

Überblick Sortieralgorithmen

- Nichts ist besser verstanden als das Sortieren…
- Einfache Sortieralgorithmen
  - Sortieren durch Einfügen (Insertion Sort) ✓
  - Sortieren durch Vertauschen (Bubble Sort) ✓
  - Sortieren durch Auswählen (Selection Sort) ✓
- Fortgeschrittene Sortieralgorithmen
  - Sortieren durch Divide & Conquer (Quicksort) ✓
  - Sortieren durch Mischen (Merge Sort) ✓
  - Sortieren durch Bäume (Heap Sort)
- Spezielle Sortieralgorithmen
  - Sortieren durch Zählen (Count Sort)
  - Sortieren durch Abbilden (Map Sort)
- Untere Schranke für vergleichsbasierte Sortieralgorithmen

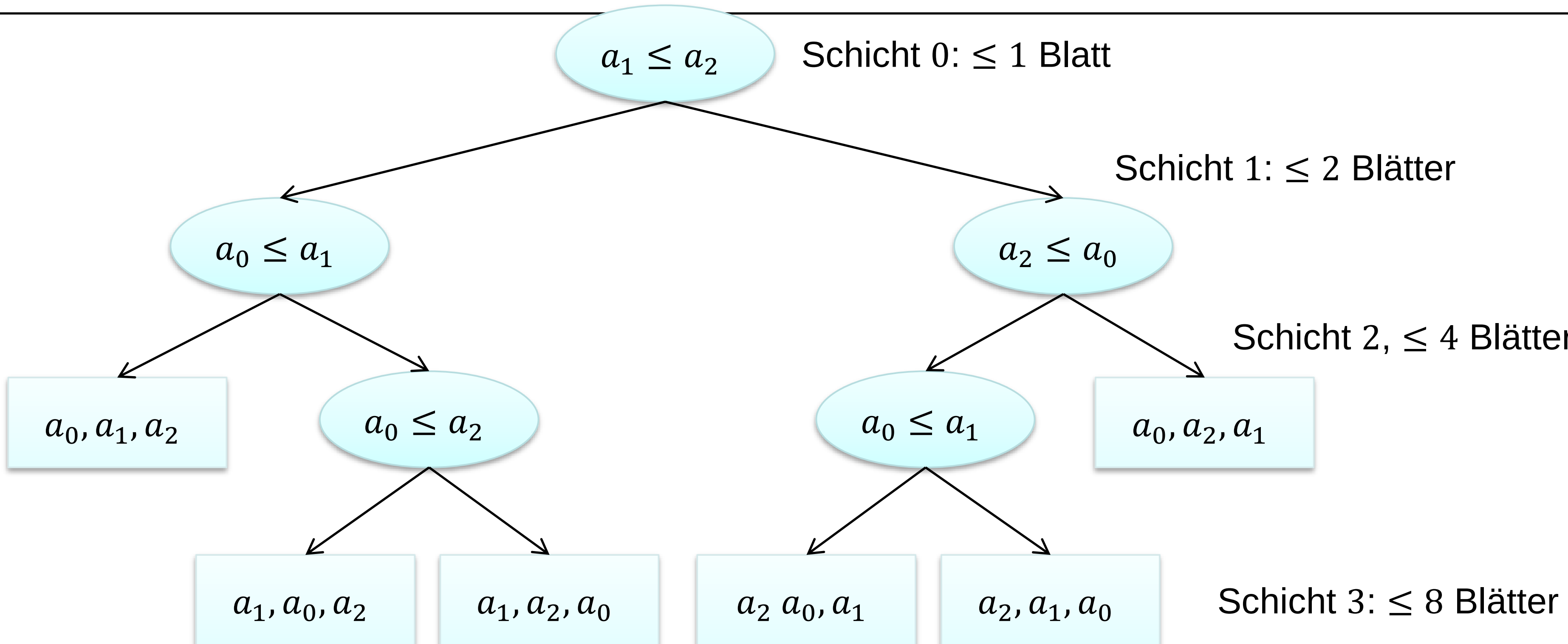
Stabilität von Sortieralgorithmen

- Ein Sortieralgorithmus ist stabil, wenn die relative Ordnung der Elemente mit gleichen Schlüsseln durch den Sortierprozess nicht verändert wird
- Beispiel: Sortierung nach Anfangsbuchstaben (1-26)
  - Eingabe: Markus, Sven, Walter, Michael, Franz
    13, 19, 23, 13, 6
  - Ausgabe 1: Franz, Markus, Michael, Sven, Walter
    6, 13, 13, 19, 23
  - Ausgabe 2: Franz, Michael, Markus, Sven, Walter
    6, 13, 13, 19, 23
- Welche der Ausgaben kommt von einem stabilen Sortieralgorithmus?
  - Ausgabe 1: Markus steht weiterhin vor Michael
- Welche Sortieralgorithmen sind stabil?
  - Einfache: Insertion Sort, Bubble Sort
  - Fortgeschrittene: Merge Sort

Untere Schranke für vergleichsbasiertes Sortieren I

- Bisher betrachtete Sortieralgorithmen
  - Kein a-priori-Wissen über die zu sortierenden Objekte
  - Basis: Paarweiser Größenvergleich (Vergleichsorientierte Verfahren)
  - D.h. Reihenfolge von $a, b \in$ IN wird durch $a < b$? entschieden
- Eine untere Schranke für die Laufzeit von vergleichsorientierten Sortieralgorithmen liefert ein Entscheidungs- oder Vergleichsbaum
- Ein Entscheidungs- oder Vergleichsbaum für $a_0, \ldots, a_{n-1}$ zu sortierenden Werten ist ein binärer Baum, an dessen innere Knoten Vergleiche der Form $a_i \leq a_j$? stehen (Vergleich zweier Elemente). Über den linken Ast geht es weiter, wenn $a_i \leq a_j$, ansonsten über den rechten Ast
- Eine konkrete Sortierung ergibt sich aus einem Pfad von der Wurzel bis zu einem Blatt
- Ein Entscheidungs- oder Vergleichsbaum heißt Sortierbaum, falls alle Eingaben mit dem gleichen Pfad auch die gleiche Permutation haben

Ein Sortierbaum für $n = 3$



Schicht i: $\leq 2^i$ Blätter

- Beobachtungen:
  - Ein Sortierbaum hat mindestens $n!$ Blätter (alle Permutationen müssen vertreten sein)
  - Die Höhe des Baumes ist eine untere Schranke für jeden vergleichsorientierten Sortieralgorithmus, wie hoch muss der Baum sein?

Untere Schranke für vergleichsbasiertes Sortieren II

- Damit alle Permutationen abgedeckt werden, muss gelten:

$$n! \leq 2^i$$

- Dann sind mindestens i Schritte notwendig (vergleichsbasiertes Sortieren)
- Umformen mittels Stirling-Formel ergibt (Übung):

$$i \geq \log(n!) = \Omega(n \log n)$$

- Satz: Für die Laufzeit von vergleichsbasierten Sortieralgorithmen folgt:

$$T(n) = \Omega(n \log n)$$

- Folgerungen
  - Die Laufzeit von Quicksort im Average Case ist asymptotisch optimal
  - Die Laufzeit von Merge Sort ist in allen Fällen asymptotisch optimal

Übersicht Sortieralgorithmen

Algorithmus	Best Case	Average Case	Worst Case	Stabil
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	ja
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	ja
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	nein
Quicksort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	nein
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	ja
Heap Sort	…	…	…	…
…	…	…	…	…

- Untere Schranke für vergleichsbasierte Sortieralgorithmen

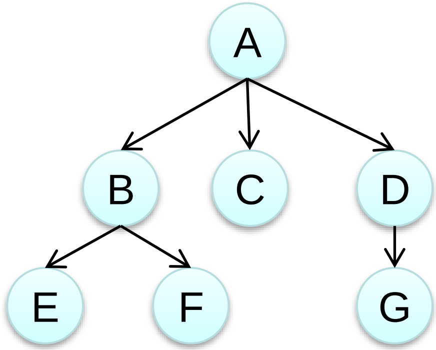
$$T(n) = \Omega(n \log n)$$

Datenstruktur: Baum (Tree)

Deutsch	Englisch	Bemerkung	Beispiel
Wurzel	root	Kein VG, allgemein: Knoten	A
Nachfolger (NF)	successor	Allgemein: Knoten	B ist NF von A
Vorgänger (VG)	predecessor	Allgemein: Knoten	D ist VG von G
Grad eines Knotens	degree	Anzahl der NF eines Knotens	D: Grad 1
Grad eines Baums	degree	Max. Grad im Baum	Baum: Grad 3
Blatt	leaf	Kein NF	C, E, F, G
Innerer Knoten	inner node	Alle NF besetzt	A
Randknoten	boundary node	Nicht alle NF besetzt	B, D
Pfad	path	Knotenfolge (Start, …, Ziel)	A-F: A, B, F
Pfadlänge	path length	Anzahl der Kanten des Pfades	A-F: 2
Tiefe/Höhe eines Knotens	depth/heigth	Pfadlänge zur Wurzel/zu einem Blatt	G: Tiefe 2/Höhe 0
Tiefe/Höhe eines Baums	depth/heigth	Maximale Tiefe/Höhe	2

Beispiel: Baum (Tree)

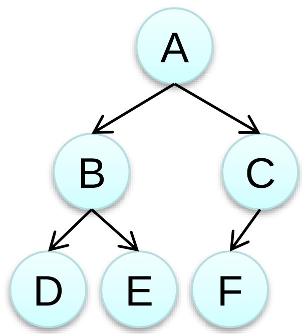
- Beispiel:



- Eigenschaften:
  - Ein Baum mit n Knoten hat genau $n-1$ Kanten
  - Eine Schicht besteht aus allen Knoten gleicher Tiefe (Schicht 0 bis Schicht h)
  - Sind alle Schichten voll, so ist der Baum vollständig

Binärer Suchbaum

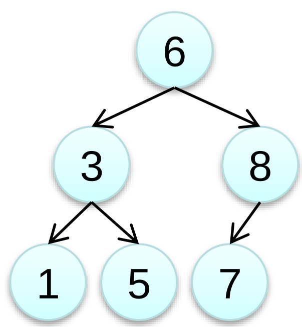
- Ein Binärbaum ist ein Baum mit zwei Nachfolgern
- Ein Binärbaum heißt linksvoll bzw. rechtsvoll, wenn der Baum bis auf die unterste Schicht vollbesetzt ist und die unterste Schicht entweder von links oder von rechts her ohne Lücken besetzt ist
- Beispiel:



Speicherung als Feld:

A	B	C	D	E	F

- Ein binärer Suchbaum ist ein linksvoller Binärbaum mit der Eigenschaft:
  - $\forall$ Knoten gilt: Werte links < eigener Wert < Werte rechts
- Beispiel:

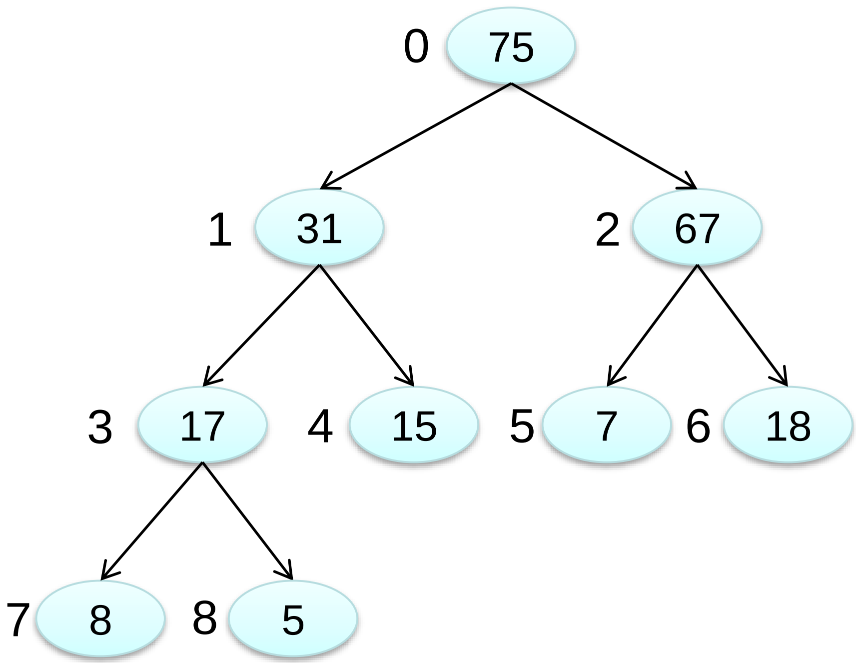


Speicherung als Feld:

6	3	8	1	5	7

Heap (Vorbereitung für Heap Sort)

- Ein Heap (Haufen) ist ein linksvoller Binärbaum mit der Eigenschaft:
  - ∀ Knoten gilt: Eigener Wert ≤ Werte der Nachfolger (Minimum), <u>oder</u>
  - ∀ Knoten gilt: Eigener Wert ≥ Werte der Nachfolger (Maximum)
- Beispiel (Max-Heap):



Speicherung als Feld a:

75	31	67	17	15	7	18	8	5

Zugriff:
- Wurzel: $a[0]$
- Aktuelle Position: i
- Vorgänger: $\left\lfloor \frac{i-1}{2} \right\rfloor$
- Linker NF: $a[2i+1]$
- Rechter NF: $a[2i+2]$

Heap-Eigenschaft: $\forall\, i\colon a[i] \geq a[2i+1]$ und $a[i] \geq a[2i+2]$

Sortieren mit Heaps (Heap Sort)

```
void Heapify(int a[],int f,int l,int root)
{
   int largest, left=f+(root-f)*2+1, right=f+(root-f)*2+2;
   if (left<=l && a[left]>a[root]) largest=left;
   else largest=root;
   if (right<=l && a[right]>a[largest]) largest=right;
   if (largest!=root) {
       swap(a[root],a[largest]);
       Heapify(a,f,l,largest); }
}
```

Bestimme max{root,left,right} und korrigiere ggf. den Heap

```
void HeapSort(int a[],int f,int l)
{
  BuildHeap(a,f,l);
  for (int i=l;i>f;i--)
  {
    swap(a[f],a[i]);
    Heapify(a,f,i-1,f);
  }
}
```

Baue aus a[] einen Heap

```
void BuildHeap(int a[],int f,int l)
{
  int n=l-f+1;
  for (int i=f+(n-2)/2;i>=f;i--)
      Heapify(a,f,l,i);
}
```

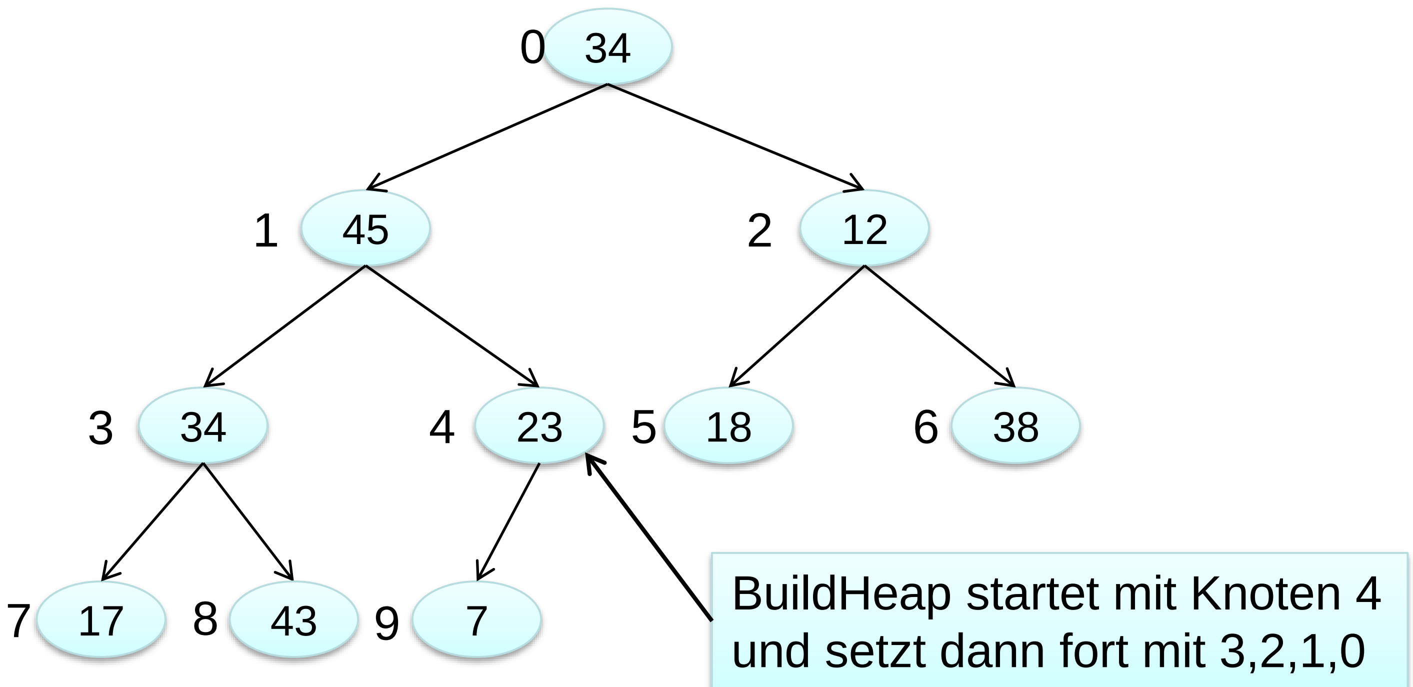
Extrahiere jeweils das Maximum aus dem Heap und baue so eine sortierte Folge

Beispiel Heap Sort

- Eingabe

34 45 12 34 23 18 38 17 43 7

- Darstellung als Heap



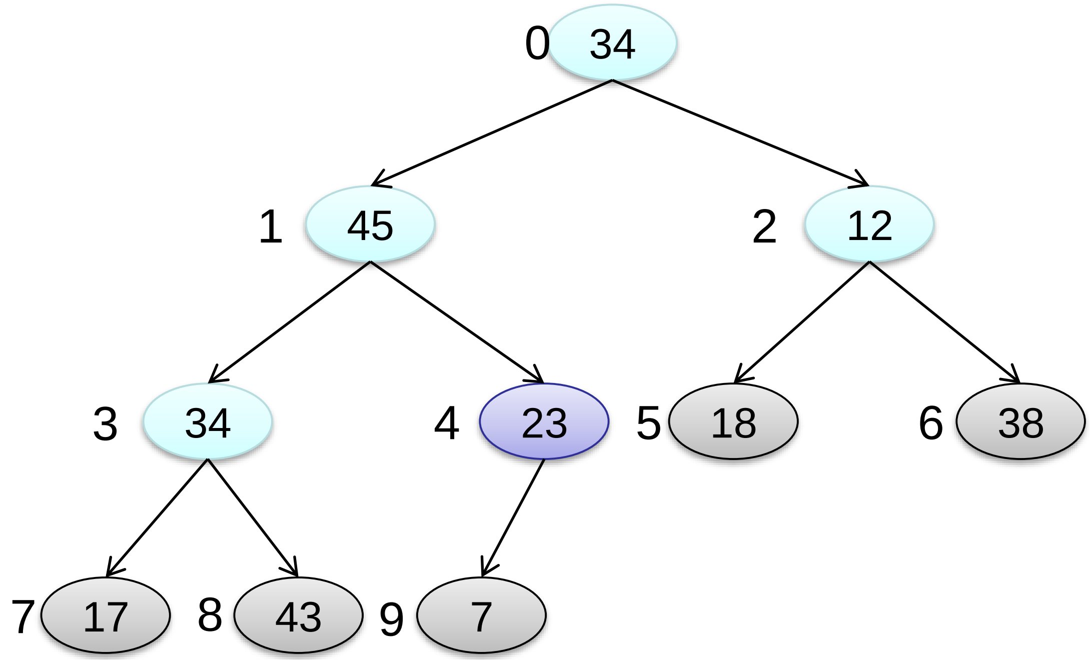
- Beobachtung: Blätter erfüllen die Heap-Eigenschaft

Beispiel Heap Sort

- Aktuelles Feld

  34 45 12 34 23 18 38 17 43 7

- Darstellung als Heap



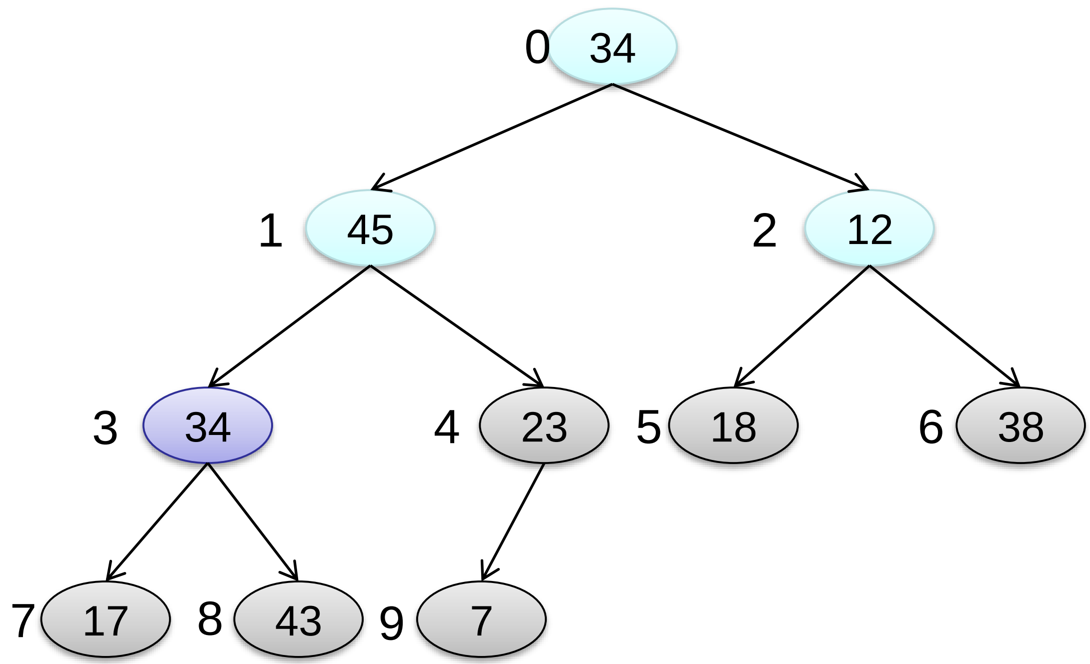
- Bearbeitung Knoten 4: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

  34 45 12 34 23 18 38 17 43 7

- Darstellung als Heap



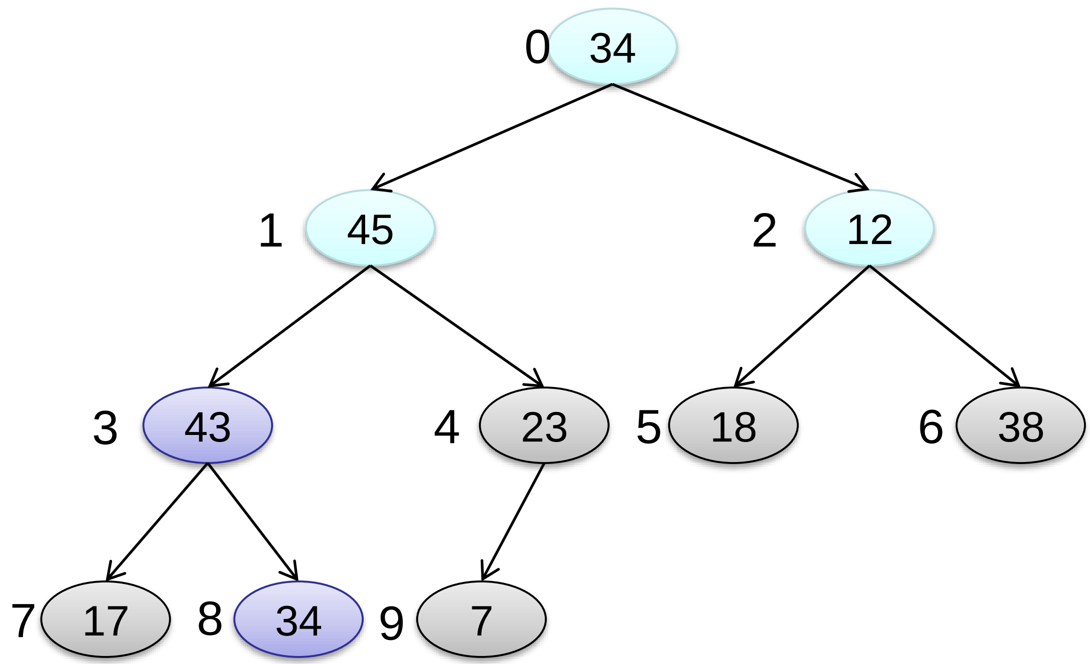
- Bearbeitung Knoten 3: Größtes Element ist der rechte Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  34 45 12 43 23 18 38 17 34 7

- Darstellung als Heap



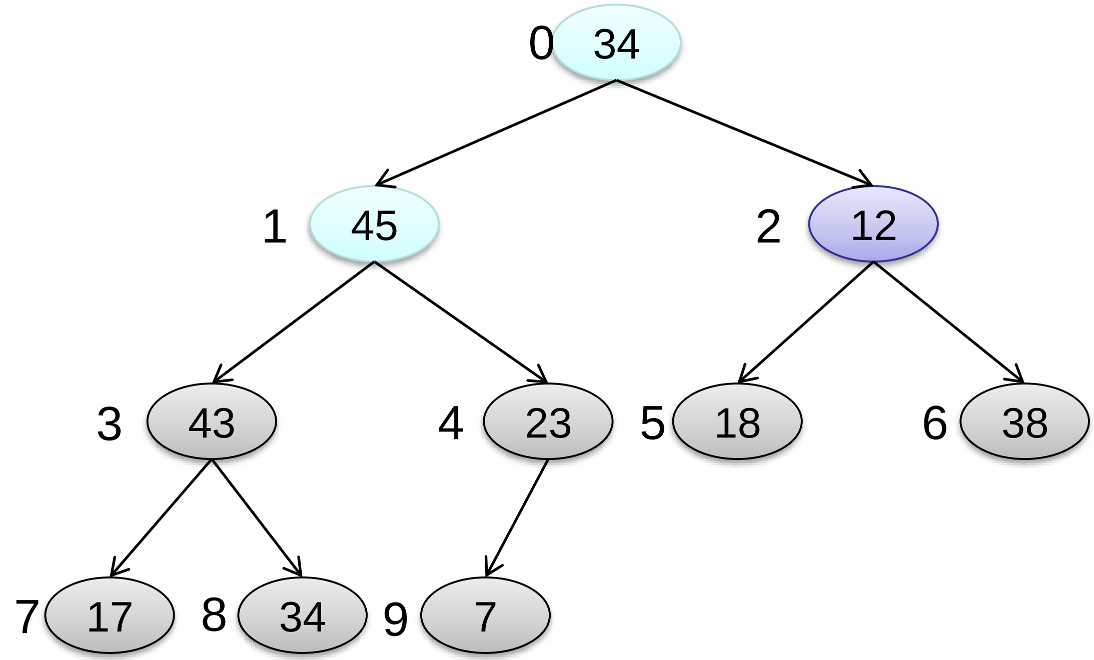
- Bearbeitung Knoten 8: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

34 45 12 43 23 18 38 17 34 7

- Darstellung als Heap



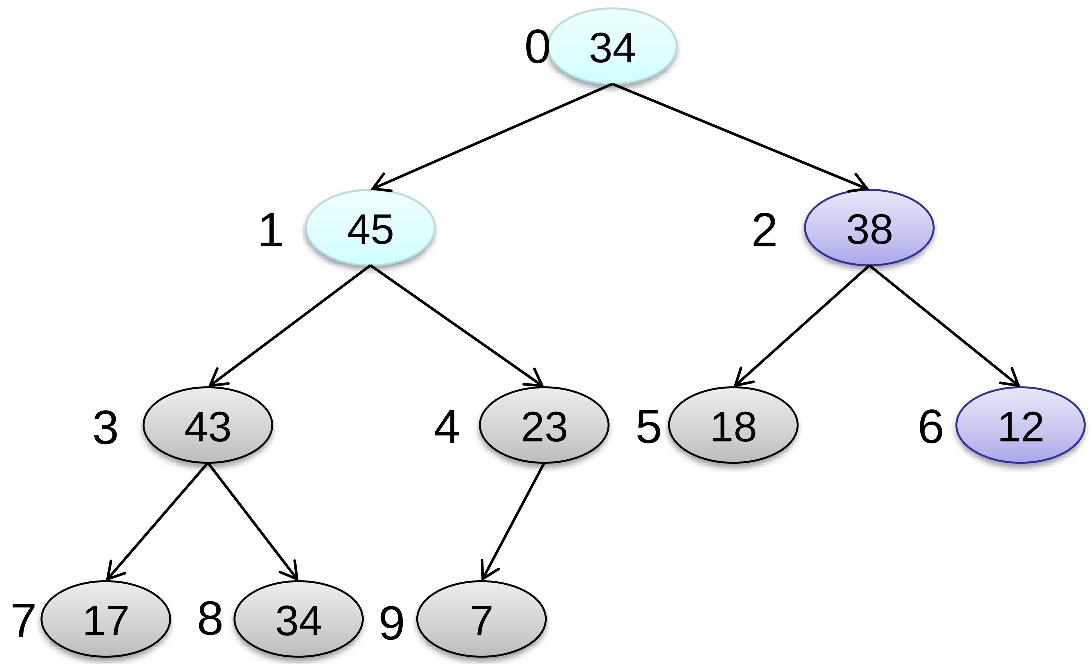
- Bearbeitung Knoten 2: Größtes Element ist der rechte Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  34 45 38 43 23 18 12 17 34 7

- Darstellung als Heap



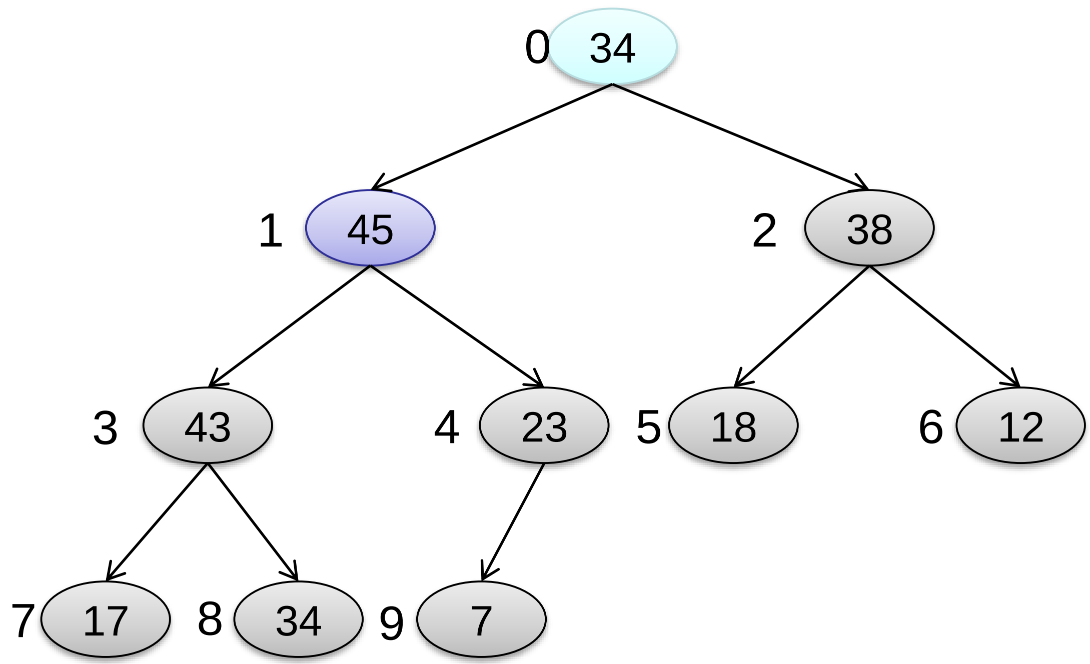
- Bearbeitung Knoten 6: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

34	45	38	43	23	18	12	17	34	7

- Darstellung als Heap



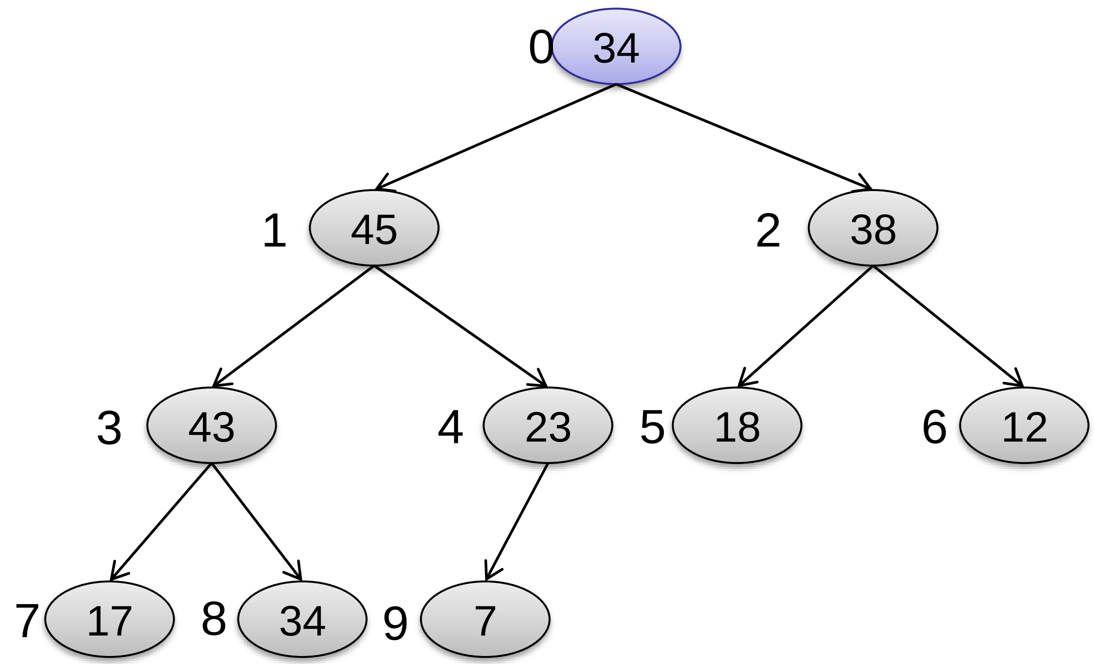
- Bearbeitung Knoten 1: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

34 45 38 43 23 18 12 17 34 7

- Darstellung als Heap



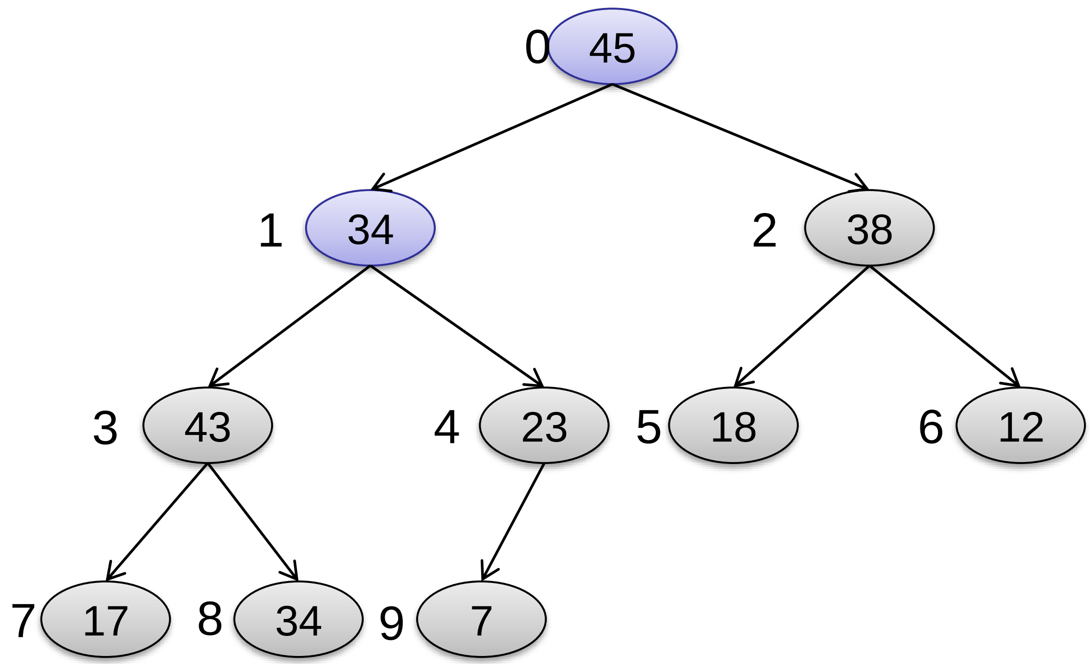
- Bearbeitung Knoten 0: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

45 34 38 43 23 18 12 17 34 7

- Darstellung als Heap



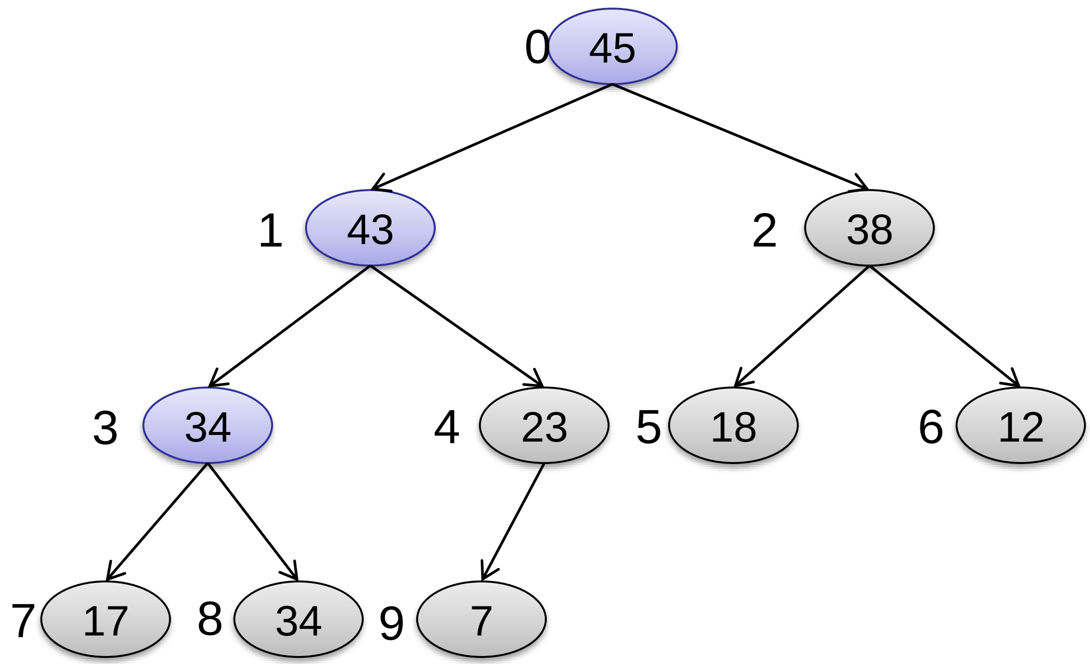
- Bearbeitung Knoten 1: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  45 43 38 34 23 18 12 17 34 7

- Darstellung als Heap



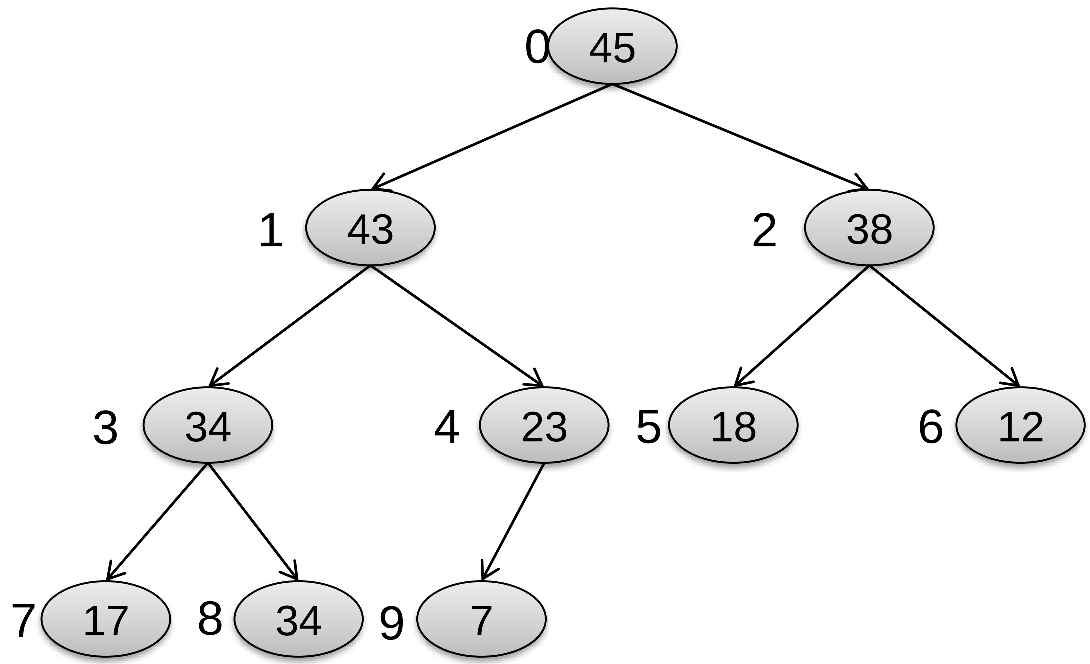
- Bearbeitung Knoten 3: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

45 43 38 34 23 18 12 17 34 7

- Darstellung als Heap



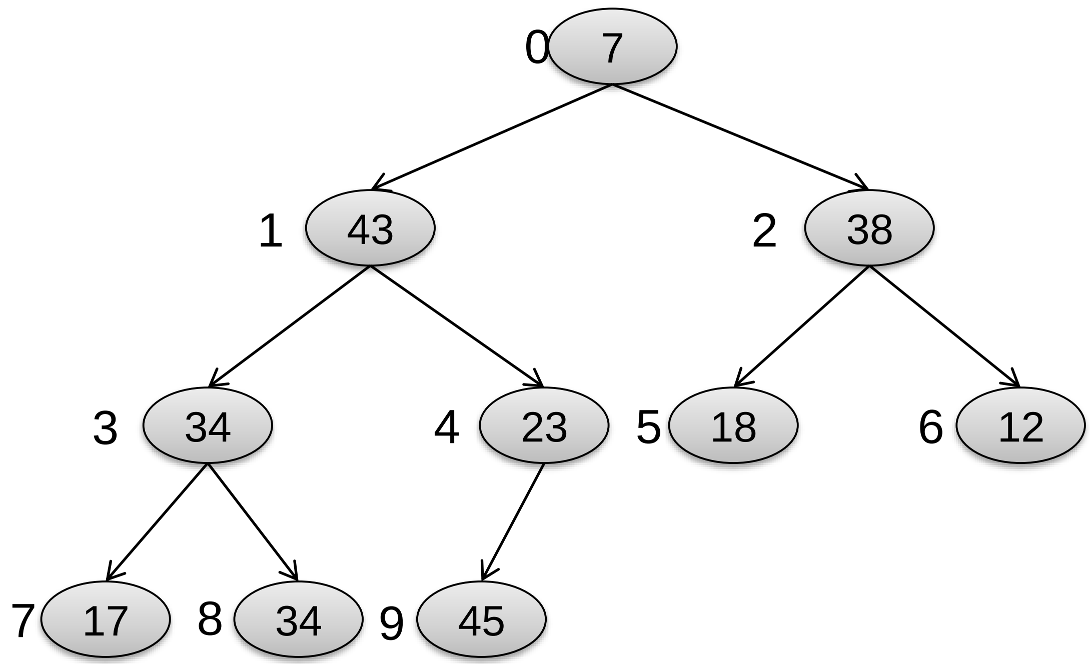
- Durchführung von BuildHeap ist beendet!

Beispiel Heap Sort

- Aktuelles Feld

7 43 38 34 23 18 12 17 34 45

- Darstellung als Heap



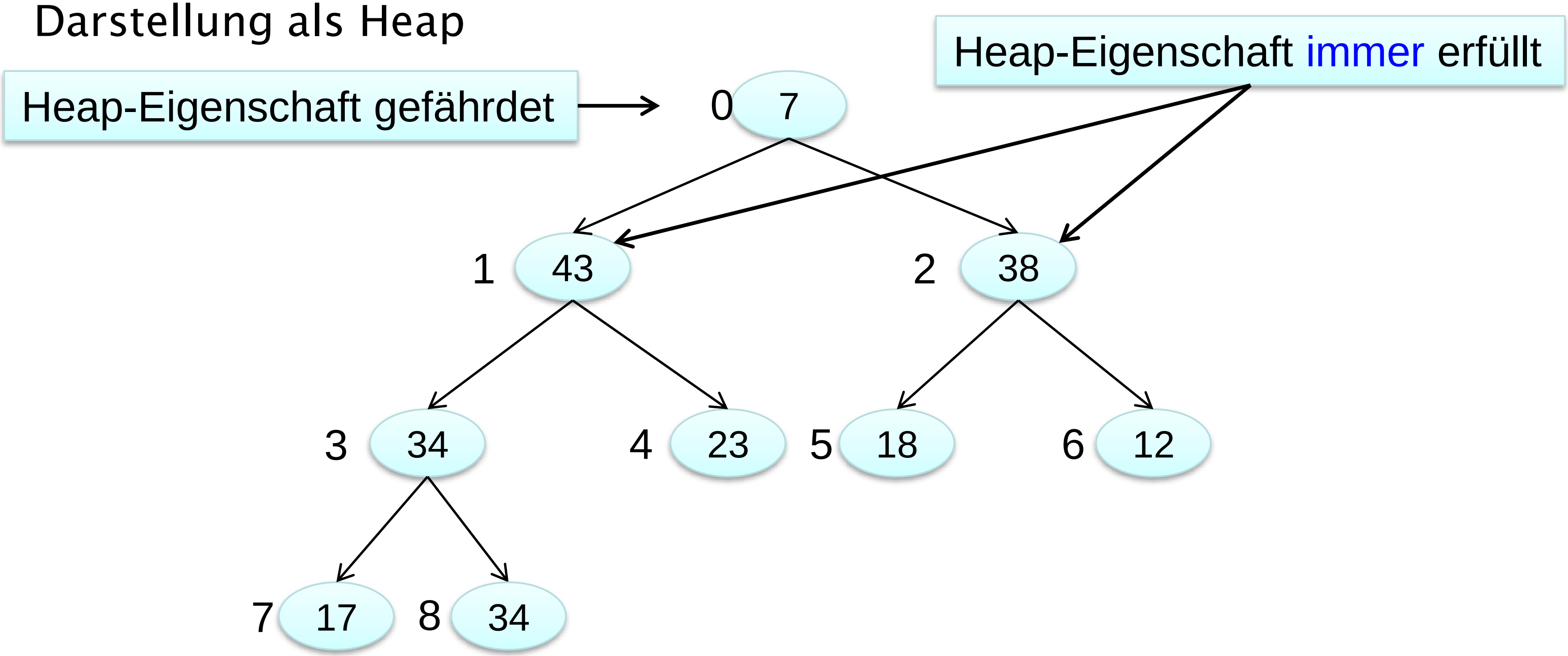
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

7 43 38 34 23 18 12 17 34 <u>45</u>

- Darstellung als Heap



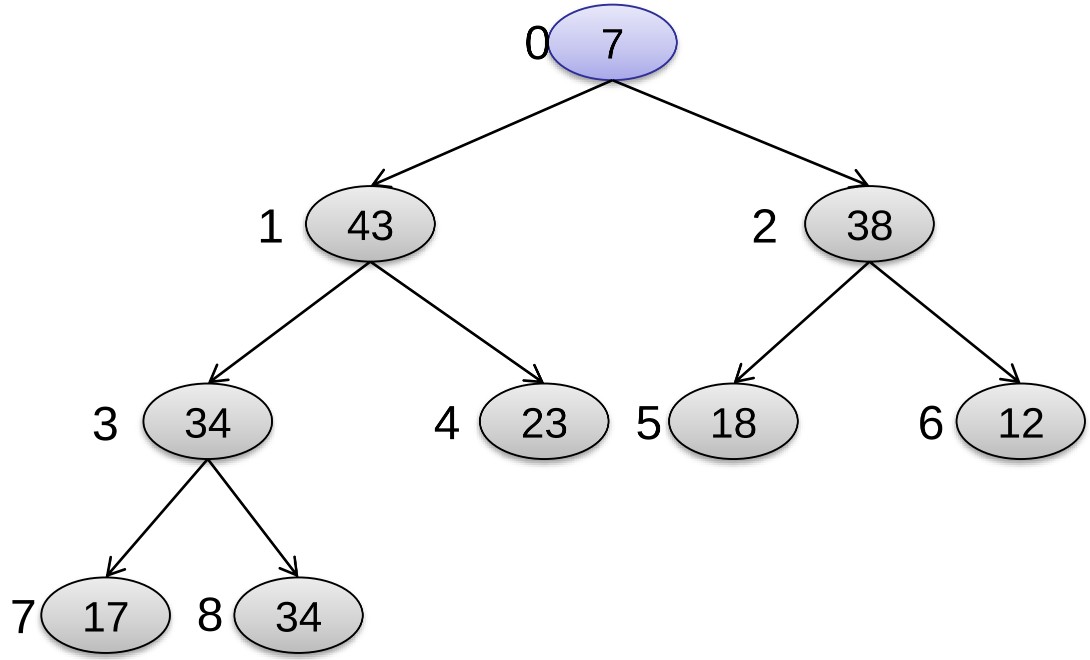
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

  7 43 38 34 23 18 12 17 34 <u>45</u>

- Darstellung als Heap



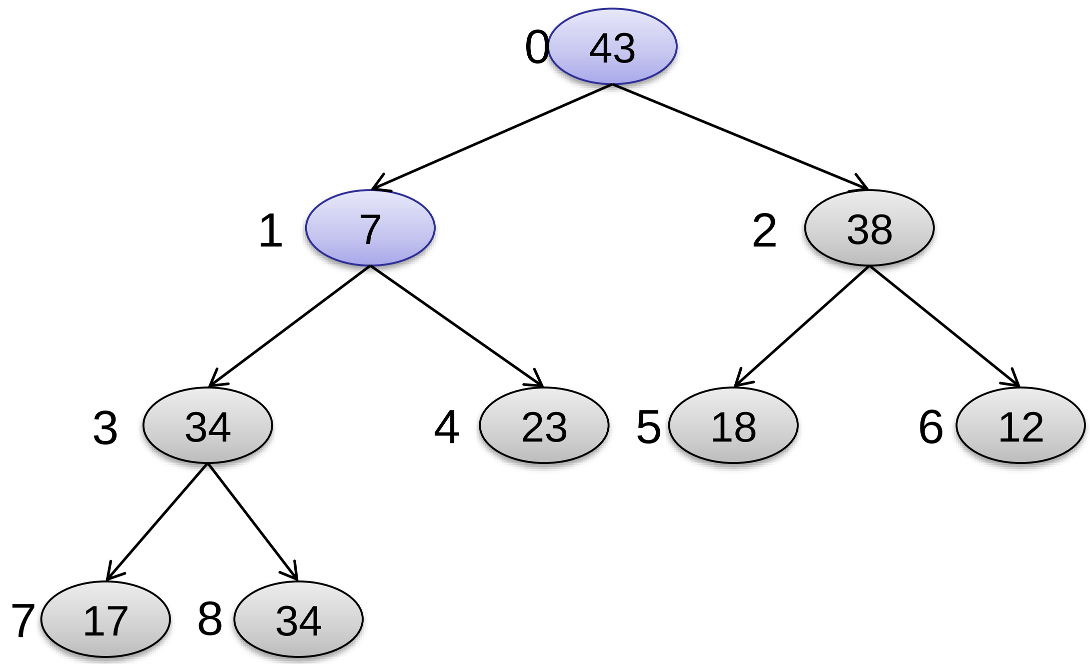
- Bearbeitung Knoten 0: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

43 7 38 34 23 18 12 17 34 <u>45</u>

- Darstellung als Heap



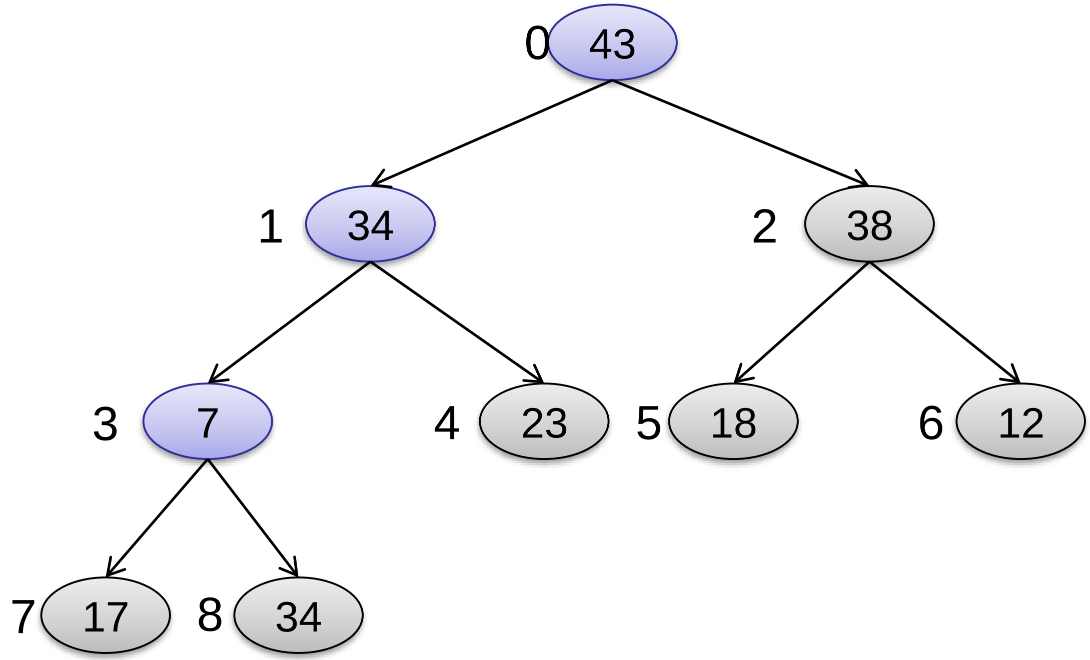
- Bearbeitung Knoten 1: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  43 34 38 7 23 18 12 17 34 <u>45</u>

- Darstellung als Heap



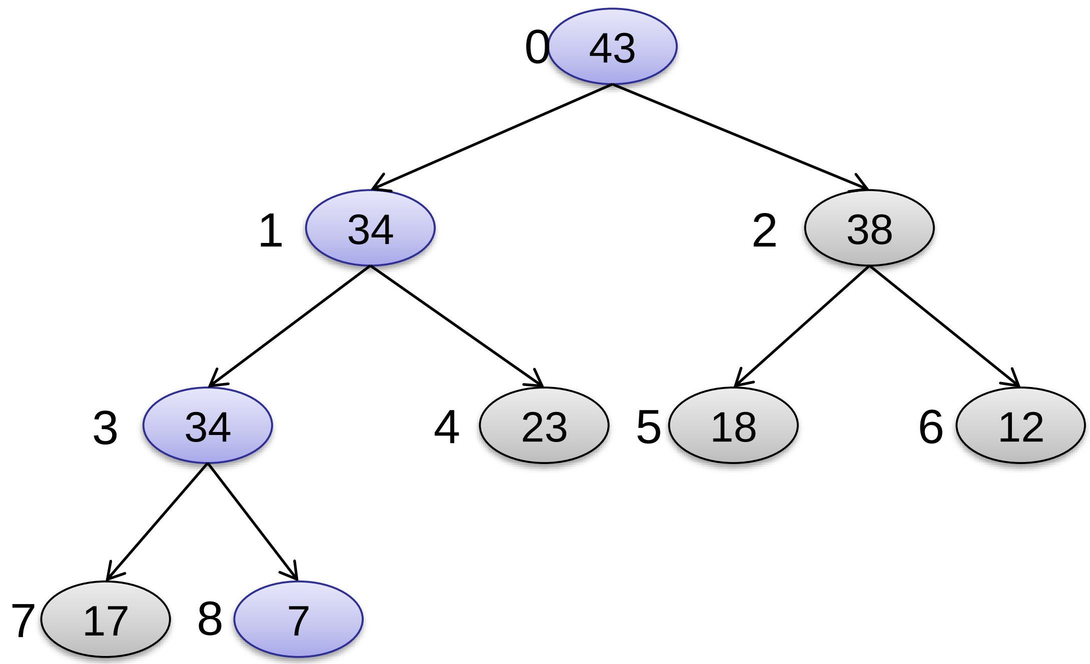
- Bearbeitung Knoten 3: Größtes Element ist der rechte Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

43 34 38 34 23 18 12 17 7 <u>45</u>

- Darstellung als Heap



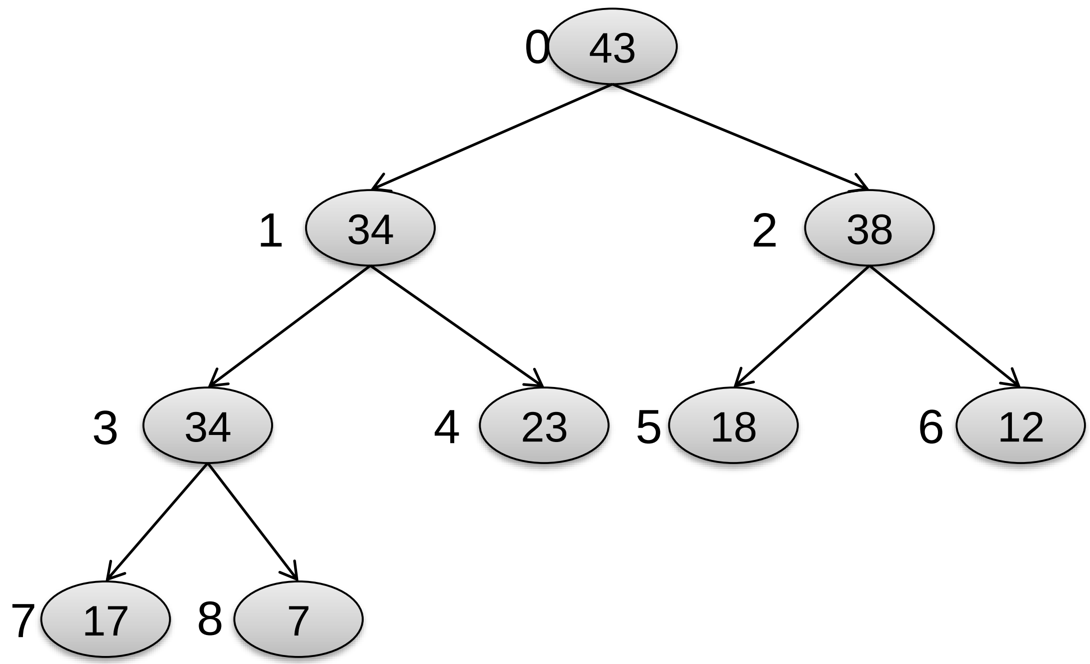
- Bearbeitung Knoten 7: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

43 34 38 34 23 18 12 17 7 <u>45</u>

- Darstellung als Heap



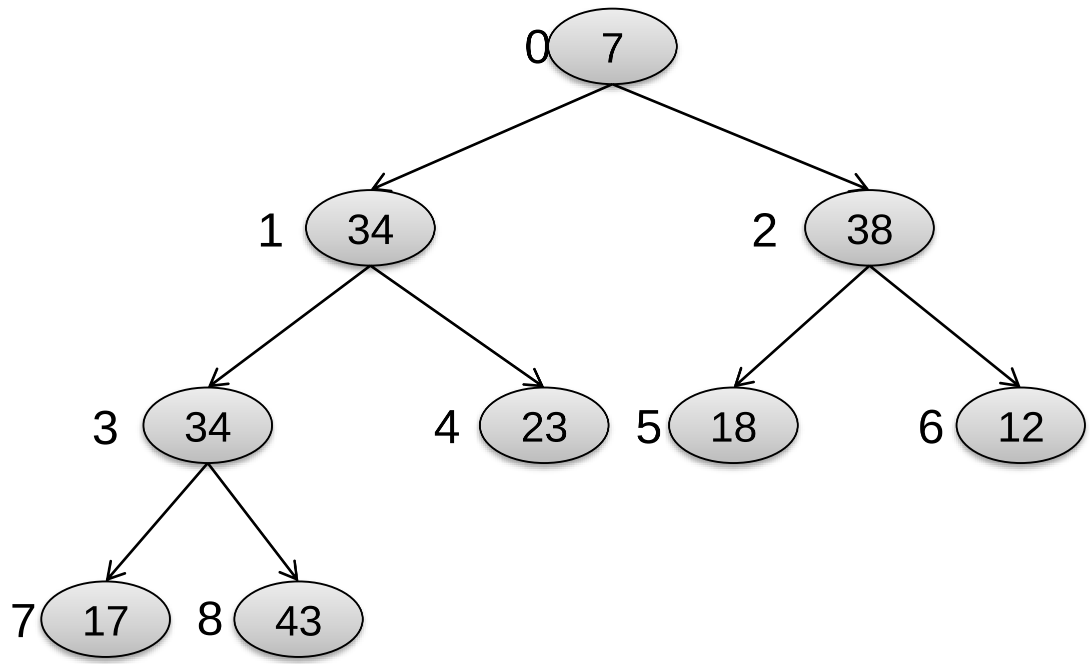
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

7 34 38 34 23 18 12 17 43 <u>45</u>

- Darstellung als Heap



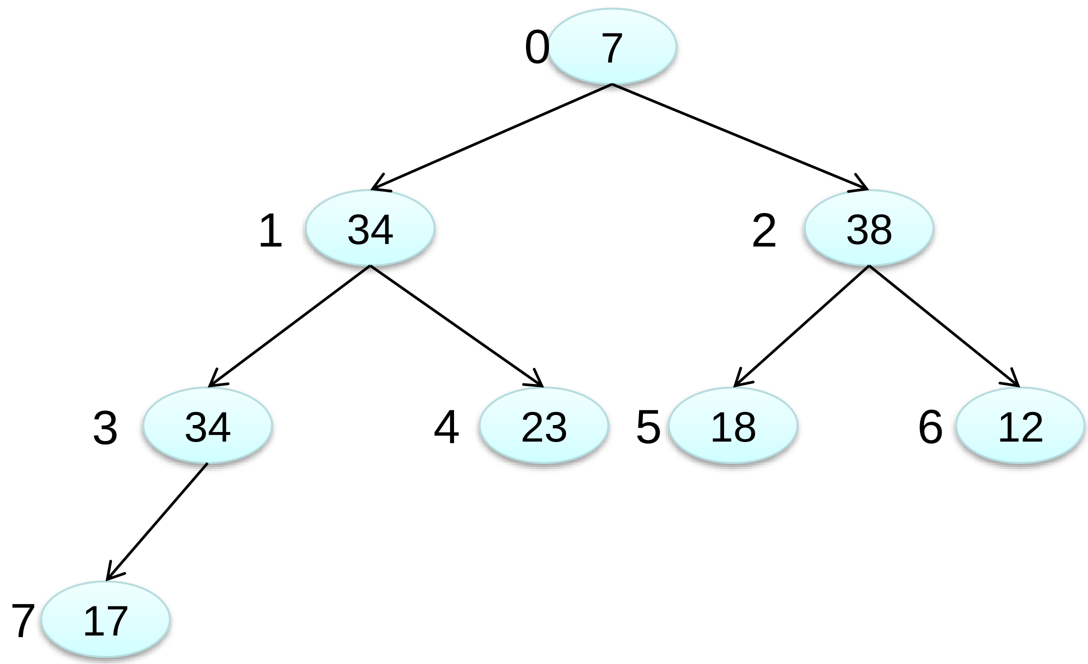
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

7	34	38	34	23	18	12	17	<u>43</u>	<u>45</u>

- Darstellung als Heap



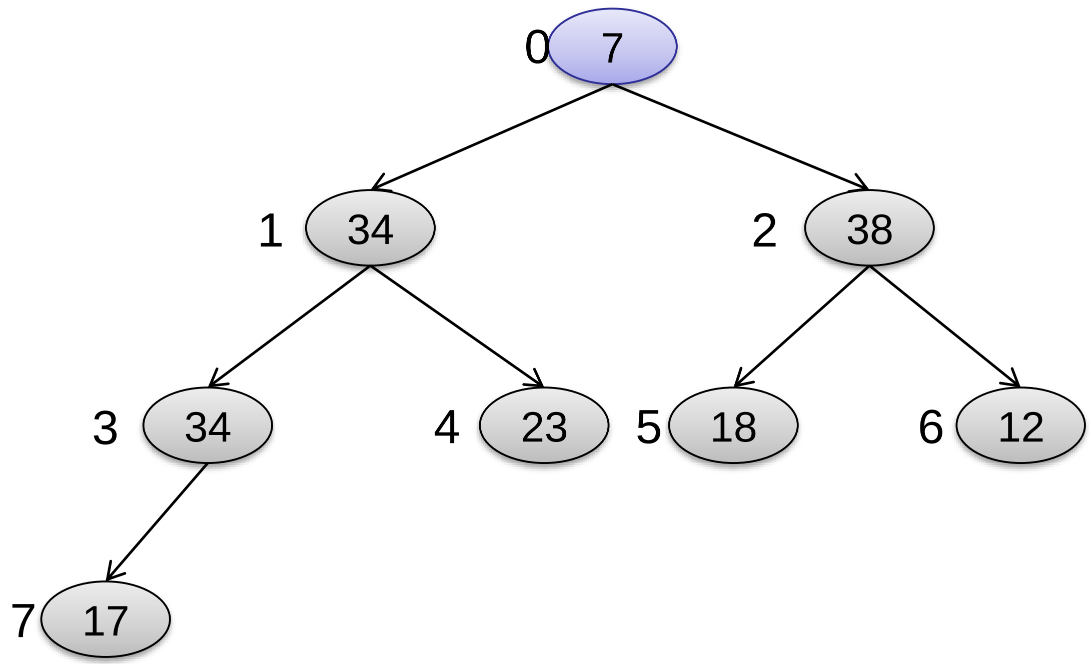
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

  7 34 38 34 23 18 12 17 43 45

- Darstellung als Heap



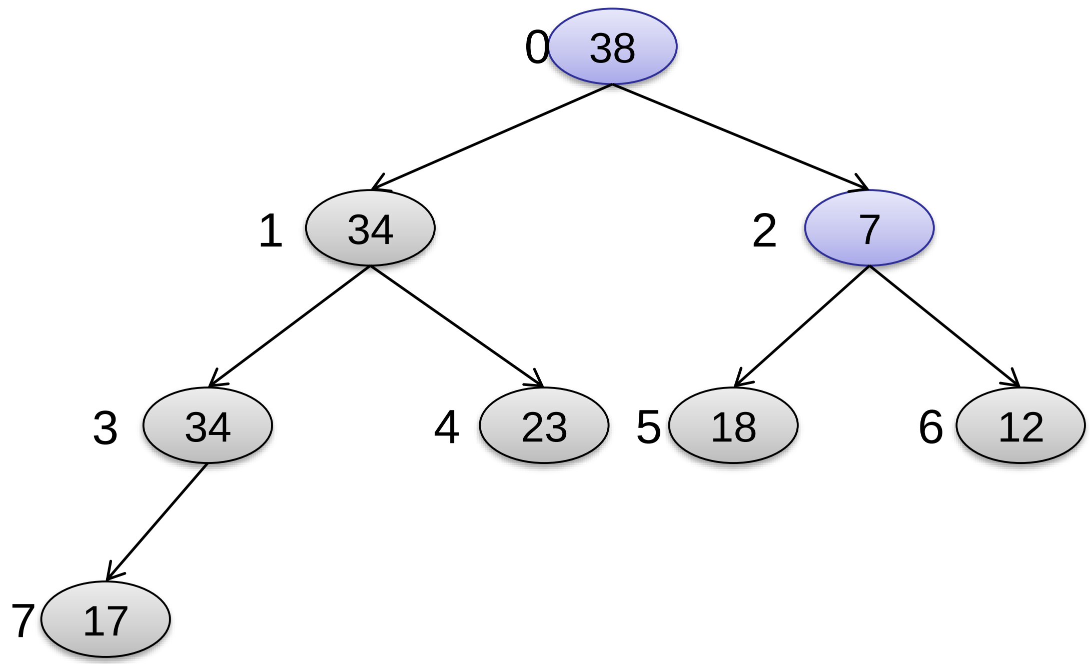
- Bearbeitung Knoten 0: Größtes Element ist der rechte Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  38 34 7 34 23 18 12 17 43 45

- Darstellung als Heap



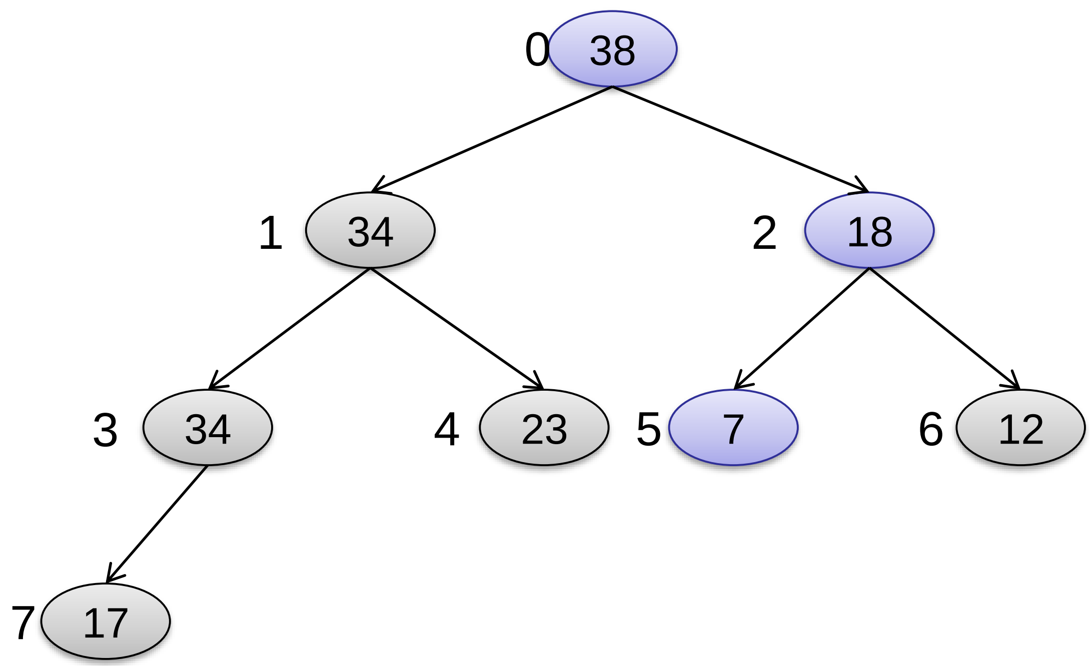
- Bearbeitung Knoten 2: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

38 34 18 34 23 7 12 17 43 45

- Darstellung als Heap



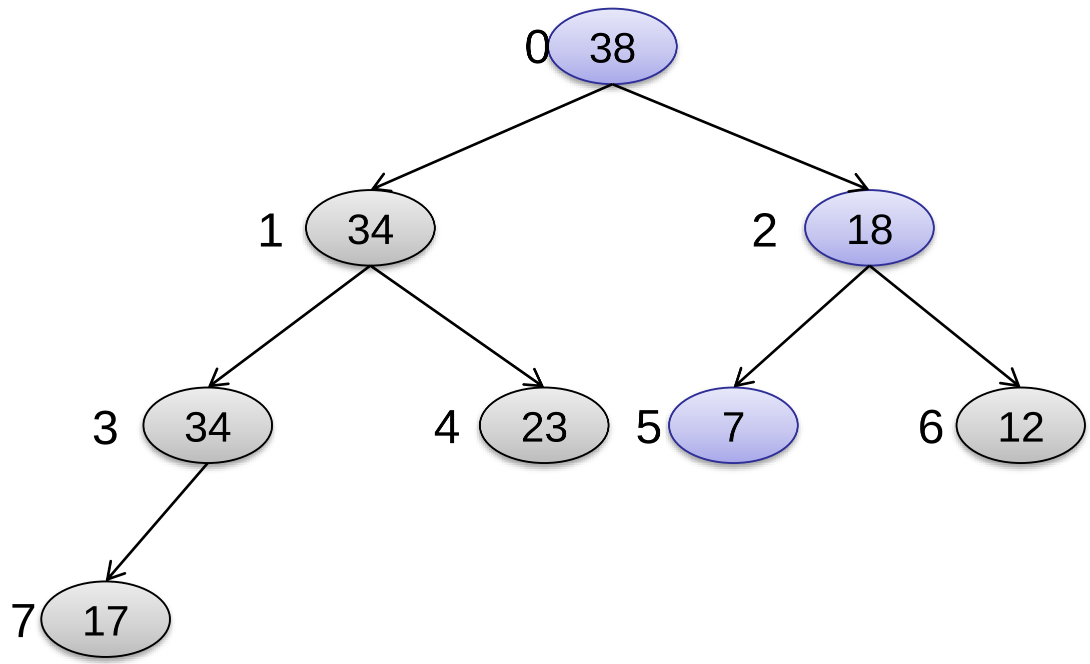
- Bearbeitung Knoten 2: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

38 34 18 34 23 7 12 17 <u>43 45</u>

- Darstellung als Heap



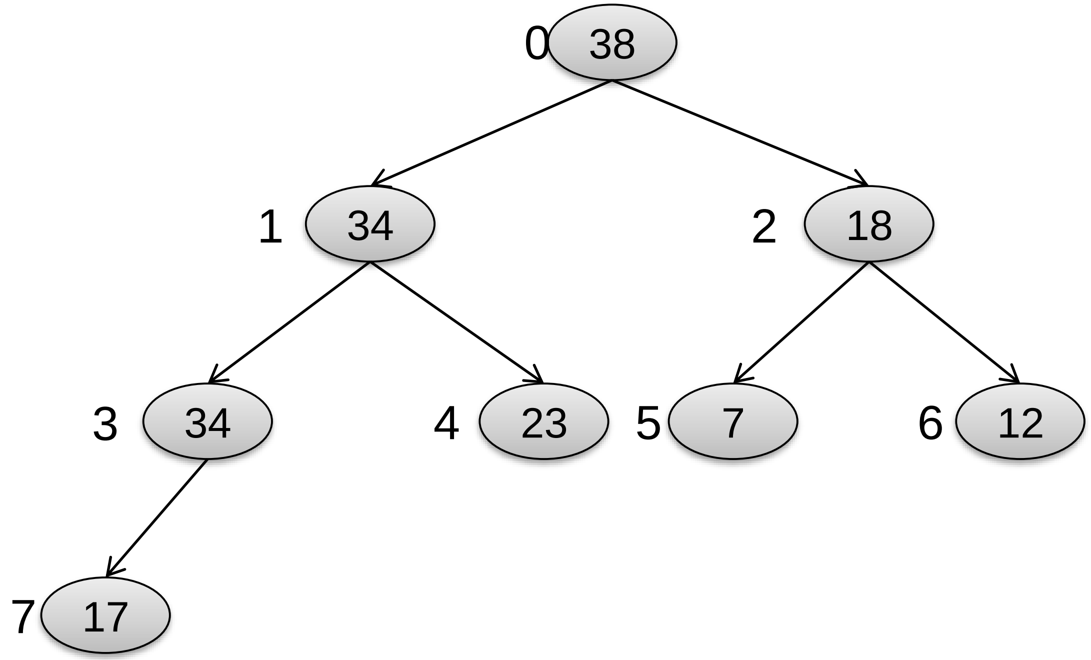
- Bearbeitung Knoten 5: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

  38 34 18 34 23 7 12 17 <u>43 45</u>

- Darstellung als Heap



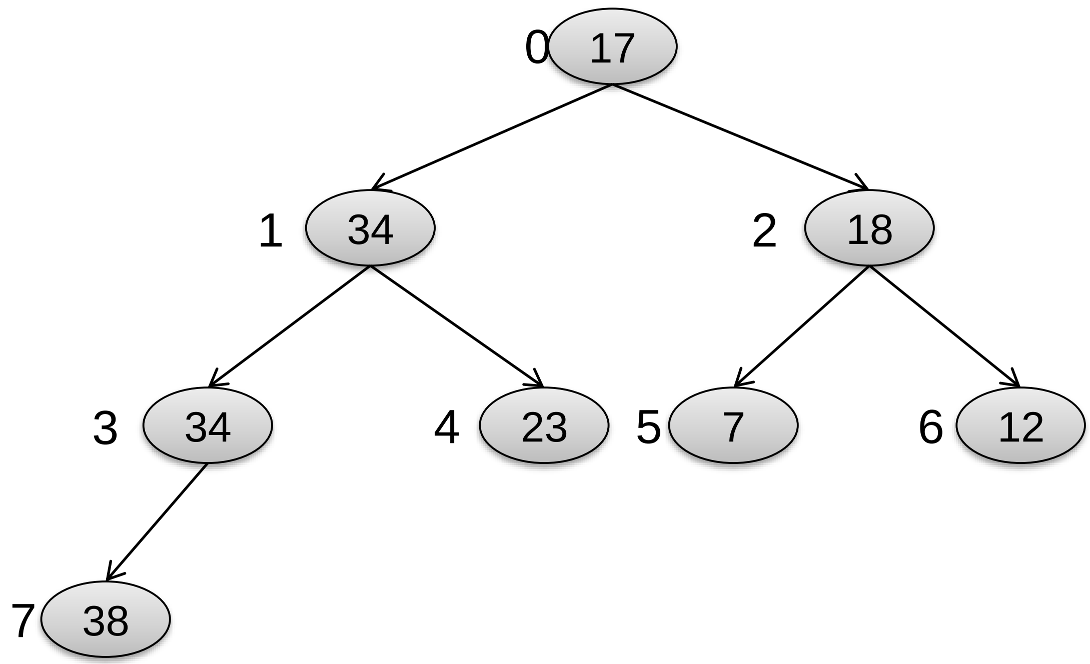
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

  17 34 18 34 23 7 12 38 <u>43 45</u>

- Darstellung als Heap



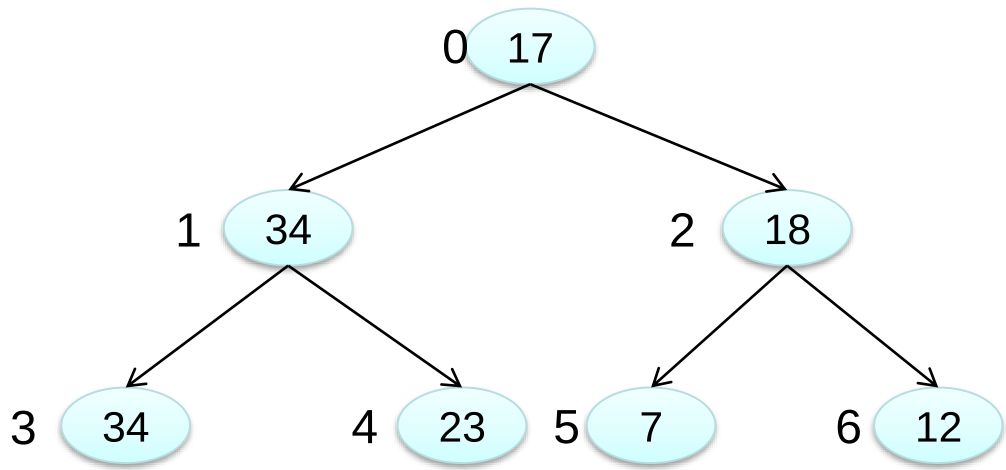
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

17 34 18 34 23 7 12 <u>38 43 45</u>

- Darstellung als Heap



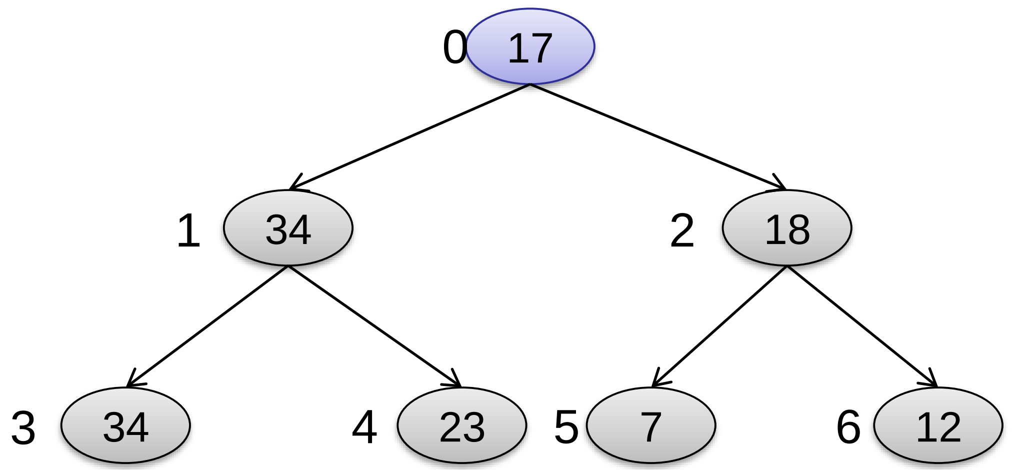
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

  17 34 18 34 23 7 12 <u>38 43 45</u>

- Darstellung als Heap



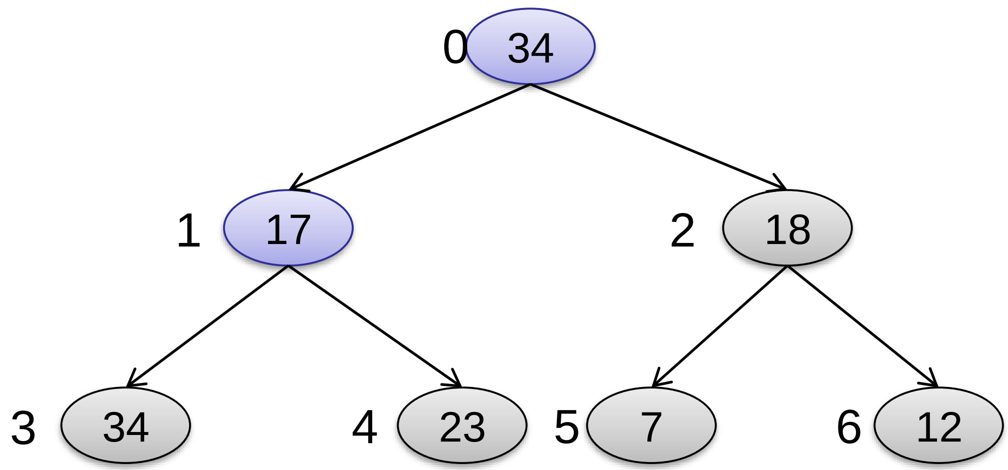
- Bearbeitung Knoten 0: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  34 17 18 34 23 7 12 <u>38 43 45</u>

- Darstellung als Heap



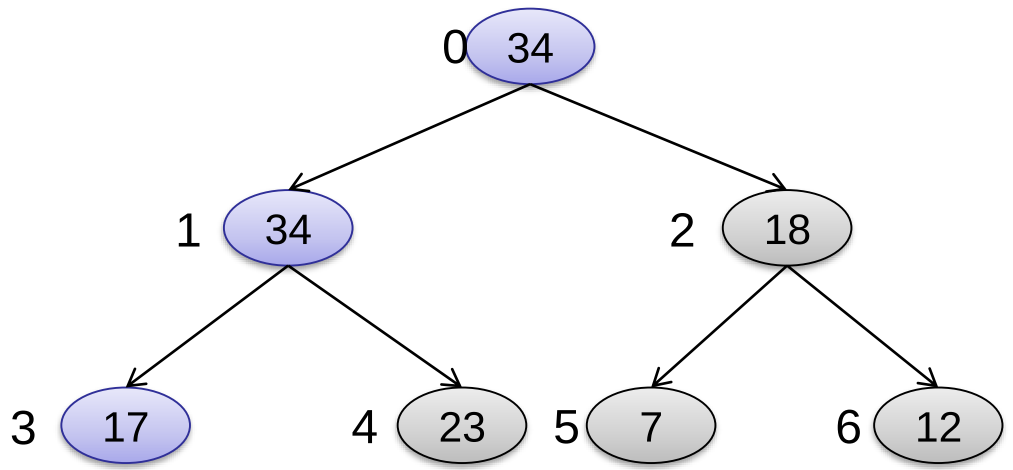
- Bearbeitung Knoten 1: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

34 34 18 17 23 7 12 <u>38 43 45</u>

- Darstellung als Heap



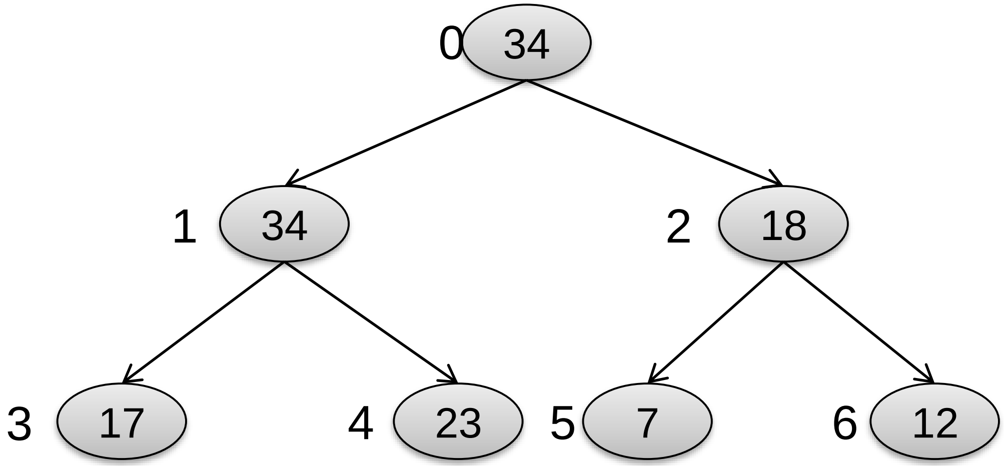
- Bearbeitung Knoten 3: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

  34 34 18 17 23 7 12 <u>38 43 45</u>

- Darstellung als Heap



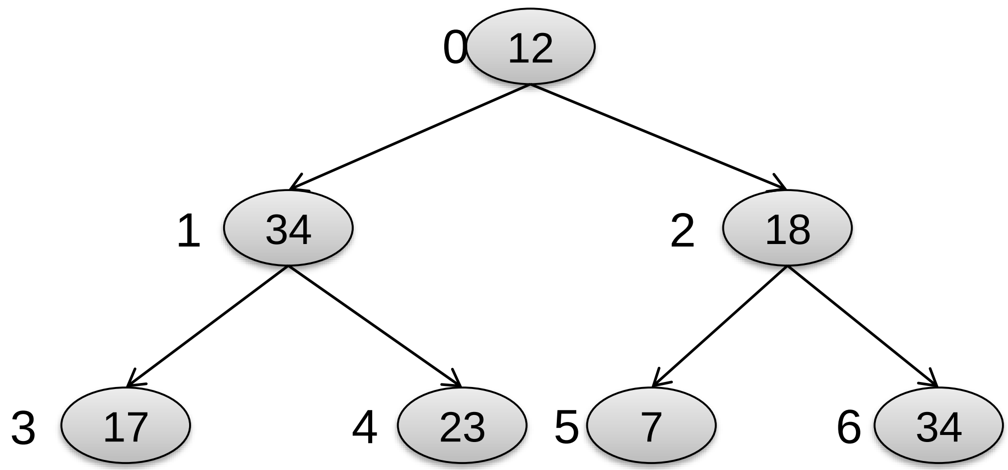
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

  12 34 18 17 23 7 34 <u>38 43 45</u>

- Darstellung als Heap



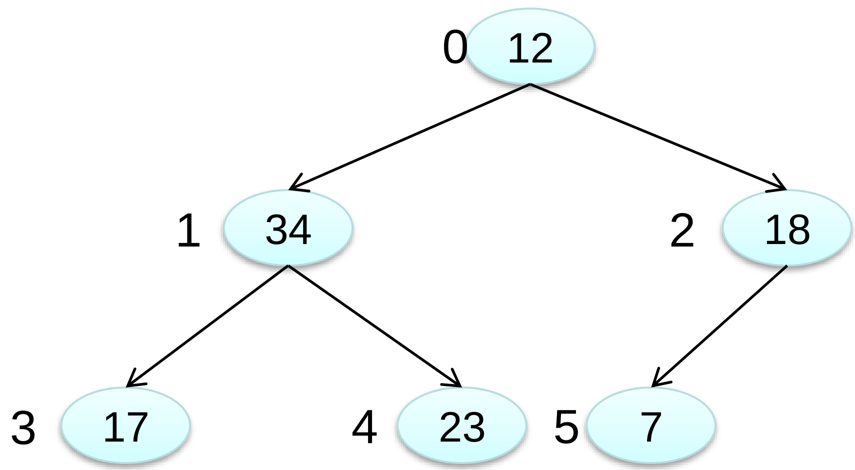
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

  12 34 18 17 23 7 <u>34 38 43 45</u>

- Darstellung als Heap



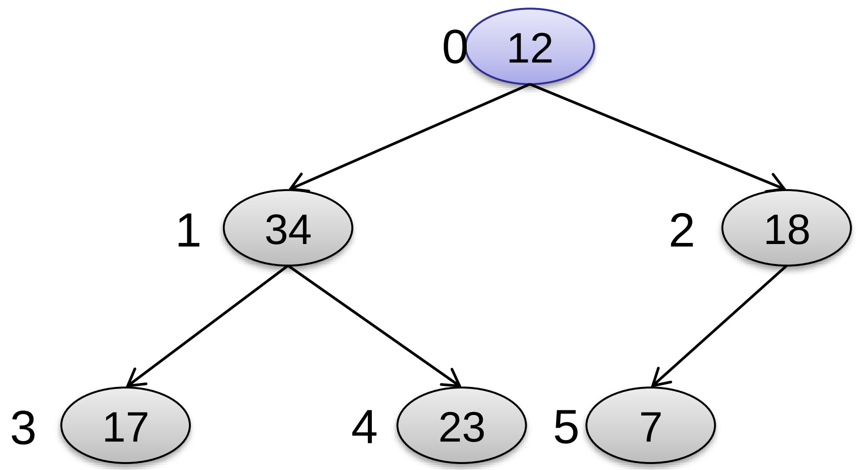
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

12 34 18 17 23 7 <u>34 38 43 45</u>

- Darstellung als Heap



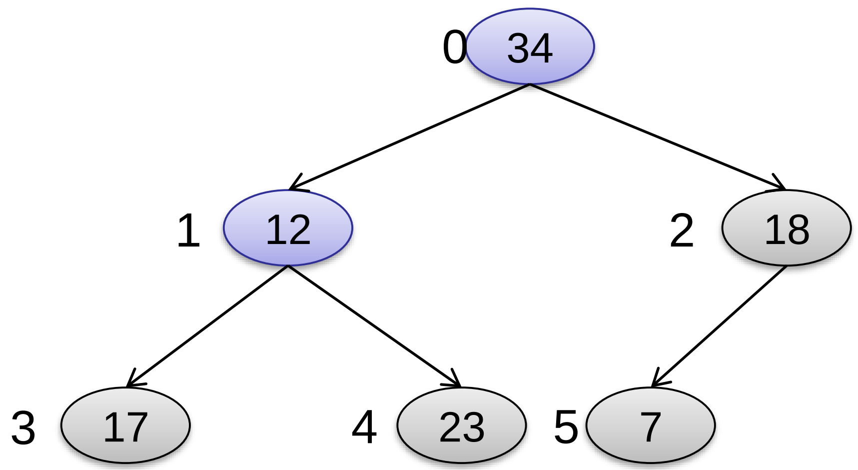
- Bearbeitung Knoten 0: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  34 12 18 17 23 7 <u>34 38 43 45</u>

- Darstellung als Heap



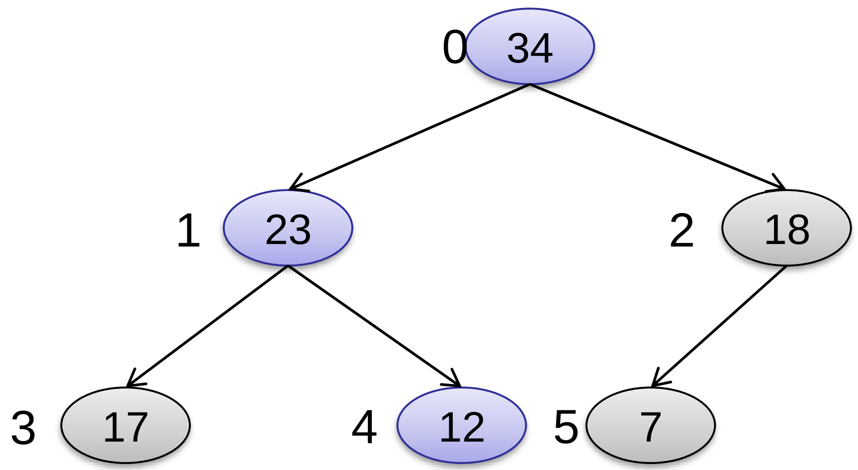
- Bearbeitung Knoten 1: Größtes Element ist der rechte Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

34 23 18 17 12 7 <u>34 38 43 45</u>

- Darstellung als Heap



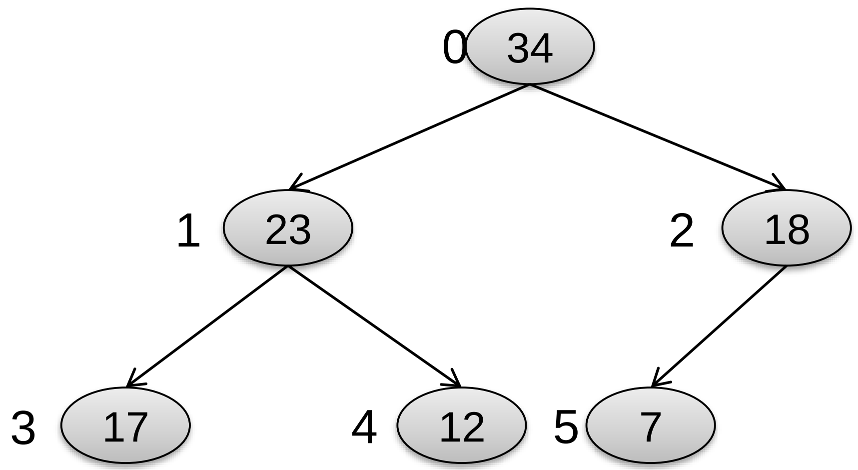
- Bearbeitung Knoten 4: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

34 23 18 17 12 7 <u>34 38 43 45</u>

- Darstellung als Heap



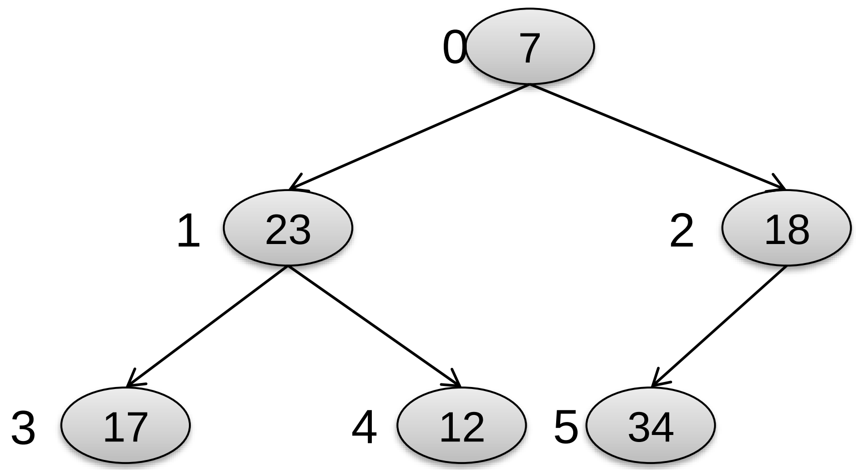
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

7 23 18 17 12 34 <u>34 38 43 45</u>

- Darstellung als Heap



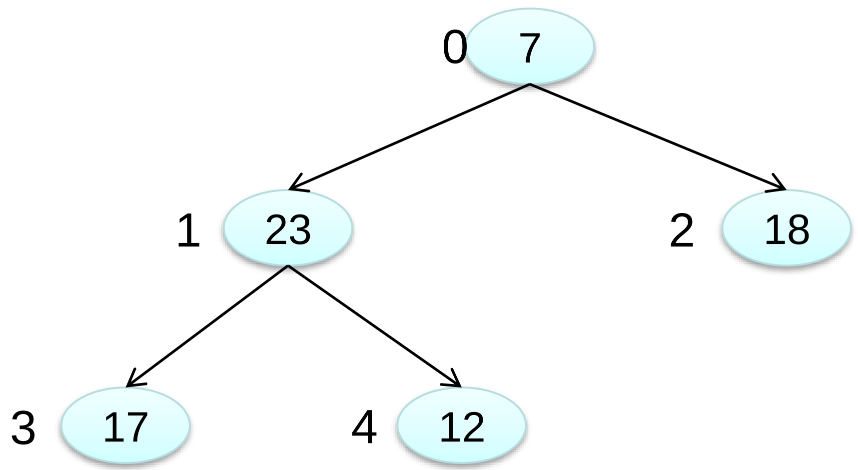
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

7 23 18 17 12 <u>34 34 38 43 45</u>

- Darstellung als Heap



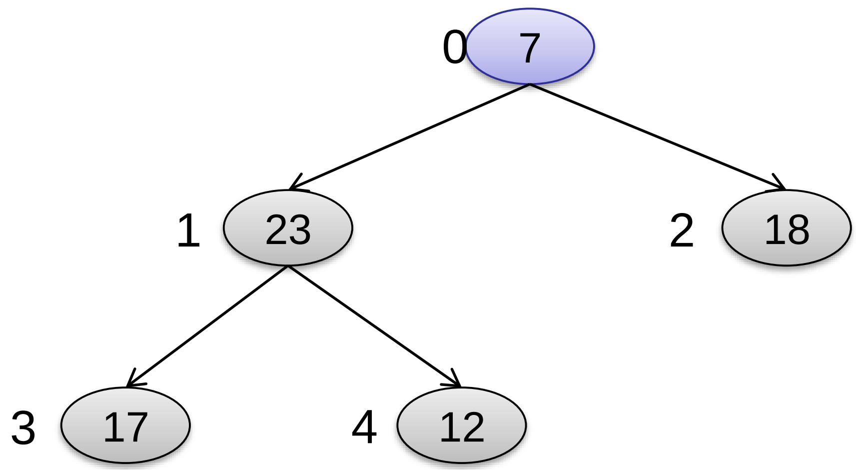
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

7 23 18 17 12 <u>34 34 38 43 45</u>

- Darstellung als Heap



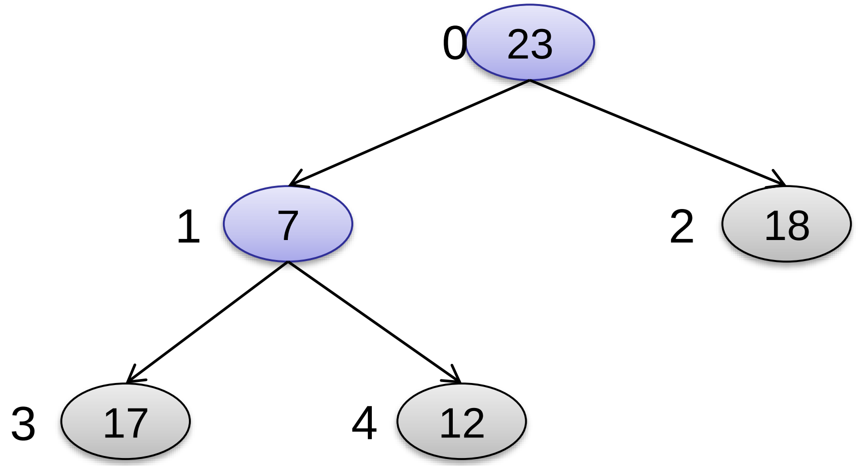
- Bearbeitung Knoten 0: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

23 7 18 17 12 <u>34 34 38 43 45</u>

- Darstellung als Heap



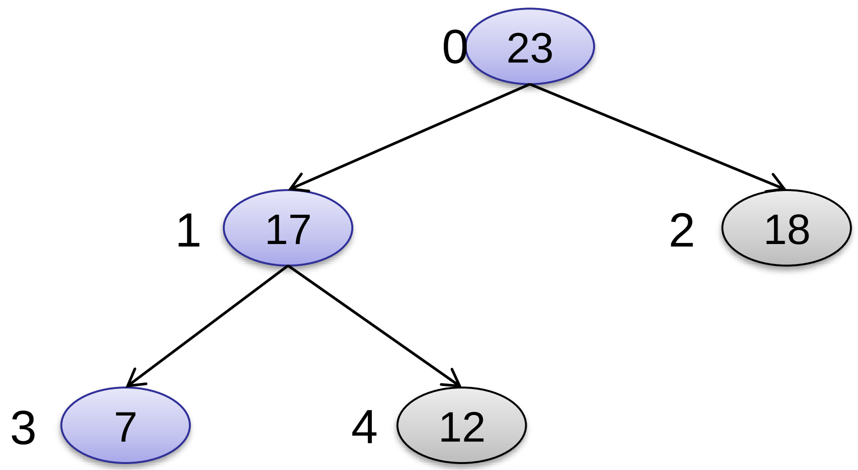
- Bearbeitung Knoten 1: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  23 17 18 7 12 <u>34 34 38 43 45</u>

- Darstellung als Heap



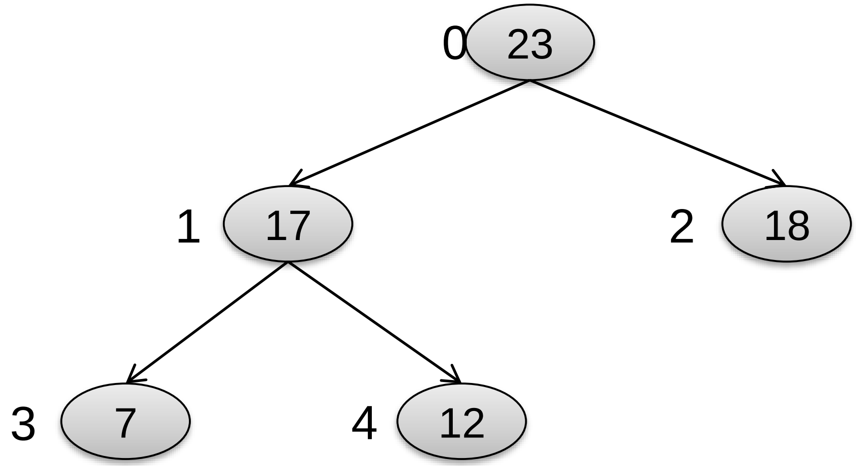
- Bearbeitung Knoten 3: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

  23 17 18 7 12 <u>34 34 38 43 45</u>

- Darstellung als Heap



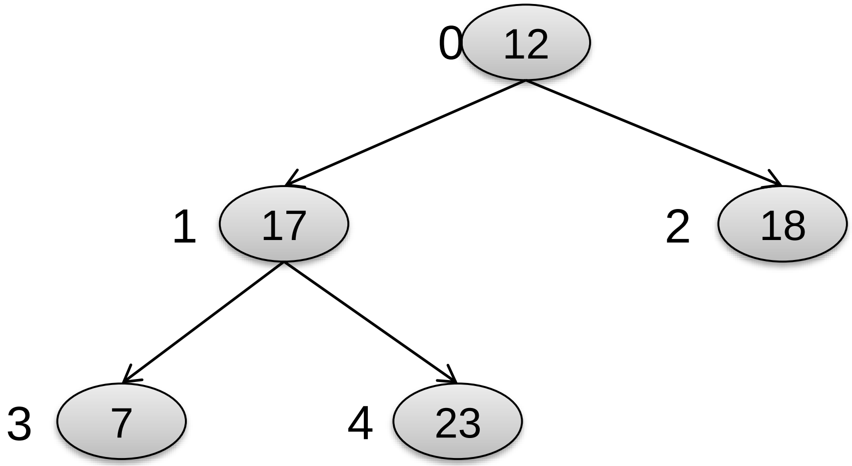
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

  12 17 18 7 23 <u>34 34 38 43 45</u>

- Darstellung als Heap



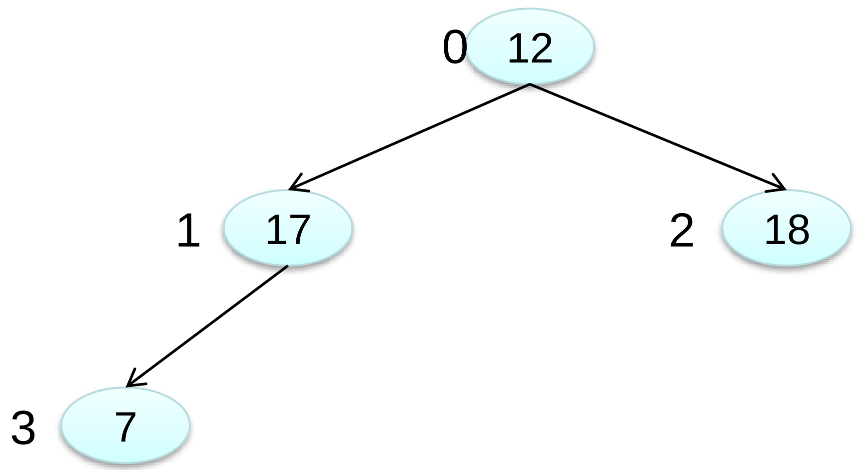
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

12 17 18 7 <u>23 34 34 38 43 45</u>

- Darstellung als Heap



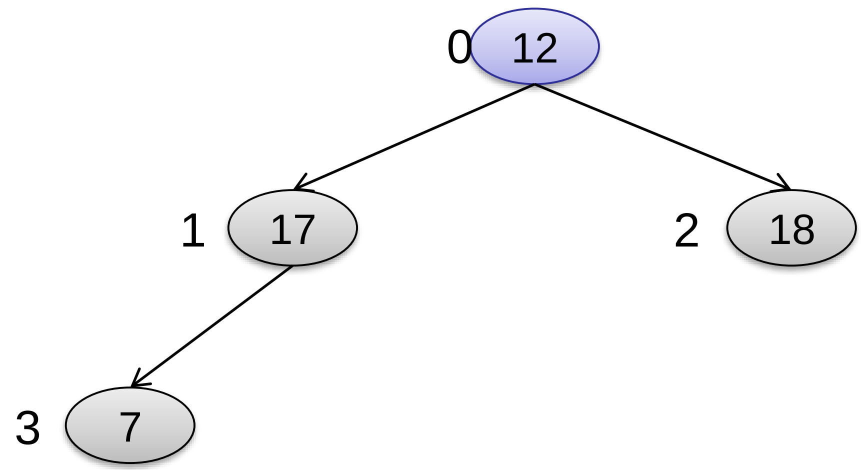
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

12 17 18 7 <u>23 34 34 38 43 45</u>

- Darstellung als Heap



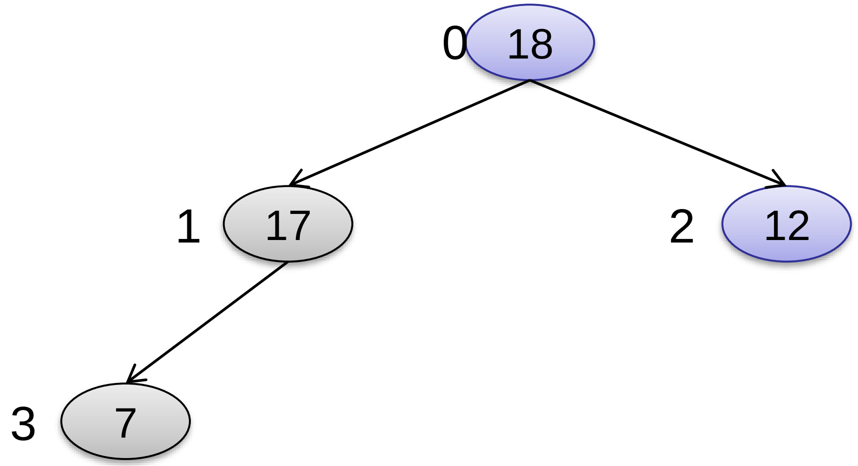
- Bearbeitung Knoten 0: Größtes Element ist der rechte Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

  18 17 12 7 <u>23 34 34 38 43 45</u>

- Darstellung als Heap



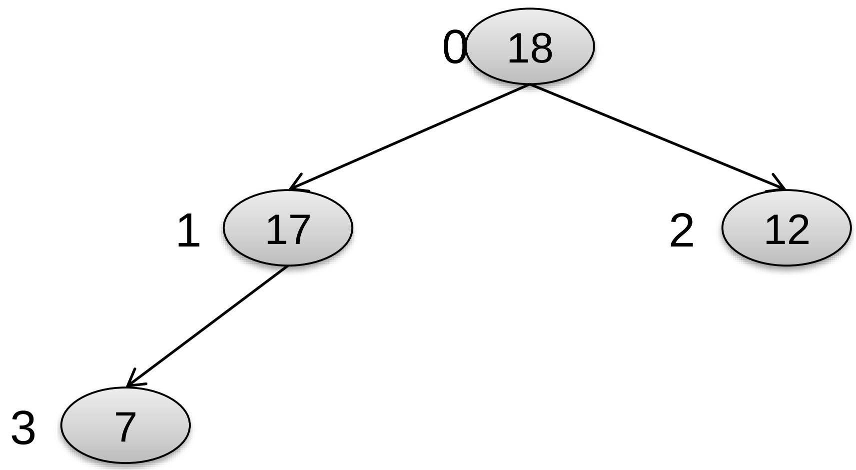
- Bearbeitung Knoten 2: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

  18 17 12 7 <u>23 34 34 38 43 45</u>

- Darstellung als Heap



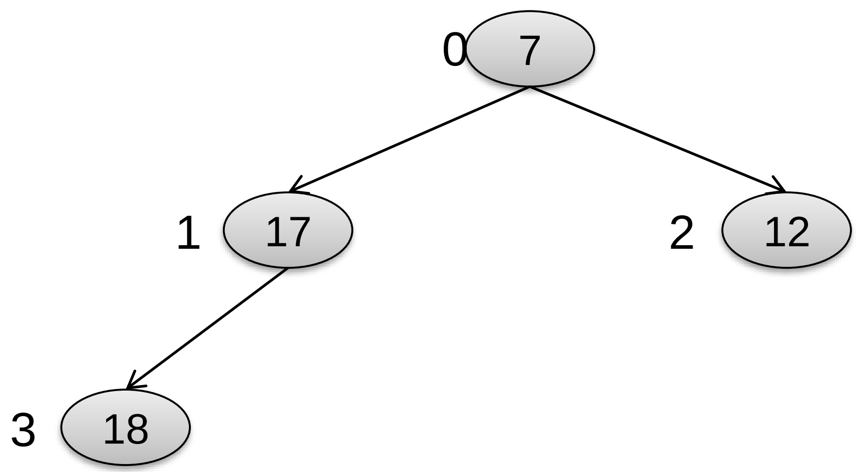
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

7 17 12 18 23 34 34 38 43 45

- Darstellung als Heap



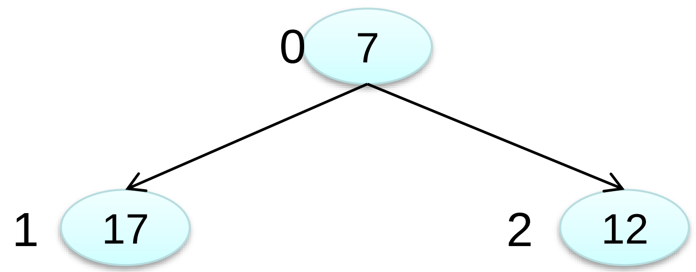
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

7 17 12 18 23 34 34 38 43 45

- Darstellung als Heap



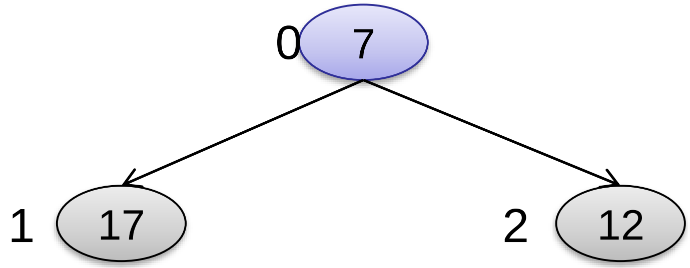
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

7 17 12 18 23 34 34 38 43 45

- Darstellung als Heap



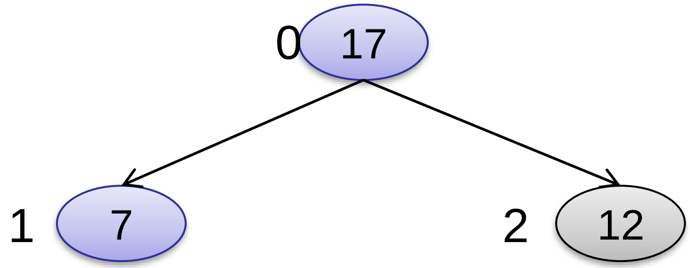
- Bearbeitung Knoten 0: Größtes Element ist der linke Nachfolger

Beispiel Heap Sort

- Aktuelles Feld

17 7 12 18 23 34 34 38 43 45

- Darstellung als Heap



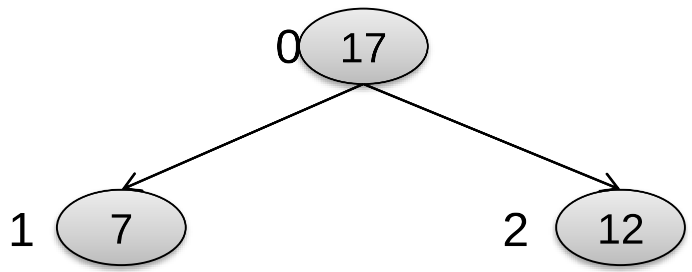
- Bearbeitung Knoten 1: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

17 7 12 18 23 34 34 38 43 45

- Darstellung als Heap



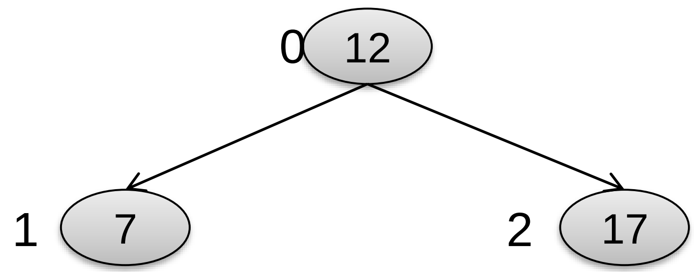
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

12 7 17 <u>18 23 34 34 38 43 45</u>

- Darstellung als Heap



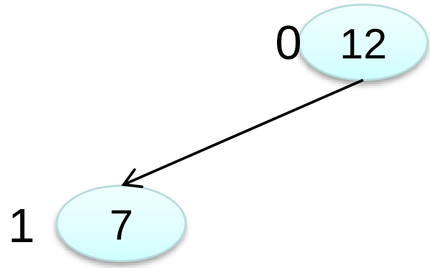
- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

12 7 <u>17 18 23 34 34 38 43 45</u>

- Darstellung als Heap



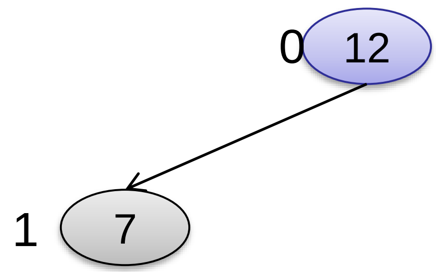
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

12 7 <u>17 18 23 34 34 38 43 45</u>

- Darstellung als Heap



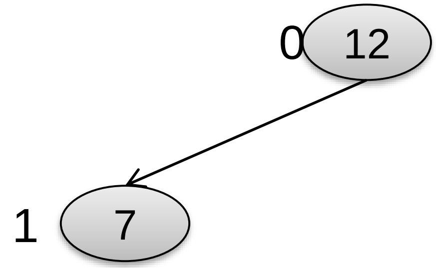
- Bearbeitung Knoten 0: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

12 7 <u>17 18 23 34 34 38 43 45</u>

- Darstellung als Heap



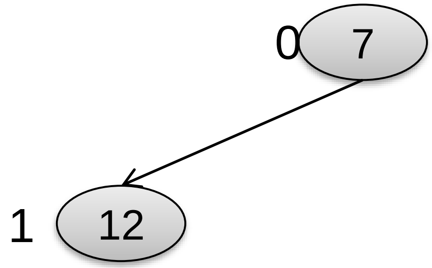
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

7 12 <u>17 18 23 34 34 38 43 45</u>

- Darstellung als Heap



- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

  7 12 17 18 23 34 34 38 43 45

- Darstellung als Heap

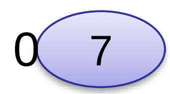
- Maximum ist an richtiger Stelle, Fortsetzung ohne das letzte Element

Beispiel Heap Sort

- Aktuelles Feld

  7 12 17 18 23 34 34 38 43 45

- Darstellung als Heap



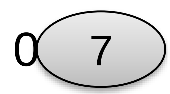
- Bearbeitung Knoten 0: Größtes Element ist die Wurzel

Beispiel Heap Sort

- Aktuelles Feld

7 12 17 18 23 34 34 38 43 45

- Darstellung als Heap



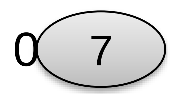
- Nächster Wert ist identifiziert

Beispiel Heap Sort

- Aktuelles Feld

  7 12 17 18 23 34 34 38 43 45

- Darstellung als Heap



- Erstes und aktuell letztes Element werden vertauscht

Beispiel Heap Sort

- Aktuelles Feld

7	12	17	18	23	34	34	38	43	45

- Darstellung als Heap

Kein weiteres Element verfügbar!

- Maximum ist an richtiger Stelle, HeapSort ist beendet

Beispiele Heap Sort

- Welche der folgenden Felder stellen einen Heap dar:
  - $a[] = \{19, 5, 7, 1, 2, 6, 0\}$
  - $b[] = \{12, 5, 7, 1, 2, 6, 0\}$
  - $c[] = \{15, 5, 7, 1, 2, 6, 8\}$
  - $d[] = \{23, 5, 7, 1, 2, 7, 2\}$
- Sortieren Sie das Feld $e[] = \{2, 4, 9, 18, 21, 37\}$ mittels HeapSort!