Fachhochschule Nordwestschweiz

05 Priority Queues (Teil 1)

Algorithmen und Datenstrukturen 2

2 Vorlesungen

Dokumente

- Folien
- Skript
- Programmieraufgaben 1 und 2
- Arbeitsblatt

Bekannte Datenstrukturen mit Zugriffsbeschränkungen

pop()

entfernt *zuletzt* eingefügtes Element

dequeue()

entfernt *zuerst* eingefügtes Element

Fachhochschule
Nordwestschweiz

Priority Queue - Prioritätswarteschlange

Eine **Priority Queue** (auch Prioritäts- oder Vorrangwarteschlange) ist eine abstrakte Datenstruktur, in der die Ausgabe-Reihenfolge von einem **Schlüssel** der Elemente abhängt, der sogenannten **Priorität**.

Die Elemente werden nach ihrer Priorität sortiert aus der Priority Queue entnommen. Die Ausgabe-Reihenfolge der Priority Queue wird deshalb durch **HIFO** (highest priority in – first out) beschrieben.

In einer Priority Queue müssen die Elemente nicht zwingend vollständig sortiert sein, da nur jeweils das Element mit der höchsten Priorität interessiert.

Die drei wichtigsten Operationen einer Priority Queue, sind das **Hinzufügen** eines Elements, das **Betrachten** des Elements mit der höchsten Priorität und das **Löschen** ebendieses Elements.

Wir unterscheiden zwischen zwei Arten von Priority Queues, je nach dem ob das Element mit dem kleinsten oder dem grössten Schlüssel die höchste Priorität hat.

Priority Queue - Prioritätswarteschlange

add(elem)	fügt ein Element elem hinzu
min	gibt das Element mit dem kleinsten Schlüssel zurück ohne die Priority Queue zu verändern
removeMin	entfernt das Element mit dem kleinsten Schlüssel aus der Priority Queue und gibt es zurück
size	gibt die Anzahl Elemente in der Priority Queue

Priority Queue - Prioritätswarteschlange

Minimum Priority Queue

add(elem)	fügt ein Element elem hinzu
min	gibt das Element mit dem kleinsten Schlüssel zurück ohne die Priority Queue zu verändern
removeMin	entfernt das Element mit dem kleinsten Schlüssel aus der Priority Queue und gibt es zurück
size	gibt die Anzahl Elemente in der Priority Queue

Priority Queue - Prioritätswarteschlange

Minimum Priority Queue

add(elem)	fügt ein Element elem hinzu
min	gibt das Element mit dem kleinsten Schlüssel zurück ohne die Priority Queue zu verändern
removeMin	entfernt das Element mit dem kleinsten Schlüssel aus der Priority Queue und gibt es zurück
size	gibt die Anzahl Elemente in der Priority Queue

Maximum Priority Queue

max	gibt das Element mit dem grössten Schlüssel zurück ohne die Priority Queue zu verändern
removeMax	entfernt das Element mit dem grössten Schlüssel aus der Priority Queue und gibt es zurück

Prioritäten und Schlüssel

Eine Priority Queue kann mit allen möglichen Arten von Elementen und Schlüsseln verwendet werden. Wir gehen hier auf zwei Möglichkeiten etwas genauer ein.

1. *Elemente mit natürlicher oder vordefinierter Ordnung*

- Die Reihenfolge, in der die Elemente verarbeitet werden sollen, hängt von einem ihrer Attribute ab.
- Dieses Attribut dient dann als Schlüssel.
- Für diese Schlüssel muss eine Ordnungsrelation definiert sein. (Sind die Schlüssel Zahlen, besitzen sie bereits eine natürliche Ordnung. Kompliziertere Schlüssel brauchen eine Vergleichsoperation.)

Prioritäten und Schlüssel

Beispiele:

Bei einfachen Elementen wie Zahlen, ist das Element gerade auch der Schlüssel und dadurch die natürliche Ordnung bereits gegeben.

Bei Objekten wie zum Beispiel Clubmitgliedern, die mit ‘Name’, ‘Adresse’ und ‘Anzahl Jahre der Mitgliedschaft’ mehrere Attribute haben, muss der Schlüssel definiert werden. Möchten wir jeweils das Mitglied mit der längsten Clubmitgliedschaft kennen, wählen wir ‘Anzahl Jahre der Mitgliedschaft’ als Schlüssel. Weil dieser Schlüssel eine Zahl ist, können die Clubmitglieder nun miteinander verglichen werden.

Komplexere Schlüssel kommen zum Beispiel als Rollen in Hierarchien vor: Professoren, Assistenten, Studenten an der Hochschule; CEO, Abteilungsleiter, Teamleiter im Büro; Dienstgrade in der Armee. Damit diese Rollen miteinander verglichen werden können muss eine Vergleichsoperation definiert werden, die zum Beispiel der Rolle eines Generals eine höhere Priorität zuweist als der Rolle eines Hauptmanns.

Prioritäten und Schlüssel

Eine mögliche Schnittstelle einer Minimum Priority Queue in Java könnte also wie folgt aussehen:

```
public interface MinPriorityQueue<K extends Comparable<? super K>> {
      boolean add(K element);
      K min();
      K removeMin();
      int size();
}
```

Für Priority Queues gilt in der Regel, dass alles erlaubt ist: Sowohl Duplikate im eigentlichen Sinne als auch mehrere Elemente mit gleicher Priorität.

Prioritäten und Schlüssel

2. Elemente mit zugewiesenen Prioritäten

Je nach Anwendung kommt es auch vor, dass ein Element erst beim Hinzufügen zur Priority Queue eine Priorität zugewiesen erhält. In diesem Fall kann die add Methode dahingehend angepasst werden, dass sie zwei Argumente annimmt: Einmal das Element selbst und einmal seine Priorität.

```
boolean add(K element, long priority);
```

Aufgabe

In dieser Aufgabe nehmen wir an, dass wir Ganzzahlen als Elemente haben, die jeweils ihrer Grösse nach sortiert aus der Priority Queue gelesen werden.

a) Auf einer anfänglich leeren Minimum Priority Queue werden die untenstehenden Operationen ausgeführt. Schreiben Sie unter jedes min und removeMin, welches Element zurückgegeben wird.

add(4), add(5), min, add(3), removeMin, removeMin, add(8), removeMin, removeMin.

a) Auf einer anfänglich leeren Minimum Priority Queue werden die untenstehenden Operationen ausgeführt. Schreiben Sie unter jedes max und removeMax, welches Element zurückgegeben wird.

add(4), add(5), max, add(3), removeMax, removeMax, add(8), removeMax, removeMax.

Priority Queue - Implementierung mit bekannten Datenstrukturen

Aufgabe: Füllen Sie die folgende Tabelle mit dem dem jeweiligen asymptotischen Laufzeitaufwand (in O-Notation) im Worst Case aus:

	Array		**Linked List**		**Binärbaum**	
Operation	**sortiert**	**unsortiert**	**sortiert**	**unsortiert**	**allgemein**	**AVL**
add(element)						
min()						
removeMin()						

Priority Queue - Implementierung mit bekannten Datenstrukturen

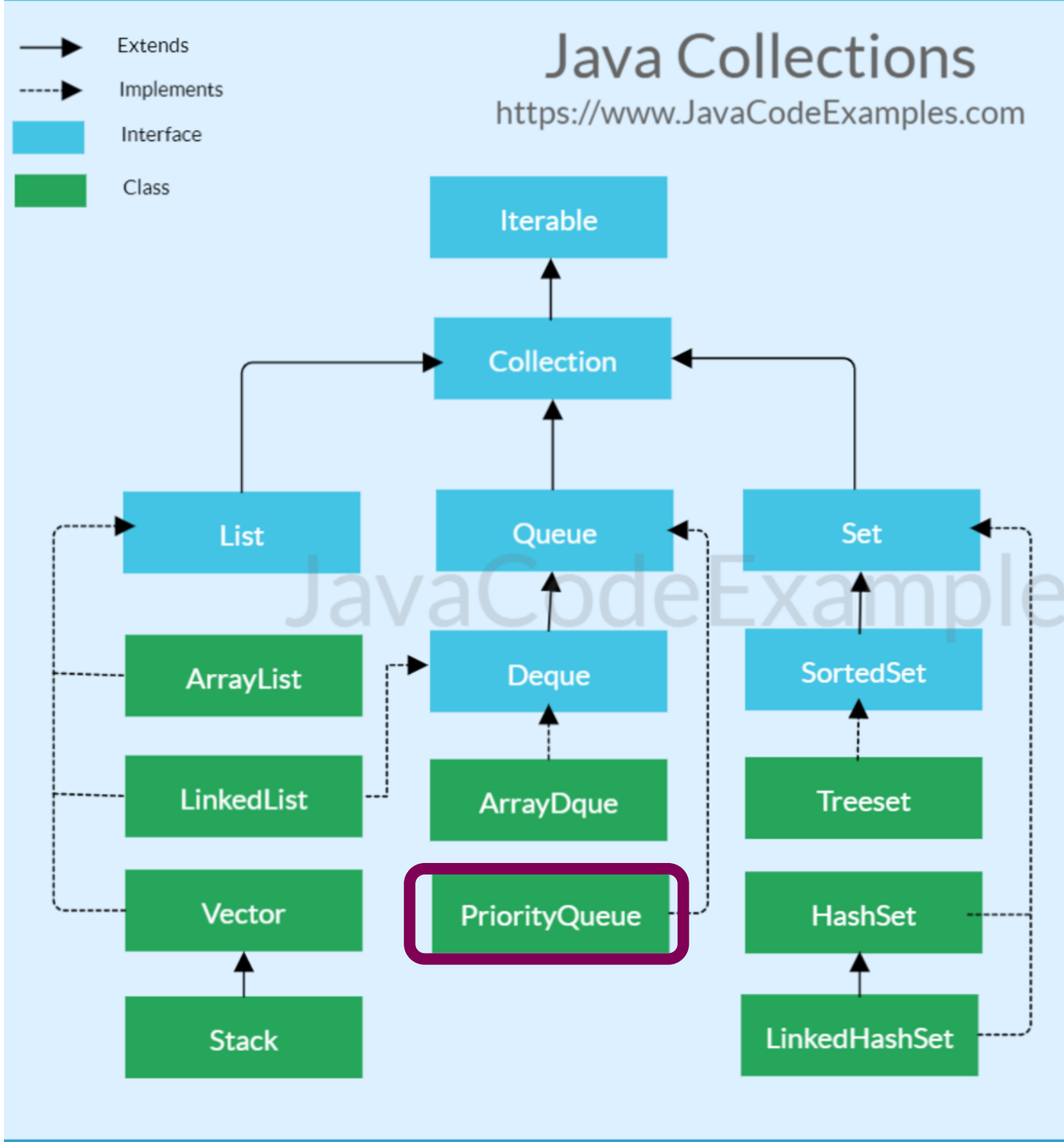
Aufgabe: Füllen Sie die folgende Tabelle mit dem dem jeweiligen asymptotischen Laufzeitaufwand (in O-Notation) im Worst Case aus:

	Array		**Linked List**		**Binärbaum**	
Operation	**sortiert**	**unsortiert**	**sortiert**	**unsortiert**	**allgemein**	**AVL**
add(element)	O(n)	O(1)	O(n)	O(1)	O(n)	O(log n)
min()	O(1)	O(n) *	O(1)	O(n) *	O(n) *	O(log n) *
removeMin()	O(1) rückwärts	O(n)	O(1)	O(n)	O(n)	O(log n)

* falls Pointer auf das kleinste: O(1) für min()

Fachhochschule Nordwestschweiz

PriorityQueue in Java Collection Framework



Aufgabe (PriorityQueue in Java Collection Framework)

Im Java Collection Framework gibt es eine Klasse PriorityQueue. Betrachten Sie die Spezifikation dieser Klasse und beantworten Sie die folgenden Fragen dazu.

a) Handelt es sich bei der Java-Klasse PriorityQueue um eine Minimum oder Maximum Priority Queue?

b) Welche Operationen der Java-Klasse PriorityQueue entsprechen unseren Methoden Hinzufügen, Betrachten und Löschen?

c) Wie könnten Sie die Java-Klasse PriorityQueue benützen, damit sie sich von aussen betrachtet wie eine Maximum Priority Queue verhält?

d) Sind in der Java-Klasse PriorityQueue Duplikate erlaubt? Werden Duplikate in einer bestimmten Reihenfolge behandelt?

https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/PriorityQueue.html

Aufgabe (PriorityQueue in Java Collection Framework)

a) Handelt es sich bei der Java-Klasse PriorityQueue um eine Minimum oder Maximum Priority Queue?

Die Java-Klasse ist als Minimum Priority Queue implementiert. Die Spezifikation sagt dazu: «The head of this queue is the least element with respect to the specified ordering.»

a) Welche Operationen der Java-Klasse PriorityQueue entsprechen unseren Methoden Hinzufügen, Betrachten und Löschen?

Fürs Hinzufügen gibt es add und offer. Diese unterscheiden sich untereinander durch das Werfen von Exceptions. Betrachten und Löschen werden vom Interface der Queue übernommen. Unser min heisst hier also peek; unser removeMin entspicht dem poll.

Aufgabe (PriorityQueue in Java Collection Framework)

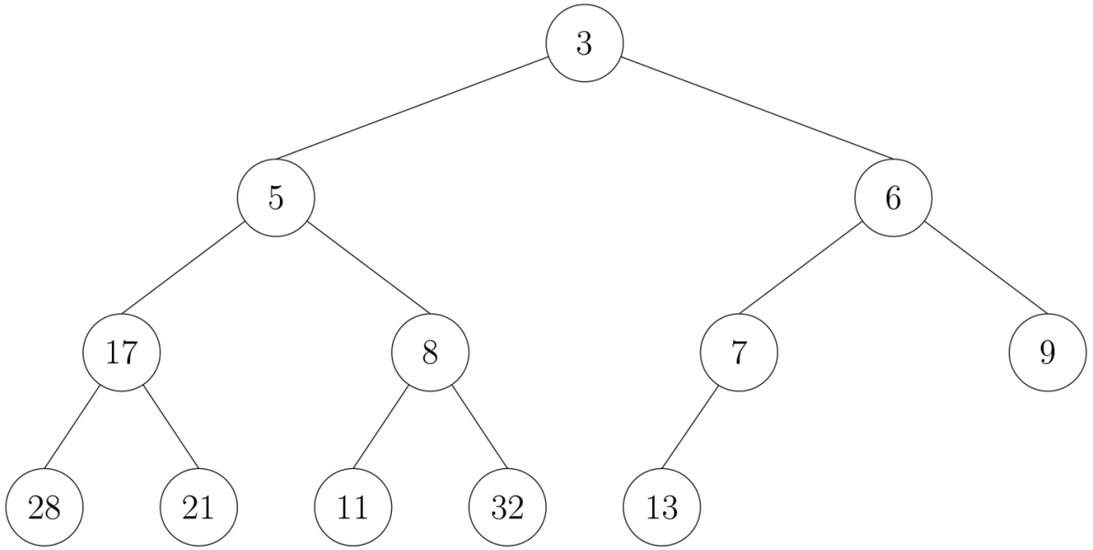
c) Die könnten Sie die Java-Klasse PriorityQueue benützen, damit sie sich von aussen betrachtet wie eine Maximum Priority Queue verhält?

   Die einfachste Möglichkeit ist es, beim Erstellen der PriorityQueue einen Comparator zu übergeben, der die Reihenfolge der Elemente genau umdreht.

c) Sind in der Java-Klasse PriorityQueue Duplikate erlaubt? Werden Duplikate in einer bestimmten Reihenfolge behandelt?

   Gleiche Schlüssel (und auch gleiche Elemente) sind erlaubt. Einerseits steht in der Spezifikation: «If multiple elements are tied for least value, the head is one of those elements – ties are broken arbitrarily.» Das bedeutet als, dass es mehrere Elemente mit gleichem Schlüssel geben könnte. Ausserdem bedeutet dies, dass die Reihenfolge, in welcher Elemente mit gleichem Schlüssel ausgegeben werden nicht festgelegt (also arbitrarily) ist. Andererseits sieht man auch an der Spezifikation von add, das immer true zurückgibt, dass identische Schlüssel also erlaubt sind.

Neue Datenstruktur **– Der Min-Heap**



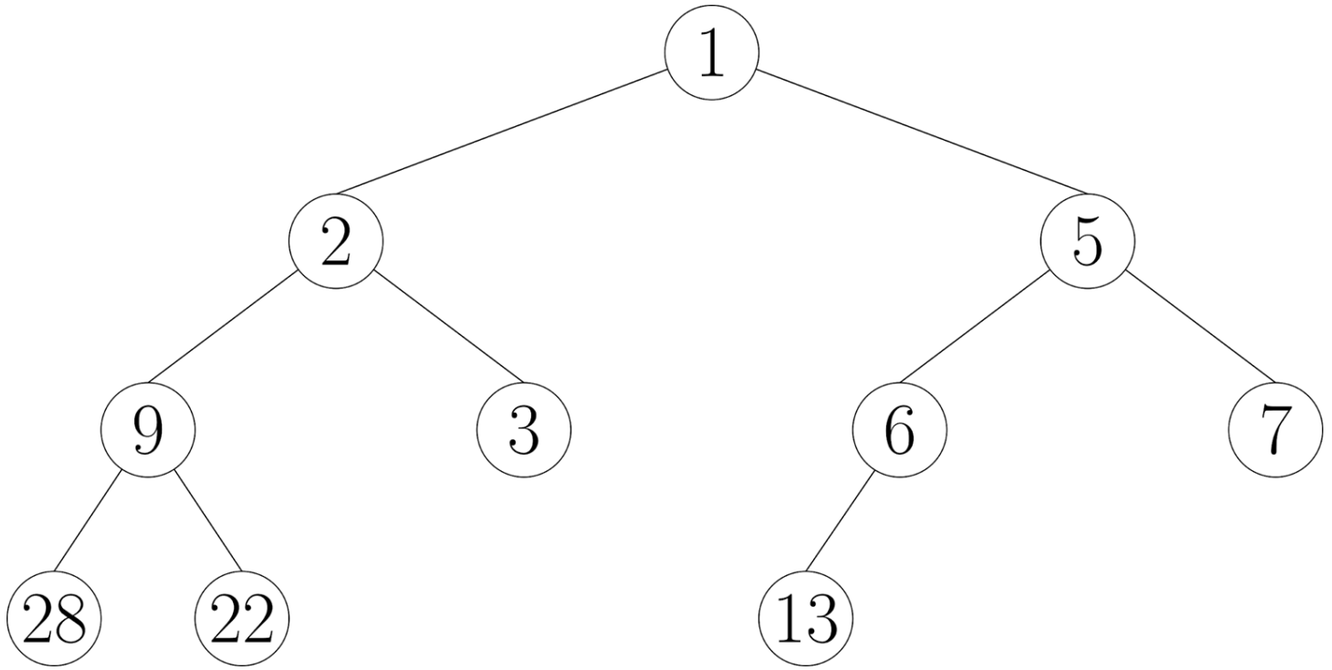
Struktur-Eigenschaft:

Vollständiger Binärbaum, mit Ausnahme der untersten Stufe; dort ist er von links nach rechts aufgefüllt.

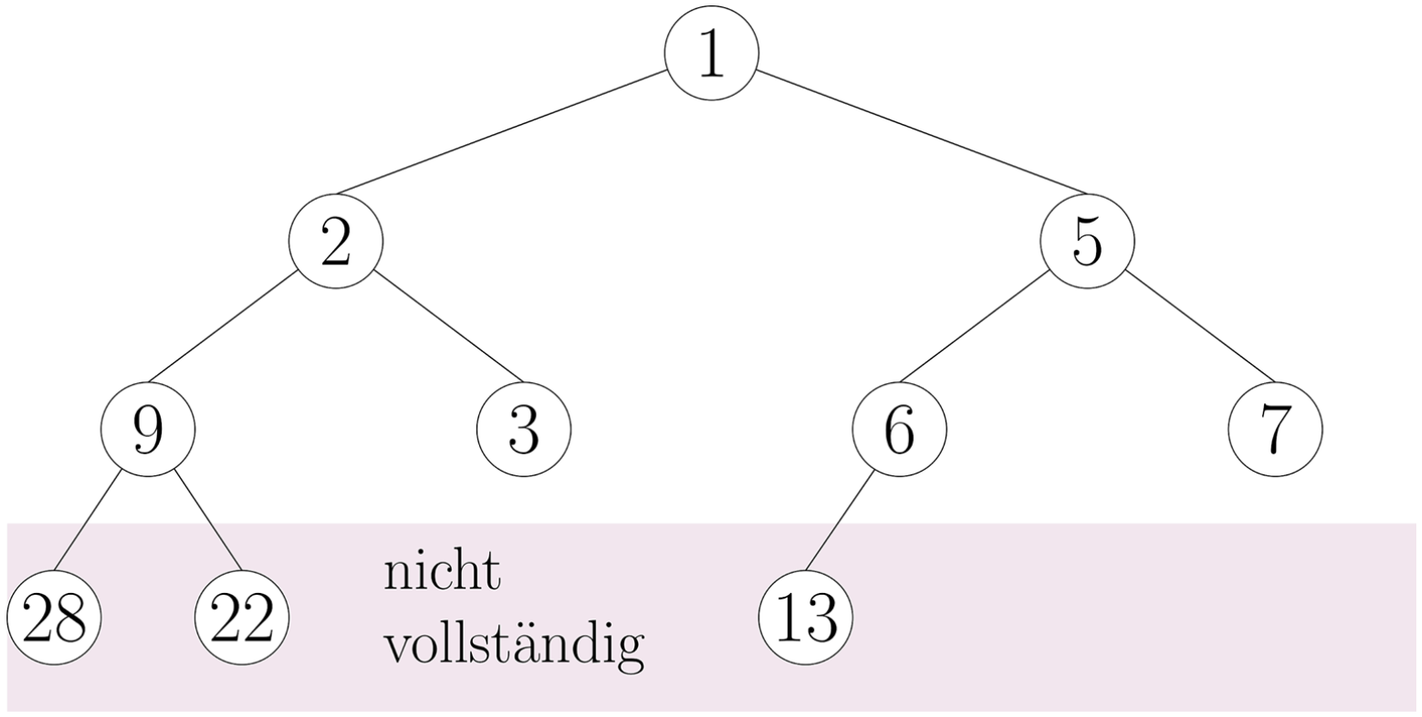
Ordnungs-Eigenschaft:

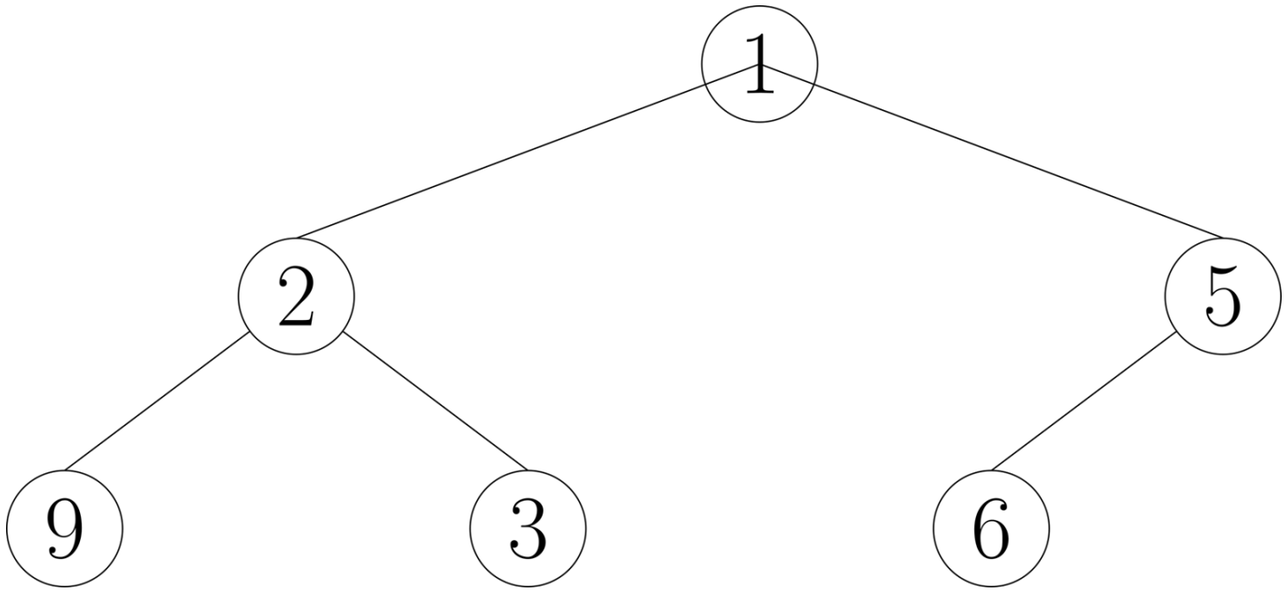
Der Schlüssel jedes Knotens ist kleiner gleich der Schlüssel seiner beider Kinder (falls vorhanden).

Heap oder kein Heap?

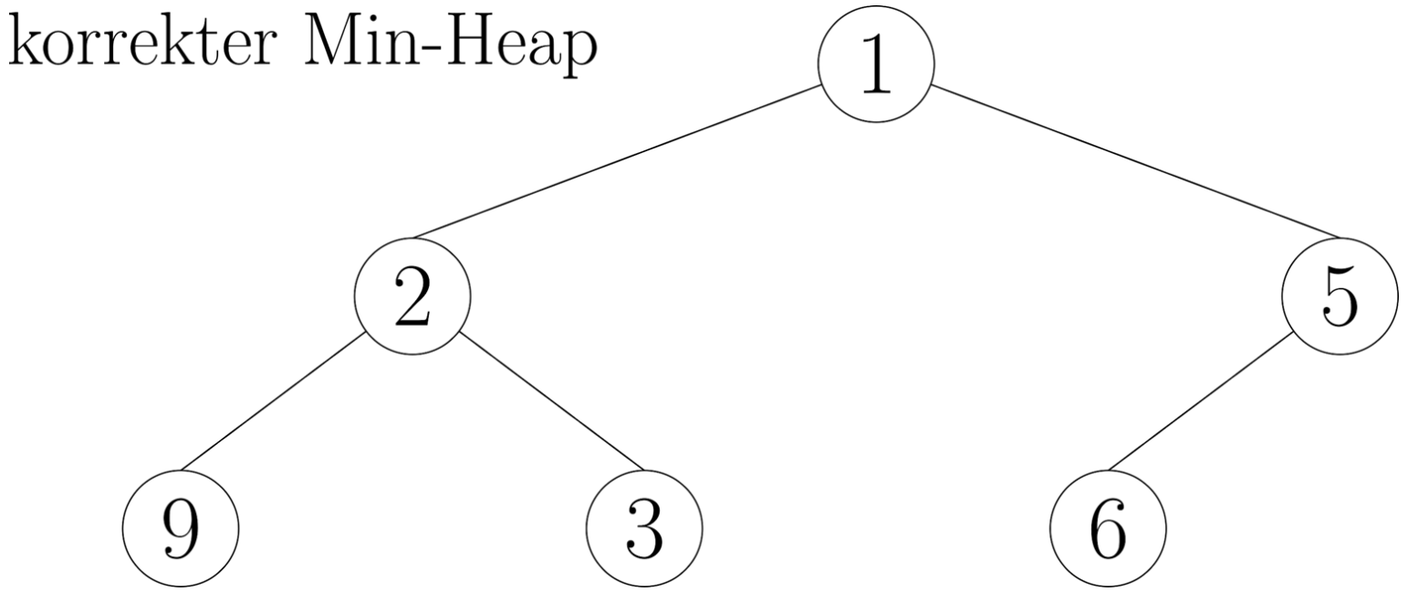


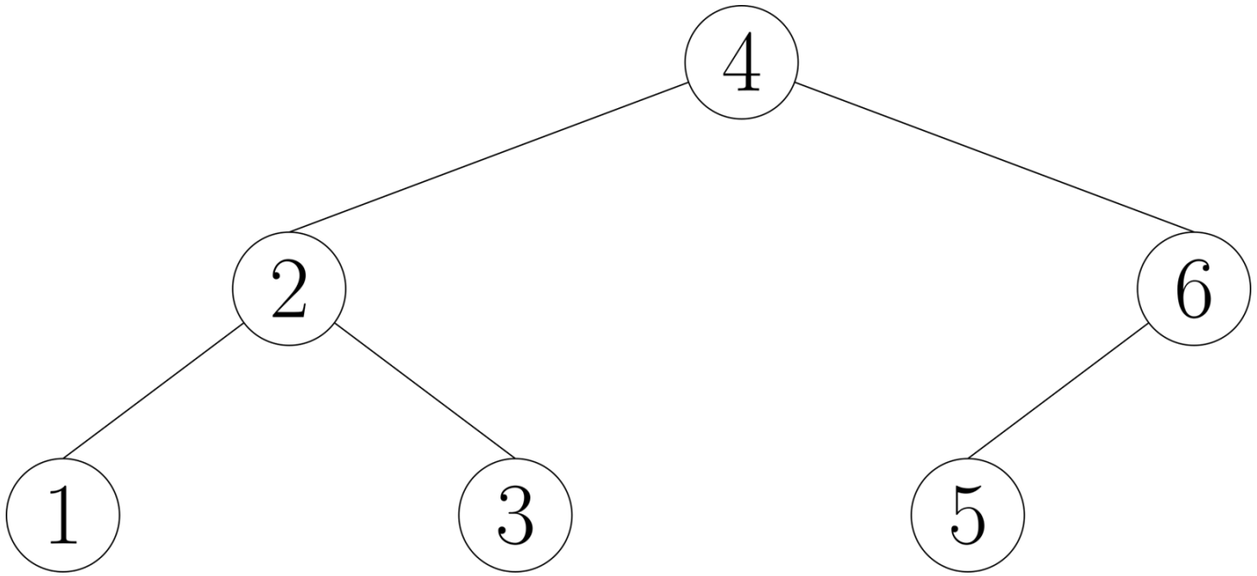
Heap oder kein Heap?


1
2
5
9
3
6
7
28
22
13
nicht
vollständig

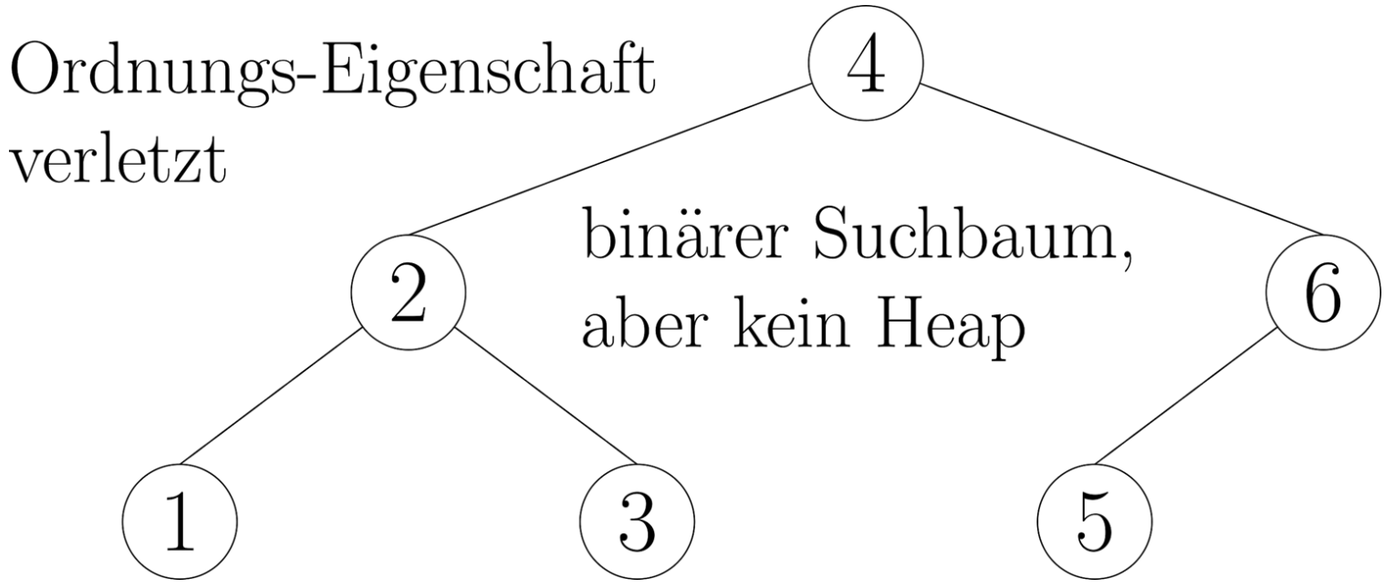


Heap oder kein Heap?


korrekter Min-Heap
1
2
5
9
3
6



Heap oder kein Heap?


Ordnungs-Eigenschaft
verletzt
4
2
6
1
3
5
binärer Suchbaum,
aber kein Heap

Fachhochschule
Nordwestschweiz

Lösen Sie die Aufgabe 4

Operationen auf einem Min-Heap

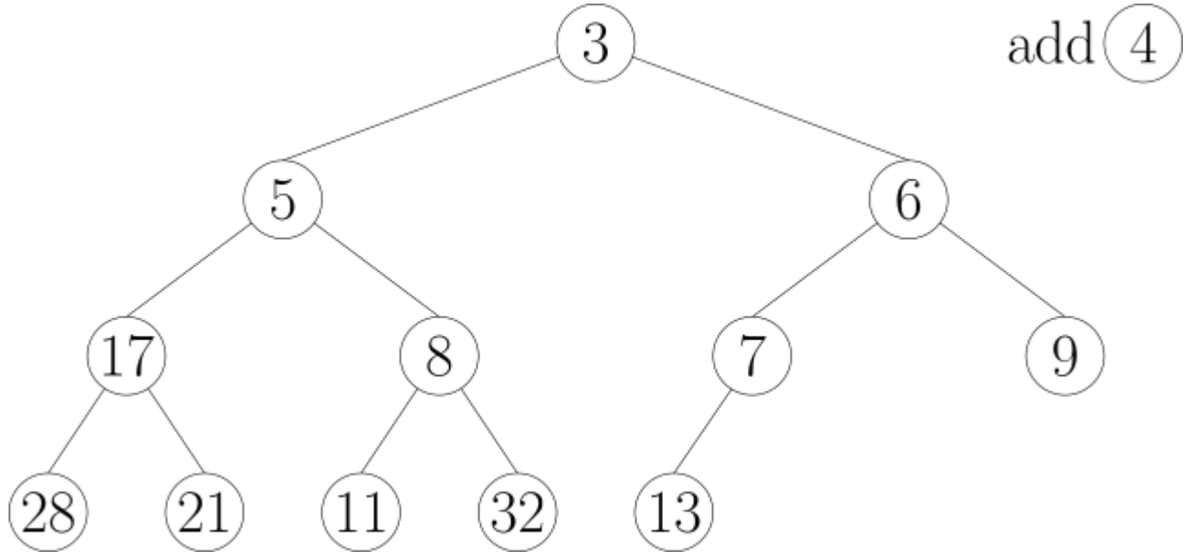
min()	gibt das kleinste Element ⟶ steht in der Wurzel
removeMin()	gibt und entfernt das kleinste Element
add(elem)	fügt ein Element elem hinzu
size()	gibt die Anzahl Elemente in der Priority Queue ⟶ wie in jeder Collection

Operationen auf einem Min-Heap

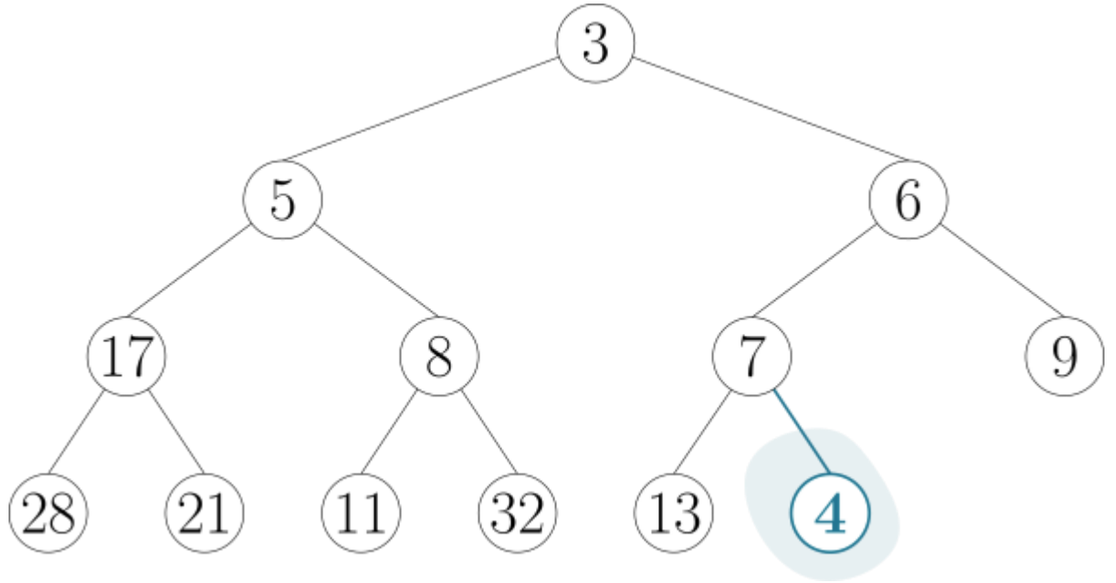
min()	gibt das kleinste Element ⟶ steht in der Wurzel
removeMin()	gibt und entfernt das kleinste Element
add(elem)	fügt ein Element elem hinzu
size()	gibt die Anzahl Elemente in der Priority Queue ⟶ wie in jeder Collection

Aufwand: jeweils O(1)

add(elem)

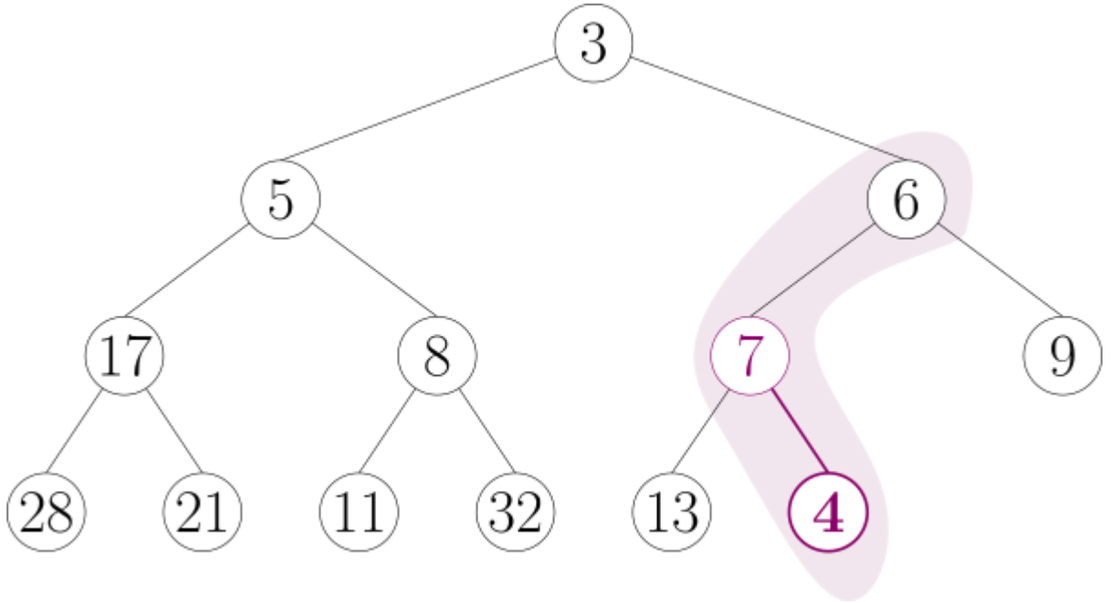


add(elem)



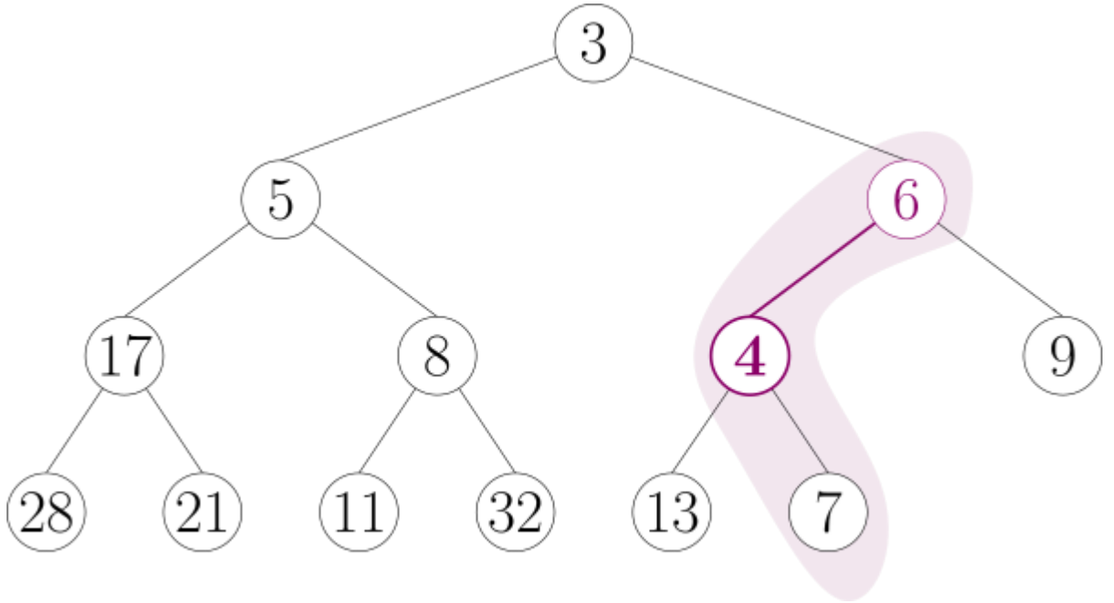
1. **Struktur-Eigenschaft:** Das neue Element kommt an die erste freie Stelle auf der untersten Stufe (oder links auf neuer Stufe).

add(elem)



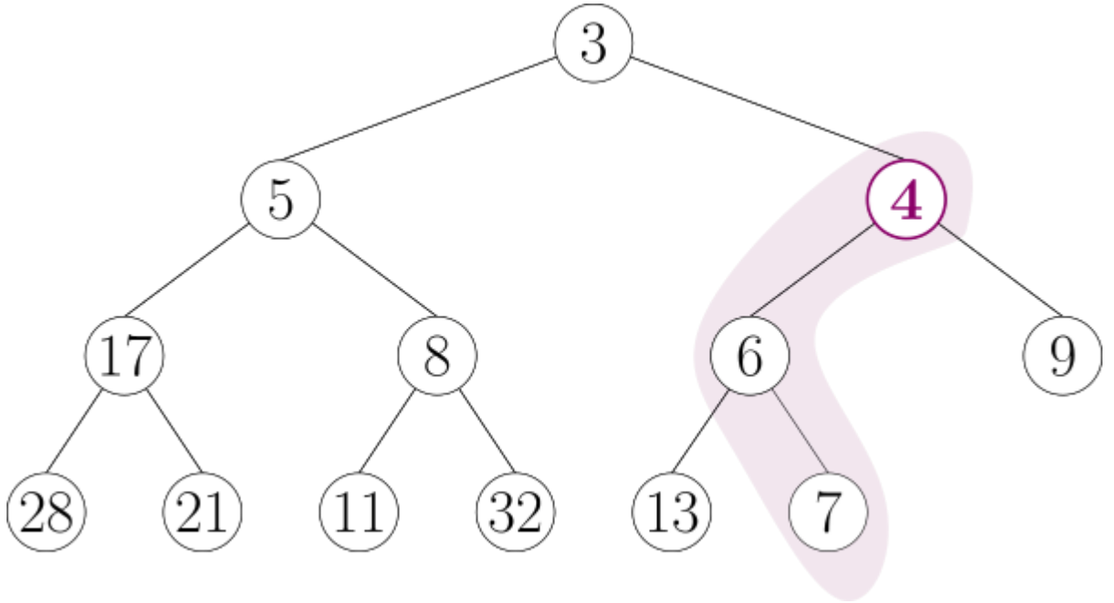
1. **Struktur-Eigenschaft:** Das neue Element kommt an die erste freie Stelle auf der untersten Stufe (oder links auf neuer Stufe).
2. **Ordnungs-Eigenschaft:** Das neue Element wandert hinauf;

add(elem)



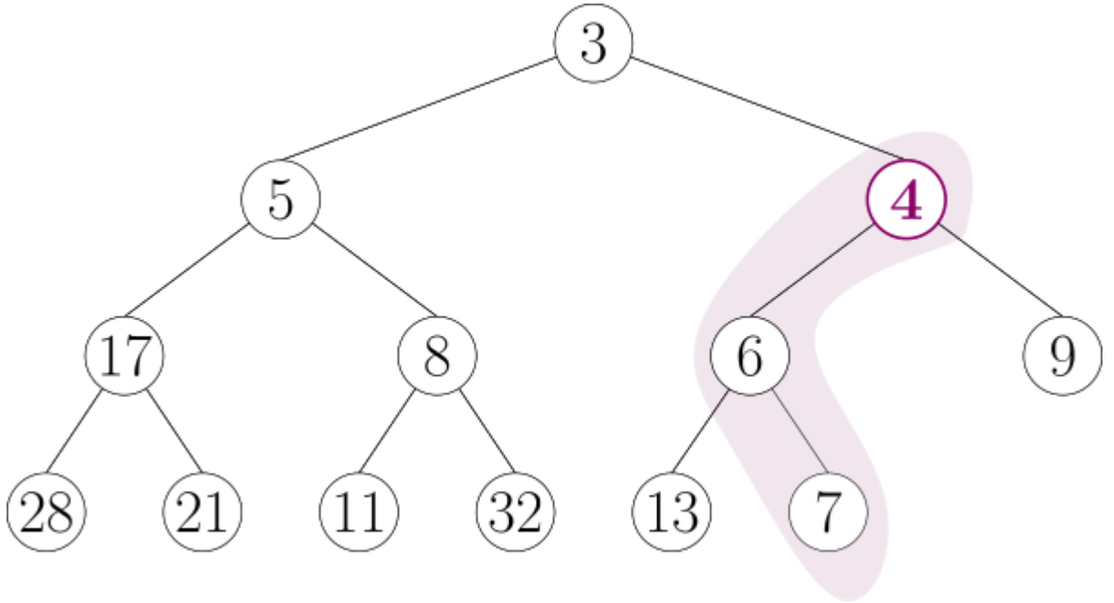
1. **Struktur-Eigenschaft:** Das neue Element kommt an die erste freie Stelle auf der untersten Stufe (oder links auf neuer Stufe).
2. **Ordnungs-Eigenschaft:** Das neue Element wandert hinauf;

add(elem)



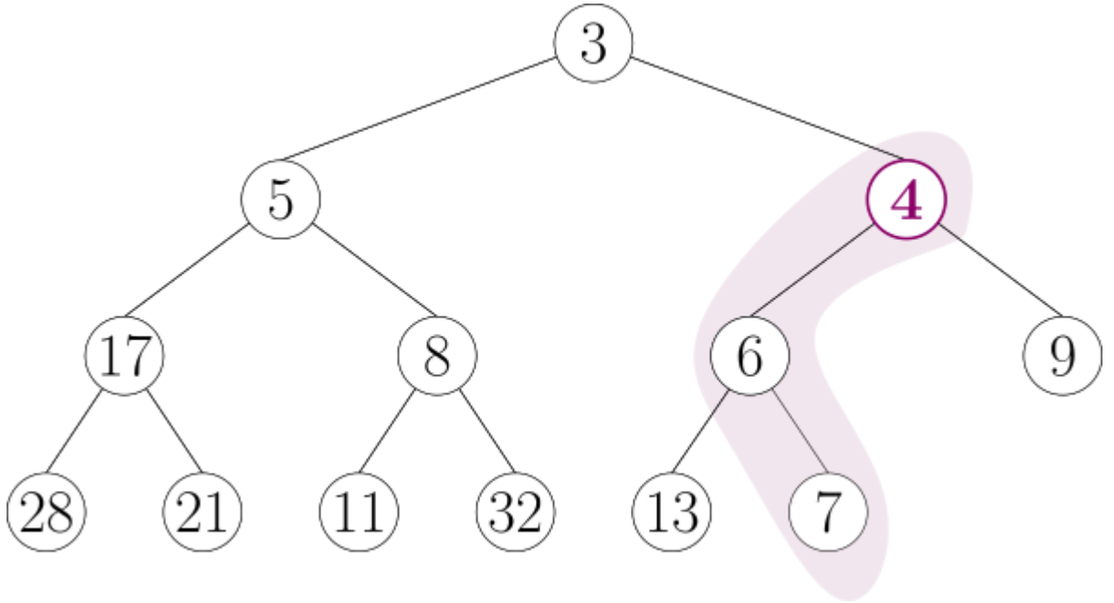
1. **Struktur-Eigenschaft:** Das neue Element kommt an die erste freie Stelle auf der untersten Stufe (oder links auf neuer Stufe).
2. **Ordnungs-Eigenschaft:** Das neue Element wandert hinauf;

add(elem)



1. **Struktur-Eigenschaft:** Das neue Element kommt an die erste freie Stelle auf der untersten Stufe (oder links auf neuer Stufe).
2. **Ordnungs-Eigenschaft:** Das neue Element wandert hinauf;
   es wird solange mit seinem Vater vertauscht, bis es grösser als der Vater ist oder in der Wurzel steht.

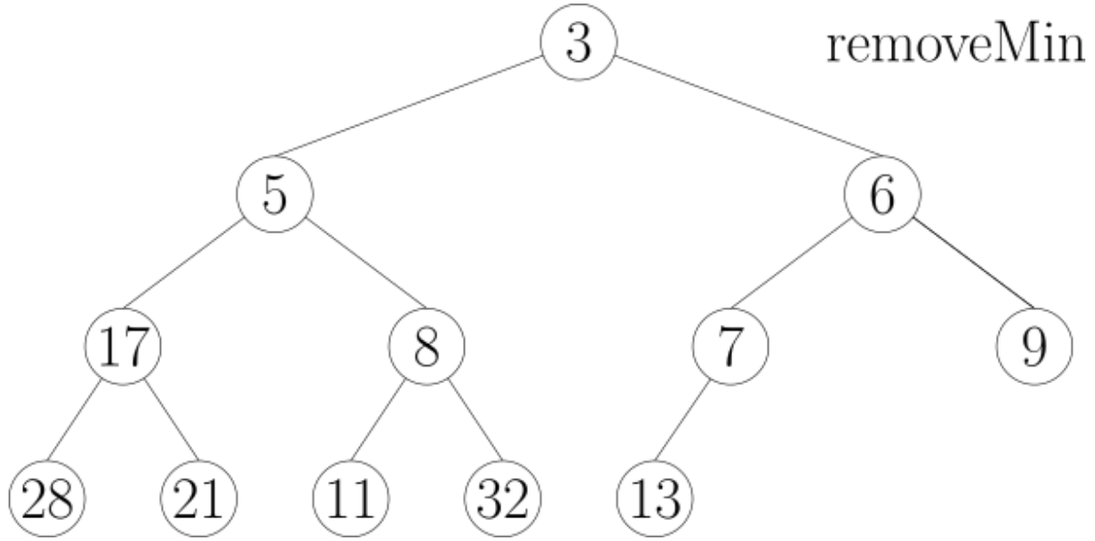
add(elem)



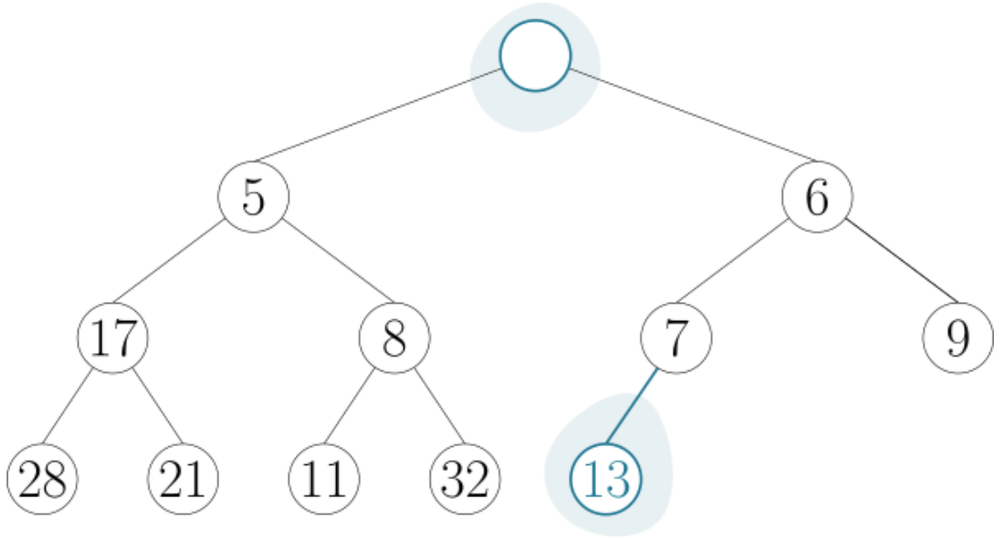
1. **Struktur-Eigenschaft:** Das neue Element kommt an die erste freie Stelle auf der untersten Stufe (oder links auf neuer Stufe).
2. **Ordnungs-Eigenschaft:** Das neue Element wandert hinauf;
   es wird solange mit seinem Vater vertauscht, bis es grösser als der Vater ist oder in der Wurzel steht.

Aufwand: O(1) + O(log n) = O(log n)

removeMin()

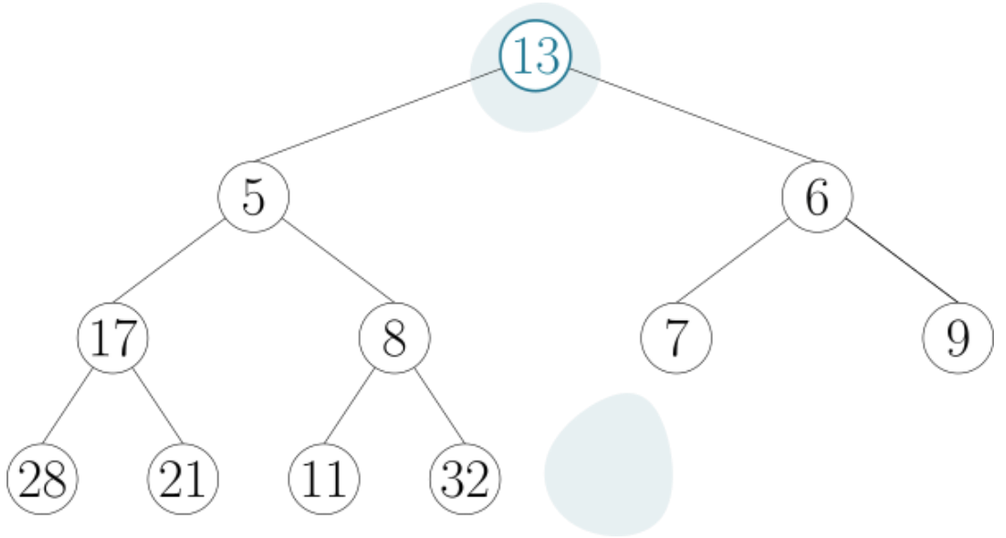


removeMin()



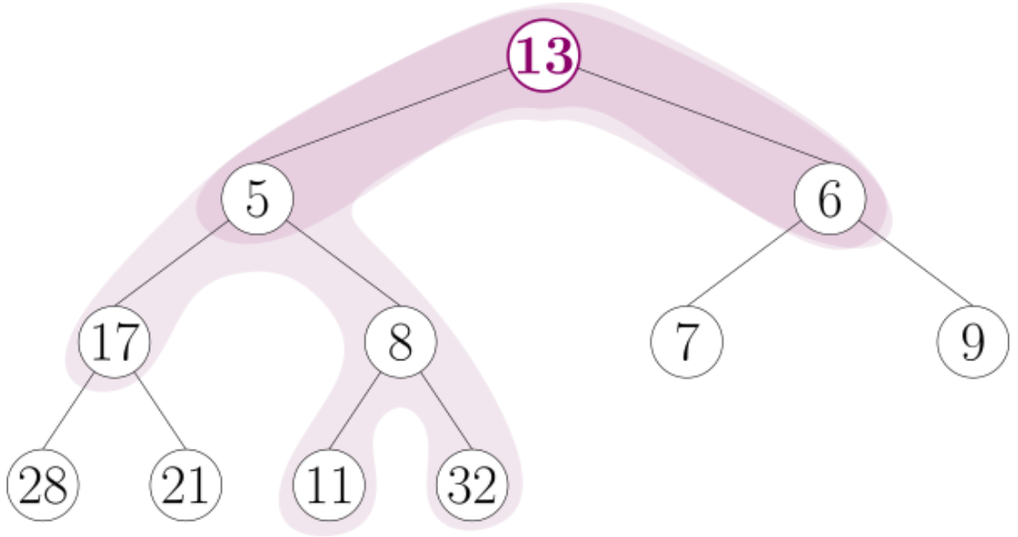
1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.

removeMin()



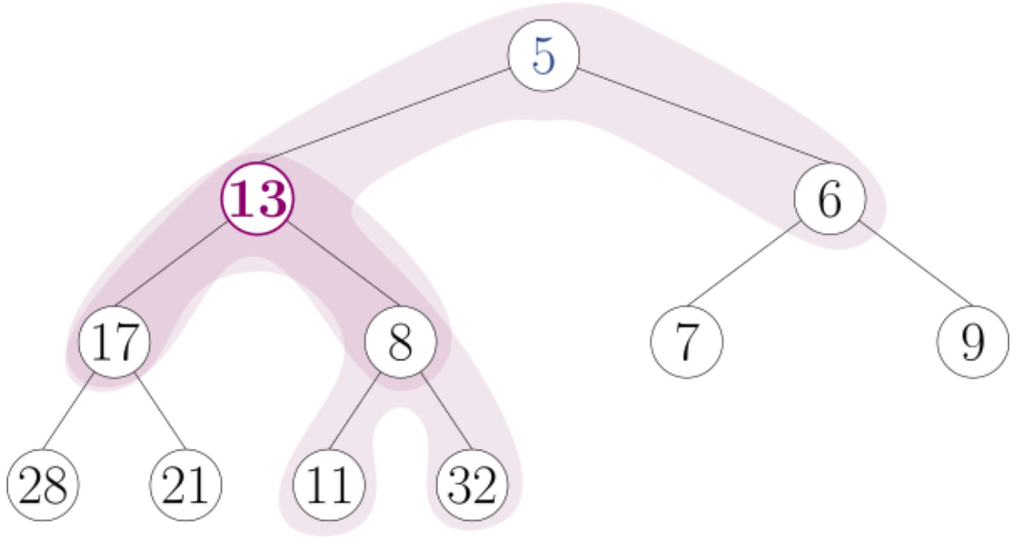
1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.

removeMin()



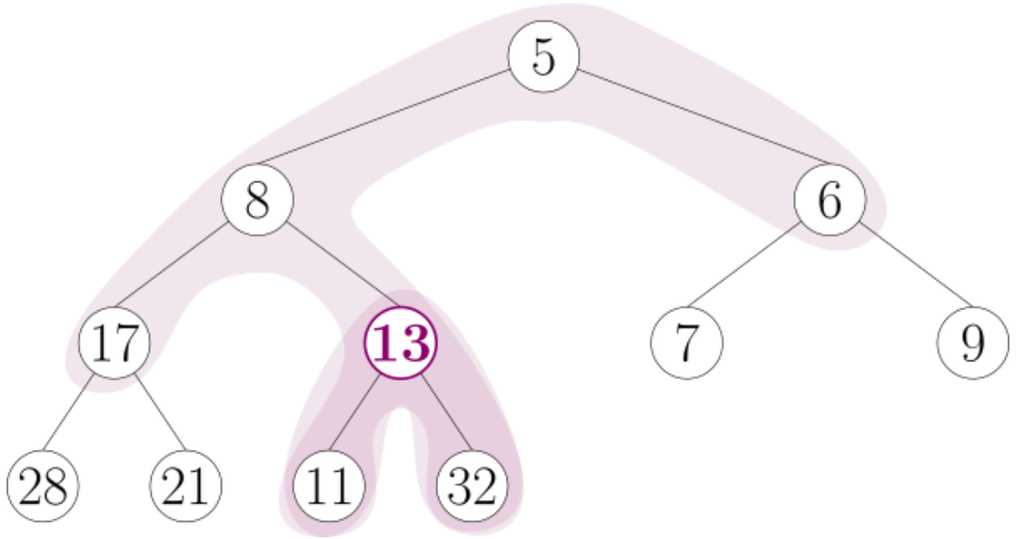
1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert;

removeMin()



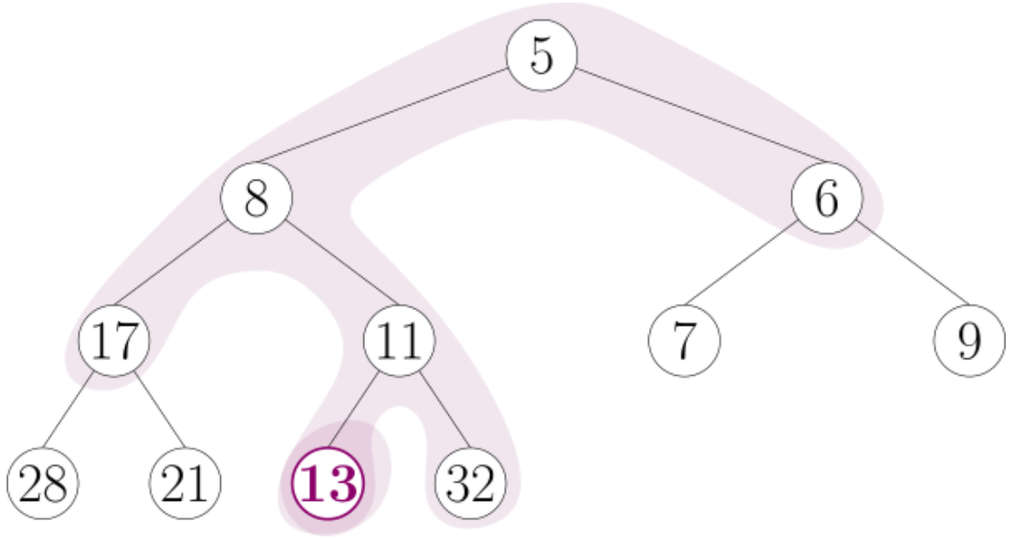
1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert;

removeMin()



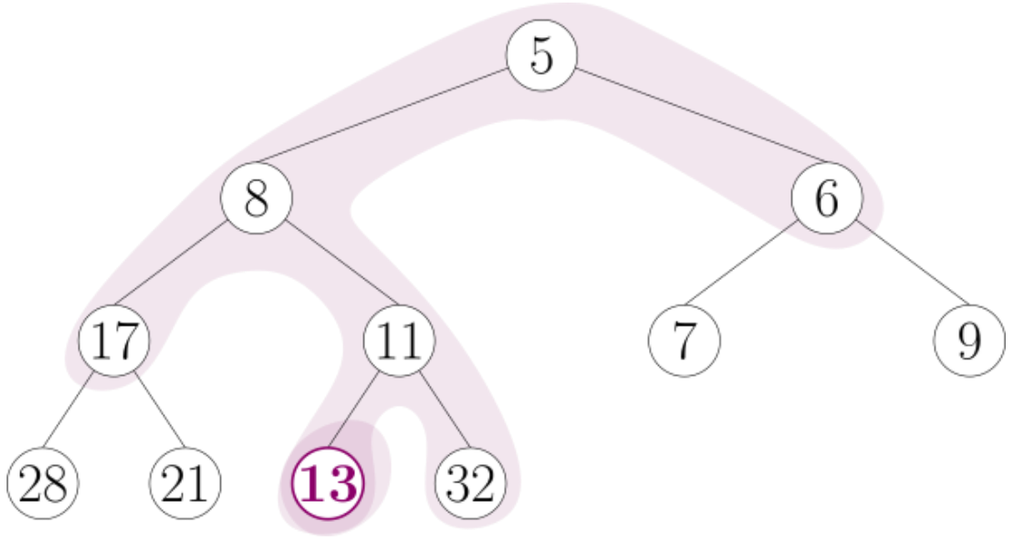
1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert;

removeMin()



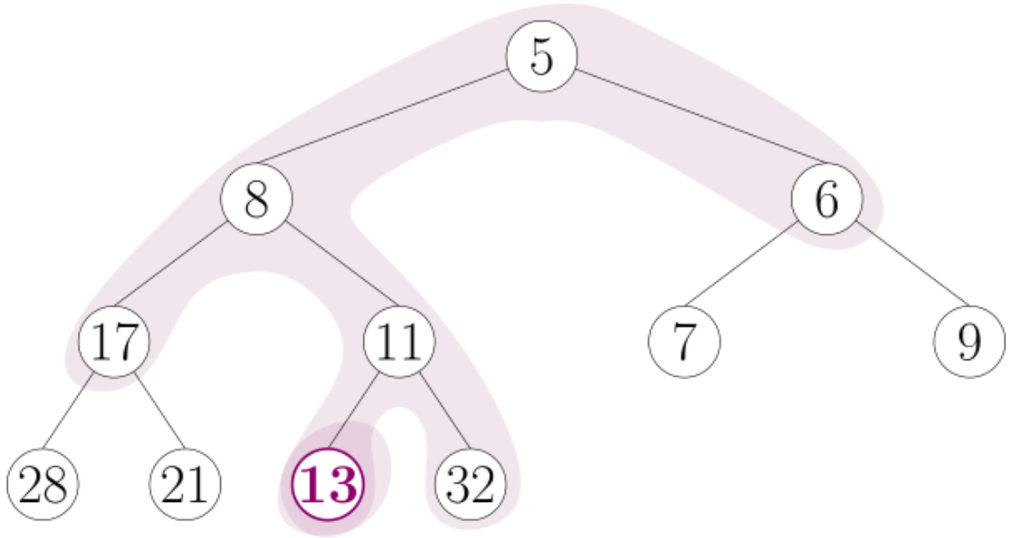
1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert;

removeMin()



1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert; es wird solange mit seinem kleineren Kind getauscht, bis es kleiner ist als alle seine Kinder oder in einem Blatt steht.

removeMin()



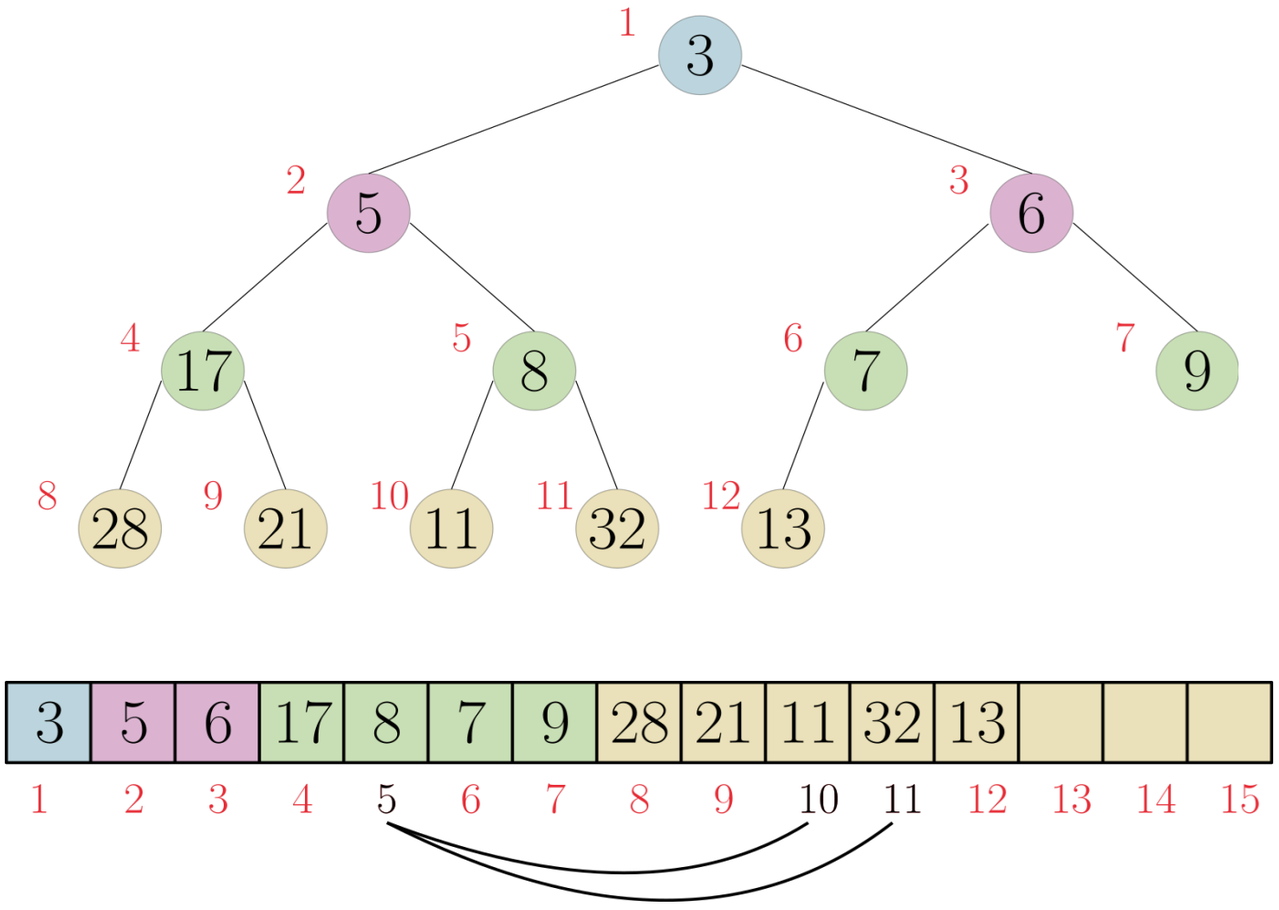
1. **Struktur-Eigenschaft:** Das Wurzelelement wird durch das letzte Element auf der untersten Stufe ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert; es wird solange mit seinem kleineren Kind getauscht, bis es kleiner ist als alle seine Kinder oder in einem Blatt steht.

Aufwand: O(1) + O(log n) = O(log n)

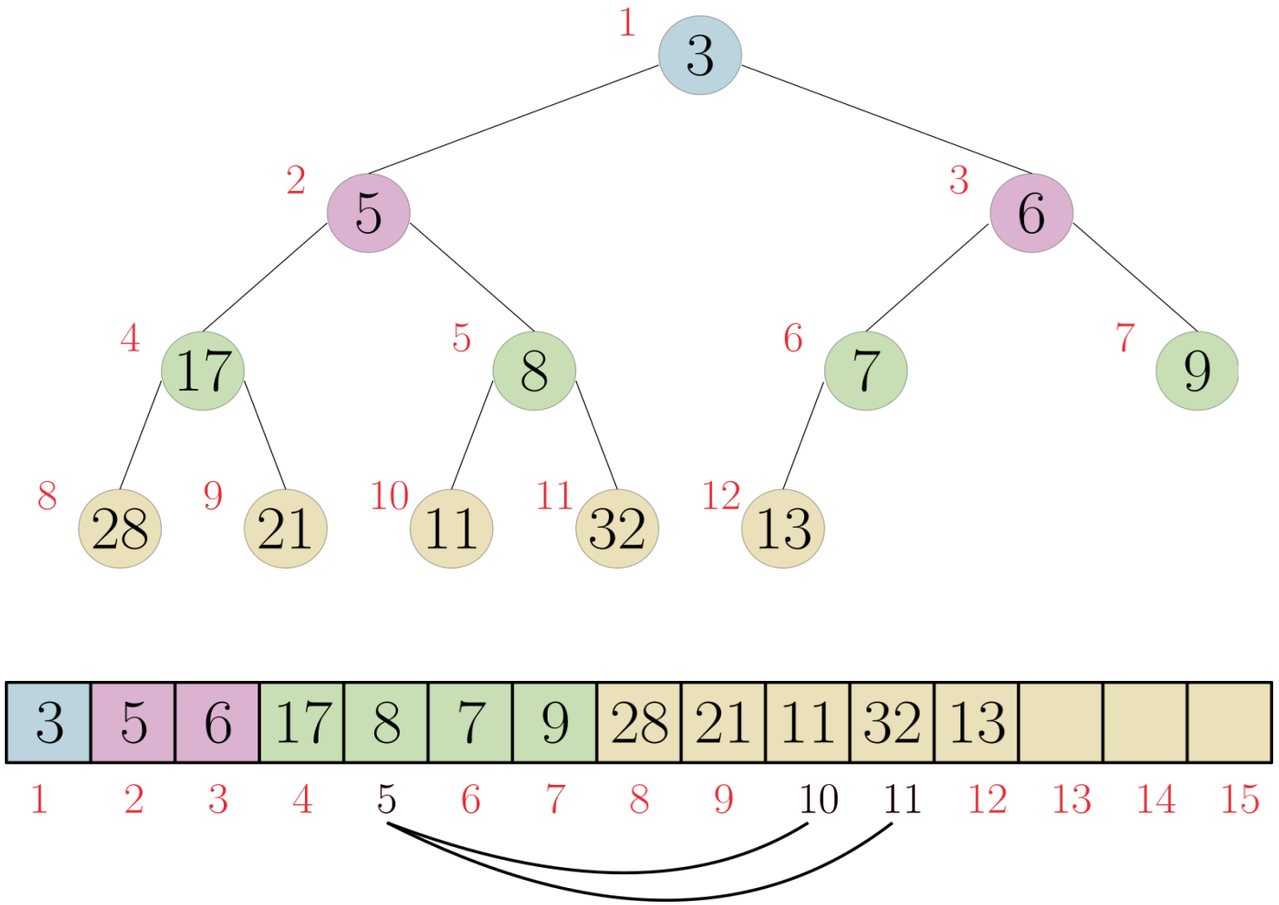
Fachhochschule
Nordwestschweiz

Lösen Sie die Aufgabe 5

Heap-Darstellung im Array



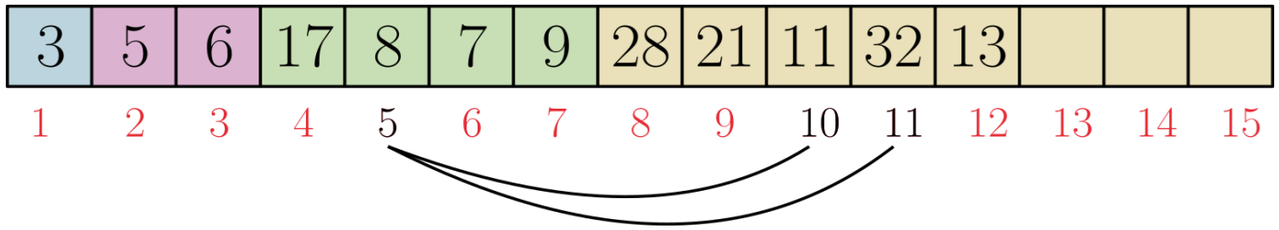
Heap-Darstellung im Array



Struktur-Eigenschaft:

- Der Array hat keine Lücken.
- Die Kinder eines Elements an Position i stehen im Array an den Positionen 2i und 2i + 1 (1-basierte Indizes).

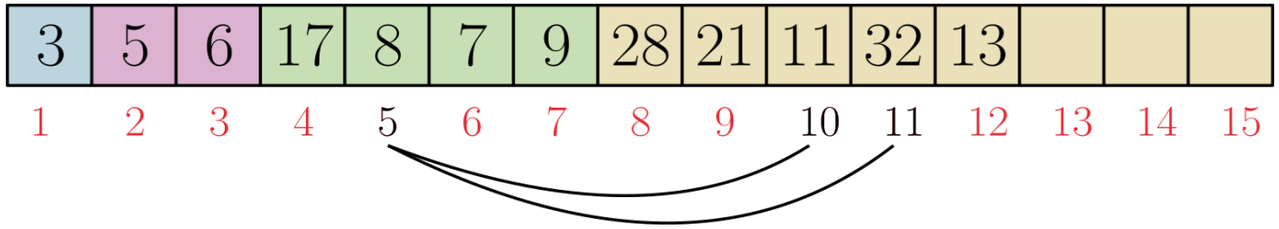
Heap-Darstellung im Array



Aufgabe: Füllen Sie die folgende Tabelle aus für 1-basierte und 0-basierte Indizes.

	Wurzel bei Index 1	Wurzel bei Index 0
Vater-Knoten von *i*		
linker Nachfolger von *i*		
rechter Nachfolger von *i*		
Indizes aller Blätter	bis	bis

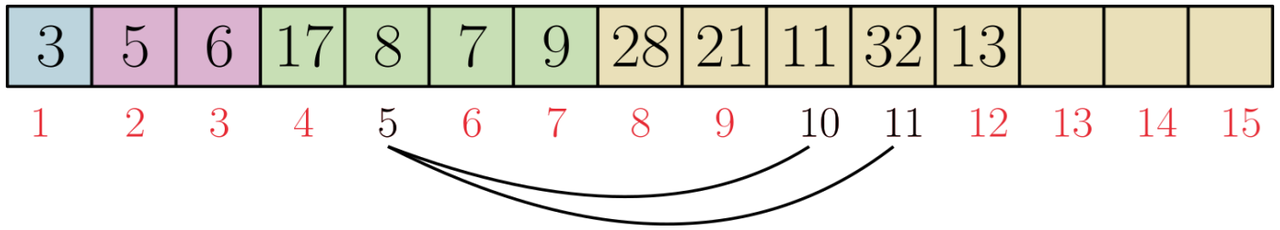
Heap-Darstellung im Array



Aufgabe: Füllen Sie die folgende Tabelle aus für 1-basierte und 0-basierte Indizes.

	Wurzel bei Index 1	Wurzel bei Index 0
Vater-Knoten von *i*	⌊i / 2⌋	
linker Nachfolger von *i*	2i	
rechter Nachfolger von *i*	2i + 1	
Indizes aller Blätter	⌊size / 2⌋ + 1 bis size	

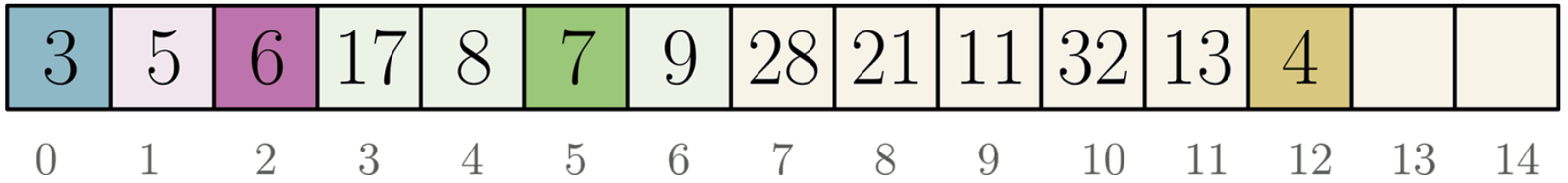
Heap-Darstellung im Array



Aufgabe: Füllen Sie die folgende Tabelle aus für 1-basierte und 0-basierte Indizes.

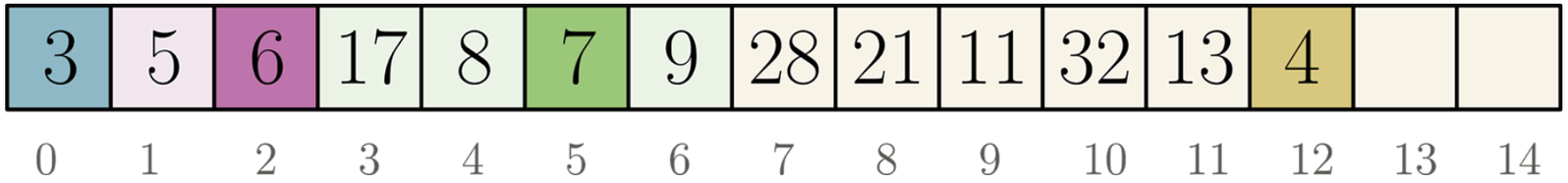
	Wurzel bei Index 1	Wurzel bei Index 0
Vater-Knoten von *i*	⌊i / 2⌋	⌊(i-1) / 2⌋
linker Nachfolger von *i*	2i	2i + 1
rechter Nachfolger von *i*	2i + 1	2i + 2
Indizes aller Blätter	⌊size / 2⌋ + 1 bis size	⌊size / 2⌋ bis size - 1

add(element)



1. **Struktur-Eigenschaft**: Das neue Element kommt an die erste freie Stelle im Array.
2. **Ordnungs-Eigenschaft**: Das neue Element wandert hinauf (eigene Methode siftUp).

add(element)

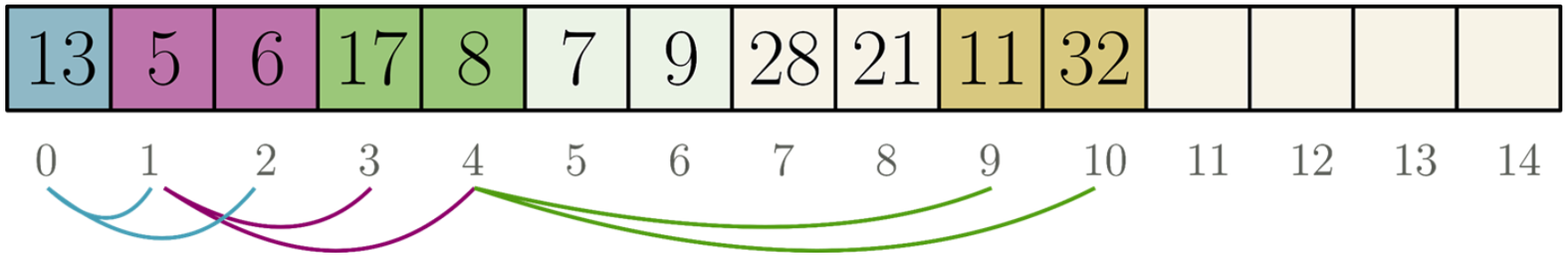


1. **Struktur-Eigenschaft**: Das neue Element kommt an die erste freie Stelle im Array.
2. **Ordnungs-Eigenschaft**: Das neue Element wandert hinauf (eigene Methode siftUp).

Beispielcode in Java (0-basierte Indizes):

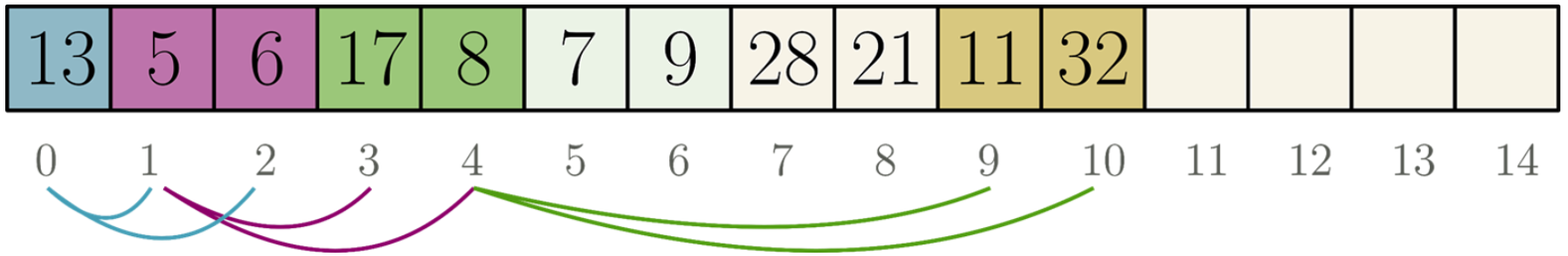
```
public void add(K element ) {
    heap[size] = element;
    heap.siftUp(size);
    size++;
}
```

removeMin()



1. **Struktur-Eigenschaft:** Das erste (Wurzel) wird durch das letzte Element im Array ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert (eigene Methode siftDown.)

removeMin()



1. **Struktur-Eigenschaft:** Das erste (Wurzel) wird durch das letzte Element im Array ersetzt.
2. **Ordnungs-Eigenschaft:** Das neue Wurzelelement versickert (eigene Methode siftDown.)

Beispielcode in Java (0-basiert):

```
public K removeMin() {
    K res = heap.min();
    heap[0] = heap[size-1];
    heap[size-1] = null;
    size--;
    heap.siftDown(0);
    return res;
}
```

Hausaufgaben

Arbeitsblatt

- Aufgaben 1 & 2 & 6

Programmieren

- Programmieraufgabe 1 (Heap)