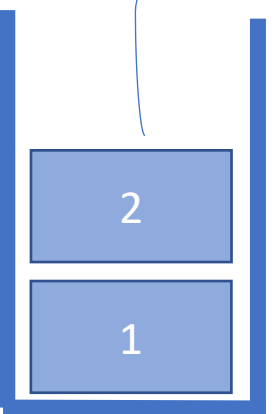
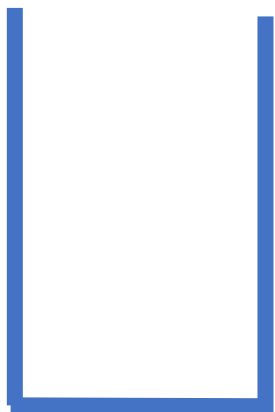


A1 Stack

Laufzeit: $O(1)$

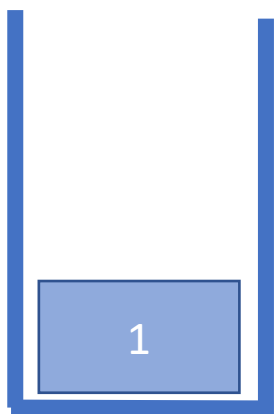
- 1 Wenn leer, dann throw new StackEmptyException

public T **pop()**



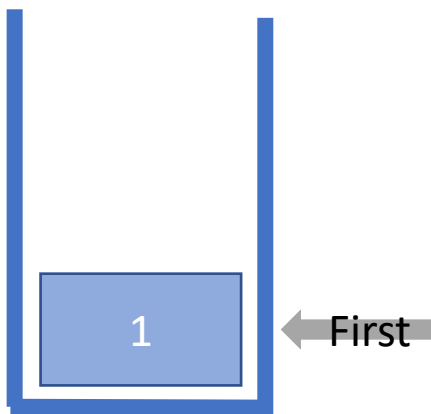
2
← First

Oberstes Element (first) wird entfernt

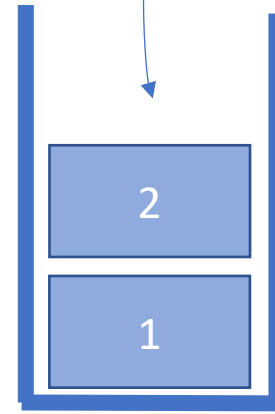


3
← First

Das oberste Element (first) wird das darunter stehende Element



← First

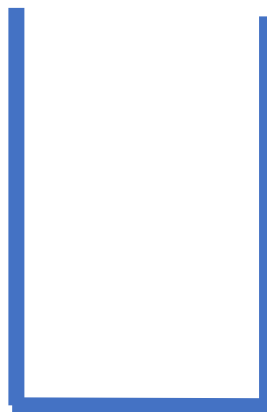


← First

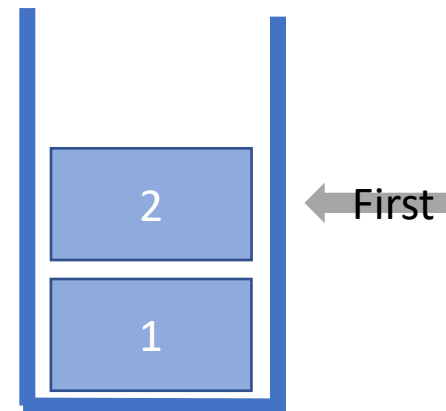
Das oberste Element (nächste Element) erhält First

public T **push()**

Wenn Stack leer (first == null), ist die Anzahl null



public int **getCount()**



← First

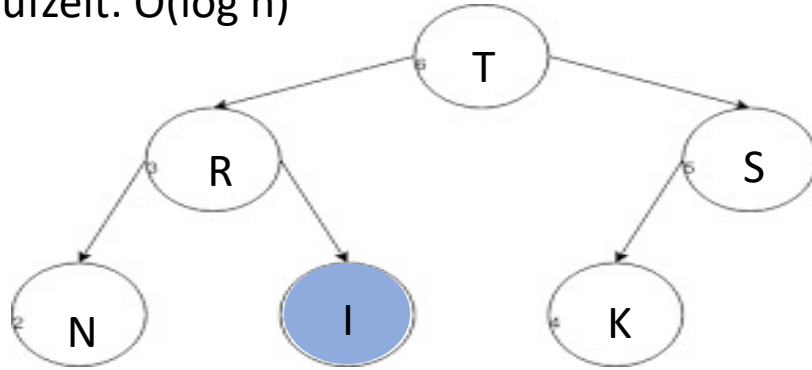
Solang es ein nächstes Element (hinuntergezählt) gibt, wird der counter i erhöht

A2 Heap

public void **insert**(Task t)

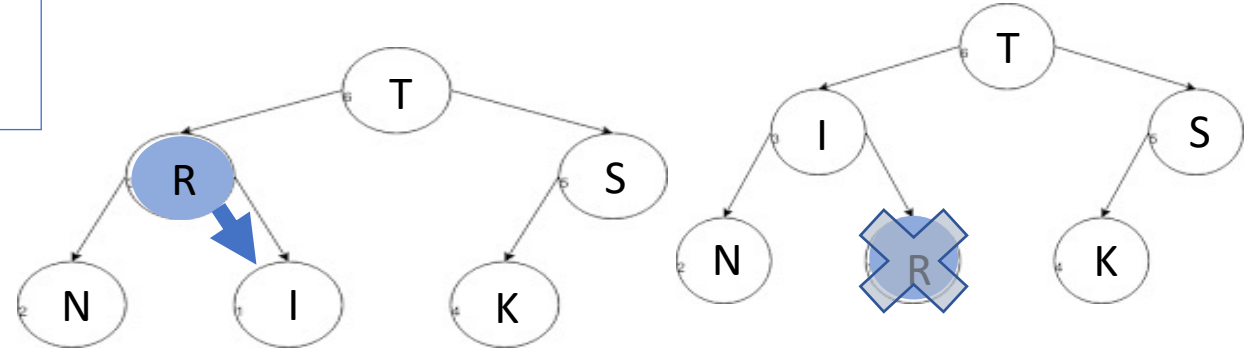
- I wird an unterster Ebene hinzugefügt
- Stelle task.size()-1
- An die letzte Position schwimmen lassen

Laufzeit: $O(\log n)$



public Task **remove**()

- Letzte Position bestimmen $\rightarrow \text{lastPos} = \text{tasks.size()-1}$;
- Wenn $\text{lastPos} > 0 \rightarrow$ tausch von (1, lastPos)
- Nun $\text{tasks.remove}(\text{lastPos})$ aufrufen, um das Element zu entfernen

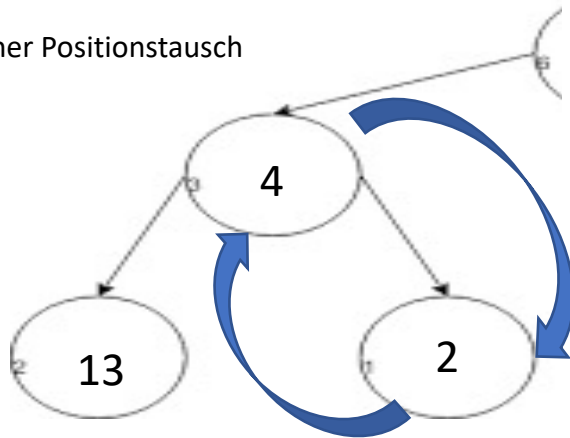


private void **swim**(int pos)

Laufzeit: $O(\log n)$

- Festlegung der Priorität der Position (swimmerPrio)
- Wenn die Priorität des Knoten an der $\text{parentPos} >$ als jene der swimmerPrio ist \rightarrow dann werden diese Position getauscht und aus dem Parentknoten wird ein Child

$4 > 2 \rightarrow$ daher Positionstausch

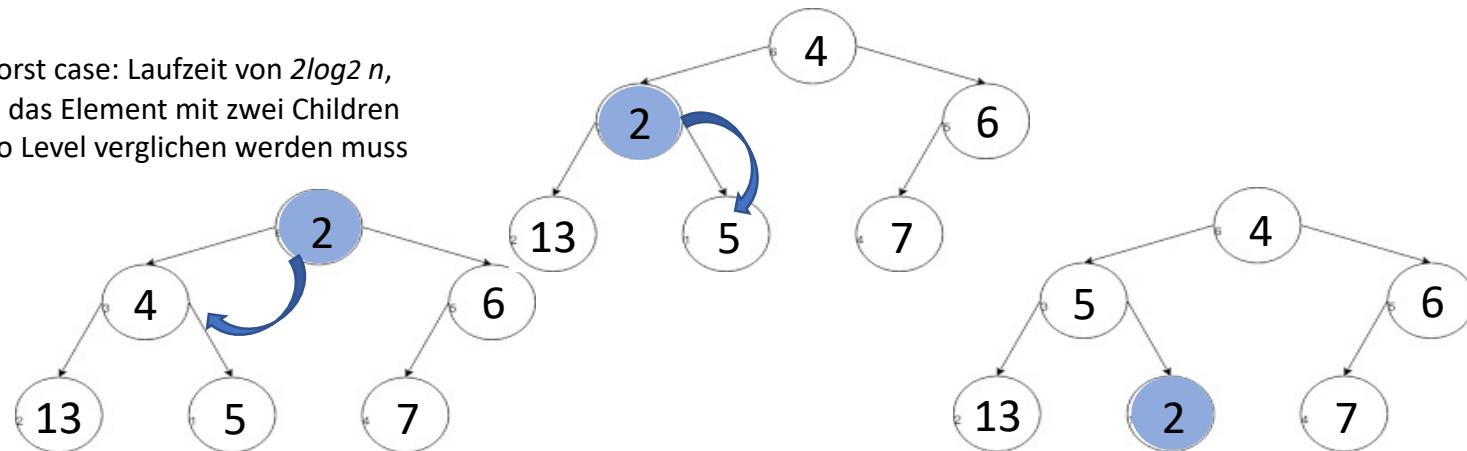


private void **sink**(int pos)

Laufzeit: $O(\log n)$

- Derzeitige $\text{sinkerPos} = 1$ und die Priorität ist in sinkerPrio gespeichert
- Solange das doppelte von $\text{sinkerPos} < \text{task.size}$ ist..
 - Entspricht die $\text{childPos} = \text{minChild}(\text{sinkerPos})$
 - Wenn die Priorität von der childPos kleiner als die $\text{sinkerPrio} \rightarrow$ sprich: das Child hat eine kleinere Priorität als das zu sinkende Element
 - Dann werden childPos und sinkerPos miteinander getauscht

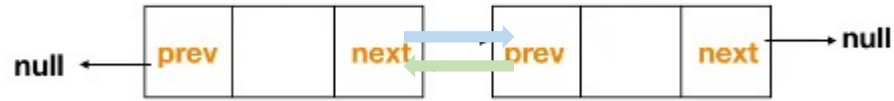
Worst case: Laufzeit von $2\log_2 n$, da das Element mit zwei Children pro Level verglichen werden muss



A3 DoubleLinked List

public void **add**(T a)

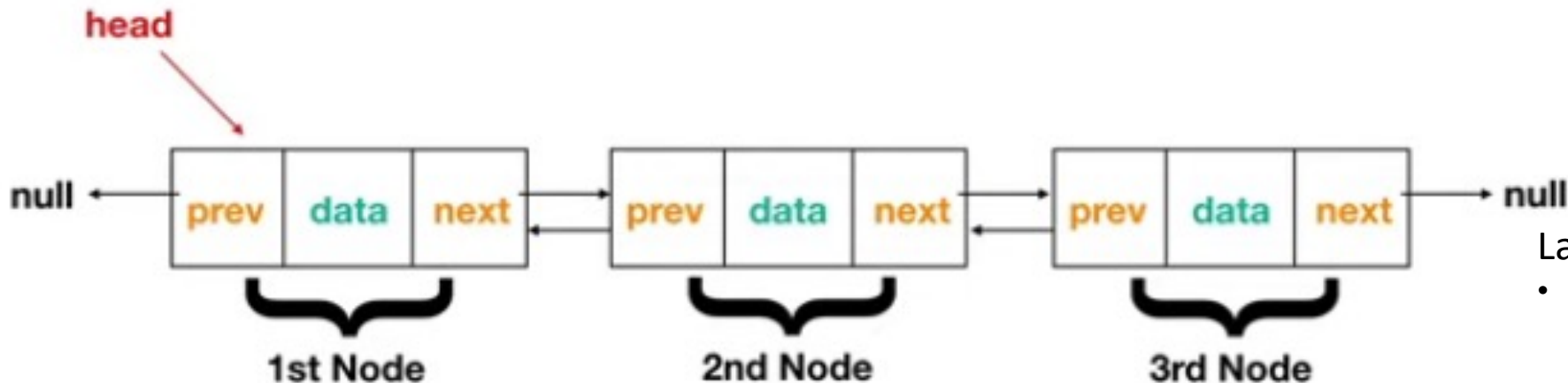
- Um ein neues Node hinzufügen, wird `Node.setPrevious(last)` mit dem vorhergehenden Node verbunden und durch `previous.setNext(Node)` mit dem folgenden Node verbunden



- Wenn Node das erste der Liste ist \rightarrow (first == null)
 - Dann ist first = tmp und last = tmp

remove(int pos)

- wir iterieren bis zur Node die gelöscht werden soll
- wir setzen als previos vom next, das previos von pos
- wir setzen als next von previos, das next von pos



Laufzeiten:

- Abhängig von Position
 - Am Ende oder am Beginn meist $O(1)$
 - An bestimmter Position $O(k)$