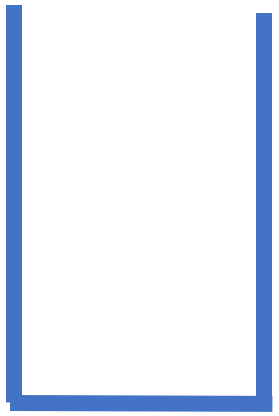
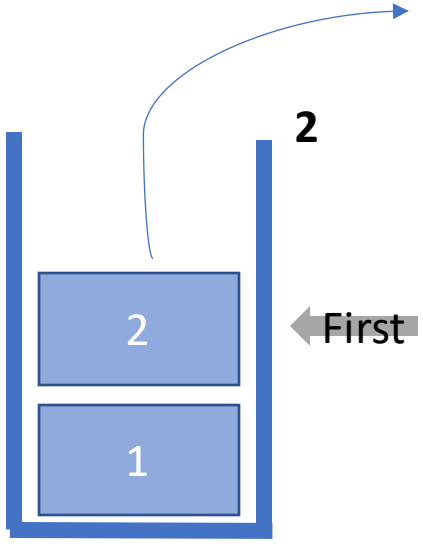
A1 Stack

Laufzeit: O(1)



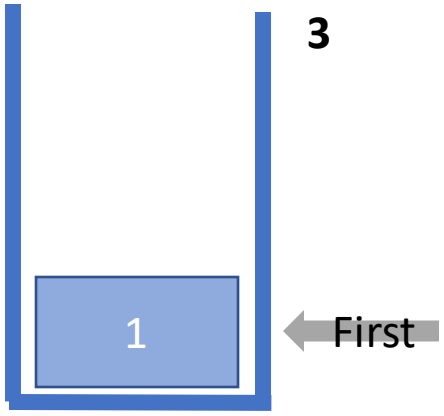
1 Wenn leer, dann throw new StackEmptyException

public T **pop**()



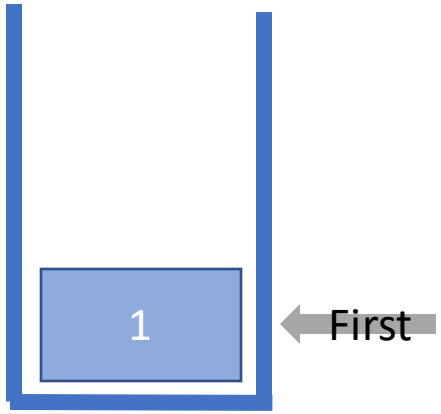
2

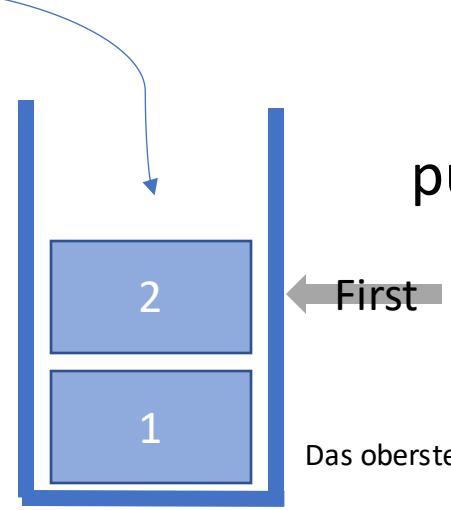
Oberstes Element (first) wird entfernt



3

Das oberste Element (first) wird das darunter stehende Element

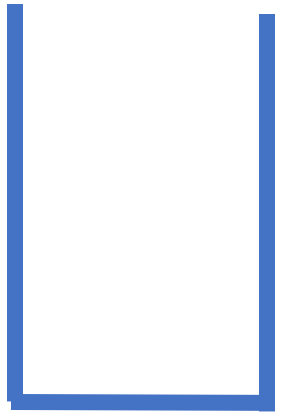




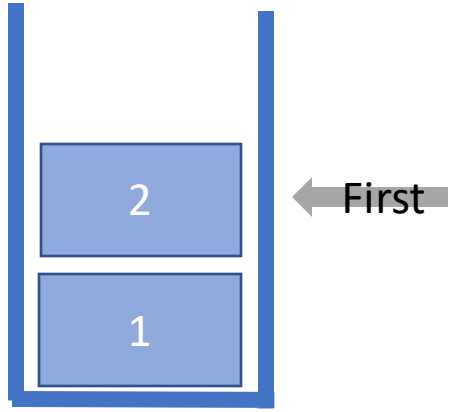
public T **push**()

Das oberste Element (nächste Element) erhält First

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



public int **getCount**()



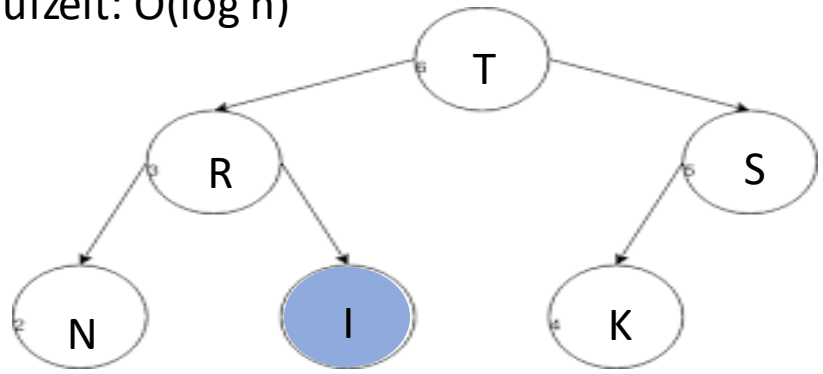
Solang es ein nächstes Element (hinuntergezählt) gibt, wird der counter i erhöht

A2 Heap

public void **insert**(Task t)

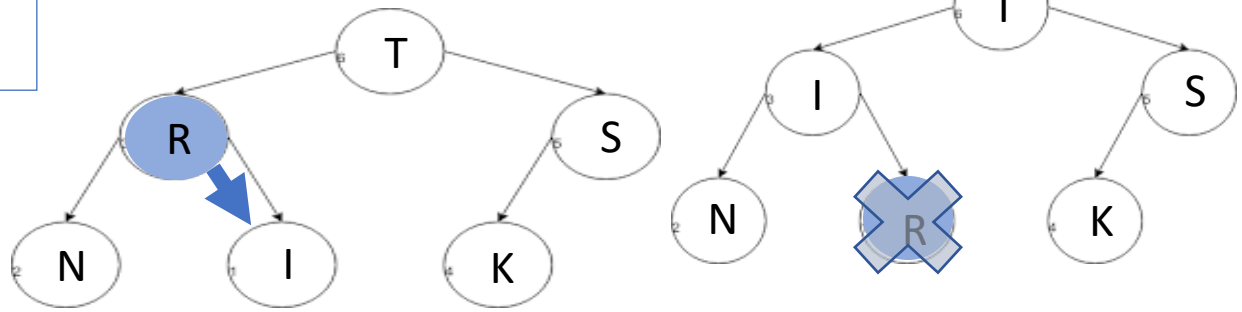
- I wird an unterster Ebene hinzugefügt
- Stelle task.size()-1
- An die letzte Position schwimmen lassen

Laufzeit: O(log n)



public Task **remove**()

- Letzte Position bestimmen → lastPos = tasks.size()-1;
- Wenn lastPos > 0 → tausch von (1, lastPos)
- Nun tasks.remove(lastPos) aufrufen, um das Element zu entfernen

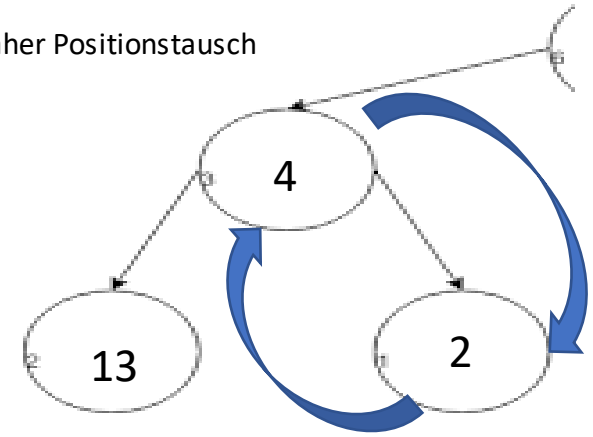


private void **swim**(int pos)

Laufzeit: O(log n)

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

4 > 2→ daher Positionstausch

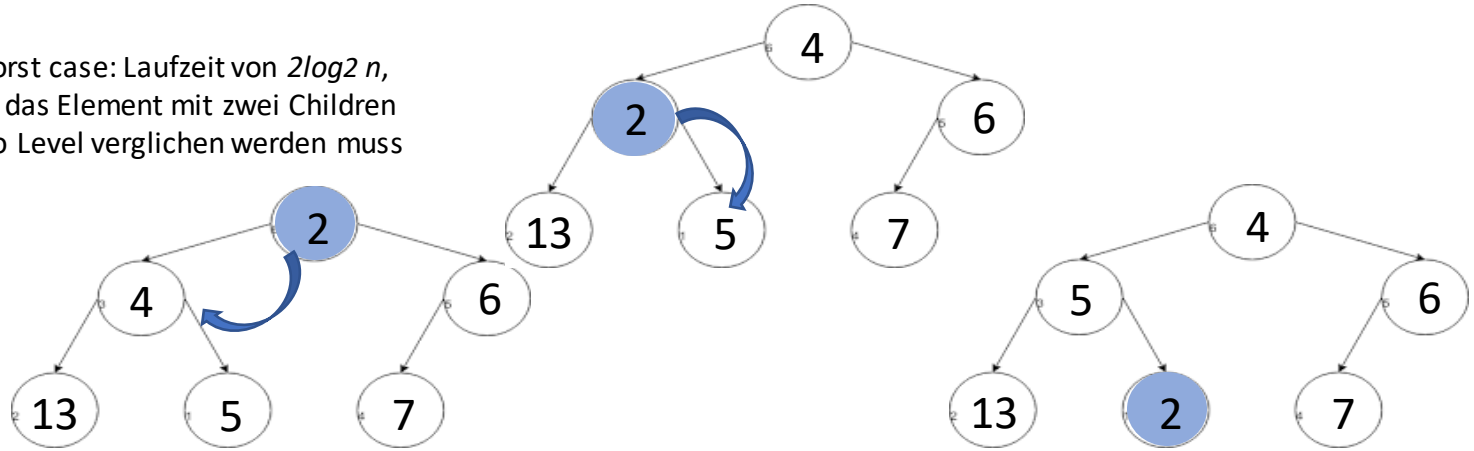


private void **sink**(int pos)

Laufzeit: O(log n)

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

Worst case: Laufzeit von *2log2 n*, da das Element mit zwei Children pro Level verglichen werden muss



A3 DoubleLinked List

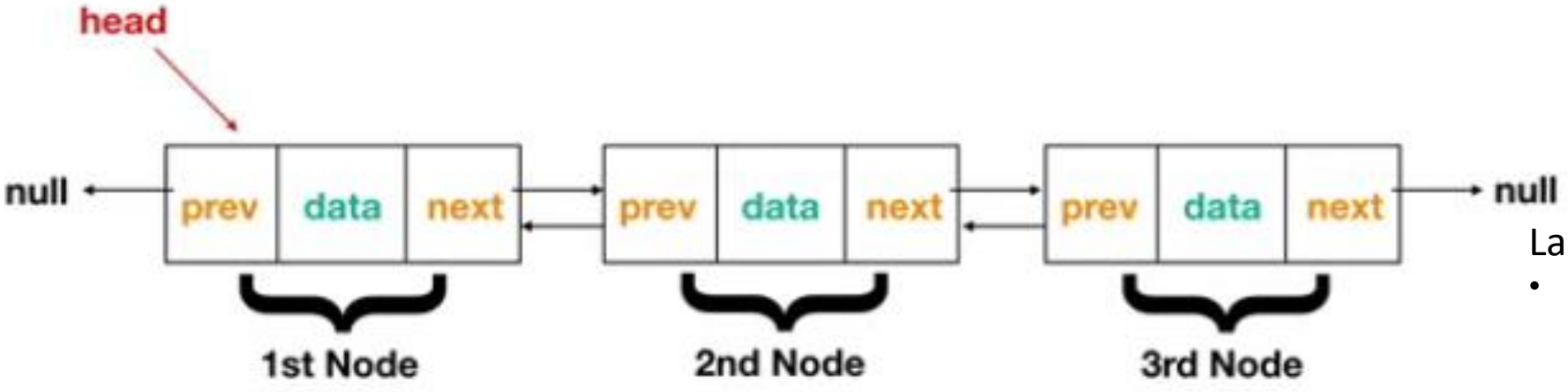
public void **add**(T a)

- 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 → (first == null)
  - Dann ist first = tmp und last = tmp

remove(int pos)

1.) wir iterieren bis zur Node die gelöscht werden soll
2.) wir setzen als previos vom next, das previos von pos
3.) 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)