Übungsblatt 4

Truong, Diebel

Aufgabe 2

a. Algorithm invert(L, head)
Input: LinkedList L, Node head is the middle of the List
Output: LinkedList L with inverted order of elements

```
let previous = null
let next = null
while (head != null) {
    // Find and keep the next node
    next = head->next;

    // Switch the next pointer to point backwards.
    head->next = previous;

    // Move both pointers forward.
    previous = head;
    head = next;
}
return L;
```

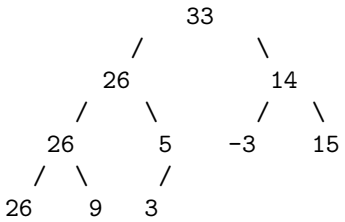
b.

Aufgabe 3

a. Array $A = [33, 26, 14, 26, 5, -3, 15, 26, 9, 3]$

Heapsize dekrementiert immer nach jeder Iteration, d.h die maximale potenzielle Heapsize ist in der erste Iteration

- Erste Iteration



- 2 Elementen(root und 26 auf die linke Seite) sind schon sortiert, d.h `A.heapsize = 8`

b. `max-heapify` wird `n-1` mal in $\mathcal{O}(logN)$ ausgeführt, also `A.heapsize/2 -1`

c. Algorithm Delete−Heap(A, i)
Input: Array A, index i in the array
Output: Heap−sorted Array A with element at index i removed

```
let temp = heap[1]
A.remove(i)                      //delete the specified element
for (int k = heap_size; K > 1 ; k--)
    swap(head[1],head[k])        //swapping with the last node in the tree
    heap_size = heap_size - 1
    max-heapify(A,i,heap_size)   //reheapify downward from root
end-for
return A
```

Begründung:

- Nach Entfernung eines Node, es kann sein dass die Ordnung des Heaps zerstört werden kann, zum Beispiel:

```
   60
  /  \
55    35
```

Wenn wir 60 entfernen, 35 wird zum `root` geschoben. Da $35 > 55$ ist, ist die Ordnung des Heaps zerstört

- Aus diesem Grund, nachdem wir ein Node entfernen, die `parents` Nodes sind geordnet aber die `children` nicht immer
- Dieses Problem kann behoben werden, wenn `heapify` nach diesem Punkt(i) nochmal ausgefürht ist