Aufgabe 2

Aufgabe 3

1. Für $A[1, n]$ nichtleere

Sei $A[1, n]$ hat nur ein element, also n = 1

$\Rightarrow A[1, n]$ ist einelementig $\Rightarrow$ i = n = 1 ist Gipfel des Arrays

Sei $A[1, n]$ hat mehr als ein Element in array, also n $\neq$ 1

- Falls der Array geordnet ist (zB {1,3,6,9,10,20}) dann ist der Gipfel entweder die 1. oder die letzte Element des Array, da die Element am großten ist
- Falls der Array nicht geordnet ist, dann es gibt immer mindesten ein Gipfel wenn die aufsteigende/absteigende Zuordnung des Array bricht (zB {1,2,6,3,4} hat 6 als Gipfel)

2.

```
peakFind(A[1,n], low, high)
Input: Array A[1,n], lower search index low, upper search index high
Output: index of first peak element in array
Time complexity: O(n)

let low = 0, mid = 1, high = 2
loop
    if (A[mid] > A[low] AND A[mid] > A[high])          //check if the middle element is a peak element
    return mid
    else low++, mid++, high++
```

3.

```
peakFind(A[1,n], low, high)
Input: Array A[1,n], lower search index low, upper search index high
Output: index of first peak element in array
Time complexity: O(log n)

let  mid = low + (high - low)/2                        //find middle index
    if (     (mid = 0 or arr[mid - 1] <= arr[mid])     //compare middle with neighbors
         AND (mid = n-1 or arr[mid+1] <= arr[mid]) )
        return mid
    else if ( mid > 0 AND arr[mid - 1] > arr[mid] )    //if left neighbor is greater then check
        return peakFind(arr, low, mid-1)               //left side first
    else reture peakFind(arr, mid+1, high)             //if right neighbor is greater then check
                                                       //right side first
```

- Linear scan
  - worst case $\mathcal{O}$(n): geordnete Array, then the last element would be a peak element
- Divine and Conquer
  - binary scan
  - Falls beide nachbarn Elements kleiner als der `middle` Element sind, dann ist der `middle` Element der Gipfel, sonst müssen wir zuerst nur die Halbe, die den Gipfel enthält, checken
  - worst case $\mathcal{O}$(log n) time