Aufgabe 10

gruppe Nikolic
Nofal
Hattinger

- Um lineare suche zu verbessern wird devide & conquer design benutzt, in dem man immer in der mitte von der liste (teilfolge oder array) schaut und vergleicht dieses element mit dem gesuchten element, wurde das element gefunden, wird die index zurückgegeben wurde nichts gefunden, wird es ~~wird~~ wieder halbiert aber es muss zu~~e~~erst überprüft werden ob das gesuchte element kleiner oder grösser ist ~~als~~ ~~ist~~ ist das gesuchte element kleiner als das element in der mitte wird in der linken seite (hälfte) weiter gesucht und halbiert, ist das gesuchte element grösser als das element in der mit, wird es nur in der rechten hälfte gesucht (und weiter halbiert falls nichts gefunden) solange wird es halbiert bis das element gefunden oder bis 1 erreicht wurde (sprich eine 1 elementige teilfolge)

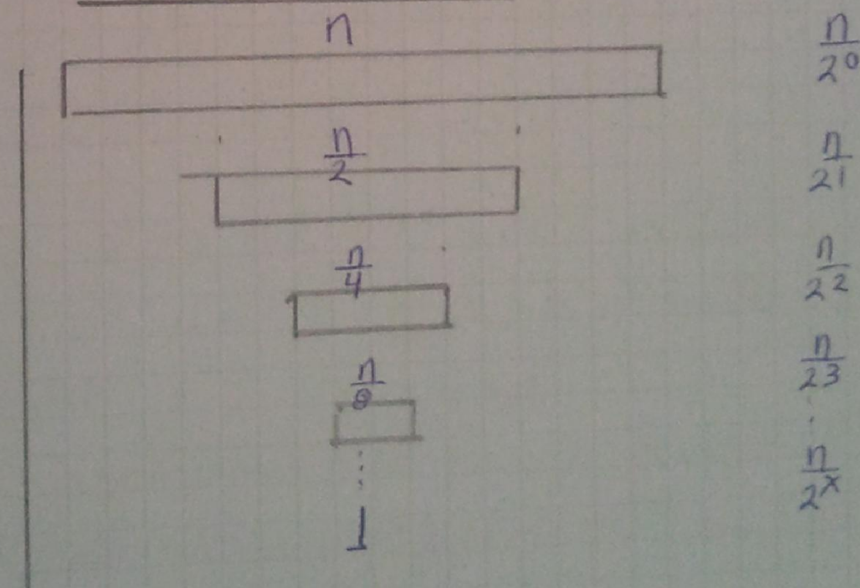
annahme: - die liste oder das array ist aufsteigend sortiert sonst ~~funktion~~ funktioniert den algorithmus nicht, das gibt's nämlich zwei implementierungen iterativ & Rekursiv

Iterativ

```
Binarysearch(A[lo..hi], x)
  While lo ≤ hi
     mid ← ⌊(lo+hi)/2⌋
     if x = A[mid]
       then return mid
     elseif x < A[mid]
          hi ← mid-1
     else
          low ← mid+1
  return "not found"
```

Rekursiv

```
Binarysearch(A[lo...hi], x)
  if lo > hi
    then return
  else
     mid ← ⌊(la+hi)/2⌋
  if x = A[mid]
    then return mid
  else if x < A[mid]
    then return Binarysearch(A, lo, mid-1, x)
  else
  then return Binarysearch(A, mid+1, hi, x)
```



Wie oft müssen wir das interval halbieren bis wir 1 erreichen aus dem obigen diagramm $\Rightarrow \frac{n}{2^x} = 1 \Rightarrow n = 2^x$

$n = 2^x \Rightarrow \log n = \log 2^x \Rightarrow \log n = x$

$\Rightarrow$ oder anderes mit hilfe von Rekurrence

der Algorithmus braucht $O(1)$ für vergleichen und dann wird das intervall halbiert

$\Rightarrow T(n) = T(\frac{n}{2}) + 1$

mit hilfe von Master theorem

$T(n) = aT(\frac{n}{b}) + cn^k$, in diesem fall

$a = 1,\ b = 2,\ k = 0$

$a = 1 = b^k = 2^0 = 1$

$\Rightarrow T(n) \in \Theta(n^k \log n)$

$\Rightarrow T(n) \in \Theta(\log n)$

im schlechsten fall (worstcase) läuft der Algorithmus von oben $n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \ldots 1$ ~~b~~ $\Rightarrow \log n$

Beweis (für Rekursive version)

Base case $n = 0 = b - a + 1$, array ist leer

$\Rightarrow a = b + 1 \Rightarrow a > b$ Algorithm endet

Inductive step :- $n = b - a + 1 > 0$

(IA) angenommen binarysearch(A, lo, hi, x) gibt den richtigen wert zurück $\forall J : 0 \leq J \leq n-1$ wobei $J = b - a + 1$

der Algorithmus berechnet $mid = \lfloor (a+b)/2 \rfloor$,

$a \leq mid \leq b$

Wenn $x = A[mid]$, dann ist $x \in A[lo, \dots, hi]$
und der Algorithmus gibt mid zurück

Wenn $x < A[mid]$, wir wissen dass A schon sortiert ist, x ist in $A[lo \dots hi]$ genau dann wenn $A[lo, \dots, mid-1]$ nach (IA) binarysearch$(A, a, mid-1, x)$ gibt den richtigen Wert zurück da $0 \leq (mid-1)-a+1 \leq n-1$

wenn $x > A[mid]$ ist analog zu $x < A[mid]$.

wzzw.

aufgabe 11

```
DualPivotQuicksort(A, left, right)

    if right-left >= 1
        p = A[left]
        q = A[right]
        if p > q then swap p and q   endif
        l = left+1
        g = right-1
        k = l
        while k <= g
            if A[k] < p
                swap A[k] and A[l]
                l = l+1
            else
                if A[k] > q
                    while A[g] > q and k < g do g = g-1 endwhile
                    swap A[k] and A[g]
                    g = g-1
                    if A[k] < p
                        swap A[k] and A[l]
                        l = l+1
                    endif
                endif
            end if
            k = k+1
        endwhile
        l = l-1
        g = g+1
        swap A[left] and A[l]
        swap A[right] and A[g]
        DualPivotQuicksort(A, left, l-1)
        DualPivotQuicksort(A, l+1, g-1)
        DualPivotQuicksort(A, g+1, right)
    endif
```