20. Folgende als Boolesche Funktion formulierte Funktion **binarysearch**

```
def binarysearch(value, array):
    if array == [] or (array[0] !=  value and len(array) == 1): return False
    else:
      midvalue = array[len(array)//2]
      if  value == midvalue: return True
      elif value < midvalue: return binarysearch(value, array[:len(array)//2])
      else:                  return binarysearch(value, array[len(array)//2 + 1:])
```

liefert den Wert **`True`**, falls die Suche erfolgreich war, andernfalls den Wert **`False`**.

a) Formuliere eine rekursiv definierte Funktion **binarysearch**, welche bei erfolgreicher Suche den ganzzahligen Index **`index`** ($0 \le$ index $\le$ n-1) derjenigen Komponente des Arrays **a** ausgibt, für die gilt: value = a[index], andernfalls für **`index`** den Wert -1 ausgibt (beachte: der Wert -1 kommt als Index des Arrays a nicht vor).

Anleitung:

Wenn **a[begin], ... , a[end]** der jeweilige Suchbereich innerhalb des Arrays **a** ist, wäre der Aufruf der Funktion **binarysearch** zur Suche von **value** im sortierten n-elementigen Feld **a[0], ... , a[n-1]:**
binarysearch(value, a, 0, n-1) oder **binarysearch(value, a, 0, len(a)-1)** .

Somit sind **value**, das gesamte Feld **a** und die jeweiligen Bereichsgrenzen bei jedem Aufruf an die Funktion binarysearch zu übergeben; Formulierung in Python:

```
def binarysearch(value, array, begin, end):
    . . . . . .
    . . . . . .
```

b) Implementiere die in a) formulierte Funktion binarysearch innerhalb des Algorithmus MergeSort; verwende hierzu als Quelltext z. B.
https://kalle2k.lima-city.de/computerscience/Informatik_12/2023-24/BinarySearch/MergeSort.py.txt
und teste das Programm für verschiedene Eingaben.

c) Ergänze das Programm, um die Anzahl der Aufrufe und den Zeitbedarf von **binarysearch** zu ermitteln.

d) Freiwillige Zusatzaufgabe: Konvertiere den Python-Quelltext mittels eines online-tools in einen C++-Quelltext; compiliere den C++-Quelltext zu einem ausführbaren Programm. Bestätige, daß eine Compiler-Sprache (wie z. B. C++) einer Interpreter-Sprache (wie z. B. Python) hinsichtlich des Zeitbedarfs während der Laufzeit überlegen ist.

21. Die Anzahl der Komponenten im zu sortierenden oder zu durchsuchenden Feld **a** beträgt **n**; allgemein ist diese Anzahl ein Maß für die „Problemgröße" der jeweiligen Aufgabenstellung.

Beispiele für die zeitliche Komplexität A(n) zur Laufzeit:

SelectionSort:	$A(n) \sim n^2$
Mergesort:	$A(n) \sim n \cdot \log_2(n)$
Fibonacci-Folge (rekursiv):	$A(n) \sim 2^n$
BinarySearch:	$A(n) \sim \log_2(n)$
Sequentielle Suche:	$A(n) \sim n$

Bilde für vorstehende Algorithmen jeweils die Quotienten

A(10 000)/A(1000)
A(100 000)/A(1000)
A(1 000 000)/A(1000)
A(1 000 000 000)/A(1000)

und beurteile die praktische Brauchbarkeit des jeweiligen Algorithmus bei wachsender Problemgröße.