Portfolio Prüfung II WWI2022B

Bitte bearbeiten Sie alle Aufgaben direkt hier im Notebook und geben Sie die .ipynb-Datei am Ende der Portfolio-Prüfung hier ab. Für Aufgaben die ohne Code erstellt werden, steht nach der Aufgabenstellung ein Markdown Antwortfeld zur Verfügung. Für Aufgaben bei denen ein Code verlangt wird, befindet sich nach der Aufgabenstellung ein interaktives Code-Feld (ggf. mit schon vorab ausgefüllten Code-Fragementen).

Viel Erfolg

Aufgabe 1 Python Basics

Geben Sie Ihren Namen und Ihre Matrikelnummer am Ende der Codezeile aus. Name und Matrikelnummer sollten in einer getrennten Zeile erscheinen. Beispielausgabe:

```
Name: Jana Musterfrau
MatrikelNummer: 123456789
```

In [ ]:
```python
# insert code for print here:
```

In [ ]:
```python
# please run this cell once. You will need the imports for the cells below.
import matplotlib.pyplot as plt
import numpy as np
import timeit
```

b) Plotten von Aufwand

Aus der Portfolio-Prüfung I kennen Sie die olgenden Funktionen `funktion1, funktion2, funktion3` :

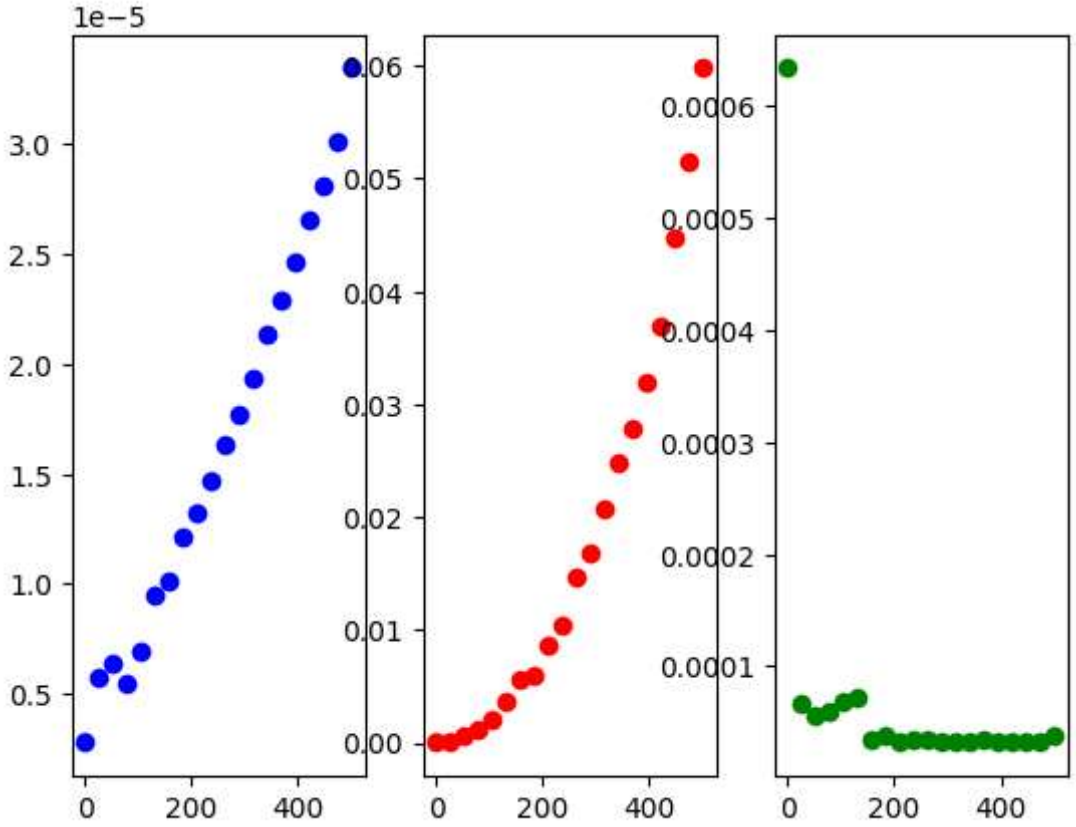
In [ ]:
```python
# run once
def funktion1(n): # O(n)
    retValue = 0
    for i in range(n,0,-1):
        retValue += i
    return retValue

def funktion2(n): # O(n^2)
    lst = []
    for i in range(n*n):
        lst.append(i)
        lst[i%n] += i
    return lst

def funktion3(n): # O(1)
    lst = []
    for i in range(100):
       lst.append(i)
       for j in range(1,100,n):
```

```
            lst[i] = j
    return lst
```

Erzeugen Sie ein Experiment um die Komplexität der drei Funktionen empirisch zu messen und erzeugen Sie eine Grafik die den Aufwand veranschaulicht. (Führen Sie die Code-Zelle der Funktionen einmal aus, sonst kennt die Code-Zelle für die Erzeugung der Grafik die Funktionen nicht.)

In [ ]:
```
ns = np.linspace(1, 500, 20, dtype=int)

ts1 = [timeit.timeit(stmt=f'funktion1({n})',
                     #setup= f'',
                     globals=globals(),
                     number=3)
       for n in ns]

ts2 = [timeit.timeit(stmt=f'funktion2({n})',
                     #setup= f'',
                     globals=globals(),
                     number=3)
       for n in ns]

ts3 = [timeit.timeit(stmt=f'funktion3({n})',
                     #setup= f'',
                     globals=globals(),
                     number=3)
       for n in ns]

fig, (ax1, ax2, ax3) = plt.subplots(1,3)
ax1.plot(ns, ts1, 'ob')
ax2.plot(ns,ts2,'or')
ax3.plot(ns,ts3,'og')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x164442243d0>]
```



Aufgabe 2 Stack und Rekursion

Gegeben ist eine Implementierung für einen verlinkten Stack. Eine Mischung aus einem Stack und einer Linked-List.

In [ ]:
```
class Stack:
    class Node:
        def __init__(self, val, next):
            self.val = val
            self.next = next

    def __init__(self):
        self.top = None

    def push(self, val):
        self.top = Stack.Node(val, self.top)

    def __iter__(self):
        s = self.top
        while s:
            yield s.val
            s = s.next

    def __repr__(self):
        return '[' + ', '.join(str(x) for x in self) + ']'
```

a) Vervollständigen der Definition

In der gegebenen Implementierung fehlt noch die Methode `pop()`, welche das oberste Element vom Stack nimmt, den Stack um dieses Element verkürzt und das Element

zurückliefert. Ergänzen Sie diese Methode:

In [ ]:
```
class Stack(Stack):

    def pop(self):
        val = self.top.val
        self.top = self.top.next
        return val
```

a) Die Methode `roll`

Implementieren Sie eine Methode `roll` als Teil des Linked-Stacks. Die Methode besitzt einen Parameter $n \geq 2$. Sie verschiebt das oberste Element des Stacks an die n-te Stelle des Stacks.

Beispiele:

- wendet man `roll(3)` auf einen Stack mit den Werten A, B, C, D, E (wobei A das oberste Element ist) an, so ergibt sich als Ergebnis B, C, A, D, E
- wendet man `roll(5)` auf einen Stack mit den Werten A, B, C, D, E an, so ergibt sich ein Stack mit den Werten (von oben nach unten) B, C, D, E, A

Einschränkungen/Annahmen:

- Ihre Implementierung darf keine Werte von Knoten (`node.val`) ändern. Stattdessen sollen lediglich die Verknüpfungen der Werte (`node.next`) verändert werden.
- Gehen Sie davon aus, dass n mindestens den Wert 2 hat, und dass der Stack mindestens n-Werte hat.

In [ ]:
```
class Stack(Stack):
    def roll(self, n):
        tempNode = self.top
        secondNode = self.top.next
        for i in range(n-1):
            tempNode = tempNode.next

        self.top.next = tempNode.next
        tempNode.next = self.top
        self.top = secondNode
```

In [ ]:
```
# you can check your implementation with the following code:
myStack = Stack()
for i in range(5):
    myStack.push(i)
print(myStack) # should print [4, 3, 2, 1, 0]
myStack.roll(2)
print(myStack) # should print [3, 4, 2, 1, 0]
myStack.roll(3)
print(myStack) # should print [4, 2, 3, 1, 0]
```

```
[4, 3, 2, 1, 0]
[3, 4, 2, 1, 0]
[4, 2, 3, 1, 0]
```

b) Rekursion durch Stack abbilden

Aus der Vorlesung kennen Sie die Fibonacci-Folge, die rekursiv über folgenden Code erzeugt werden kann:

In [ ]:
```python
def fibonacciRecursive(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacciRecursive(n-1) + fibonacciRecursive(n-2)
```

Jede rekursive Funktion kann auch iterativ implementiert werden, in dem man mehrere Schleifen und (oft) einen Stack verwendet. Verwenden Sie die oben gegebene Implementierung des Linked-Stacks um die Fibonacci-Folge ohne Rekursion, sondern mithilfe eines Stacks zu implementieren.

In [ ]:
```python
def fibbonacciIterative(n):
    stack = [n]
    result = 0
    while stack:
        #print(stack)
        n = stack.pop()
        if n == 0:
            result += 0
        elif n == 1:
            result += 1
        else:
            stack.append(n-1)
            stack.append(n-2)
    return result
```

In [ ]:
```python
# you can check the first 11 values of your implementation with the following li
for i in range(11):
    assert fibbonacciIterative(i) == fibonacciRecursive(i)
```

Aufgabe 3 Linear vs Binary Search

In dieser Aufgabe sollen die lineare mit der binären Suche verglichen werden.

a) Lineare und Binäre Suche implementieren

Vervollständigen Sie den unten stehenden Code und implementieren sie jeweils die lineare Suche und die binäre Suche:

In [ ]:
```python
def linearSearch(lst, x):
   """searches for x in the given lst using linear search
      lst must be a simple python lst
```

```
      returns True if lst contains x, else False
   """
   for y in lst:
      if x == y:
         return True
   else:
      return False

def binarySearch(lst, x):
   """searches for x in the given lst using binary search
      lst must be a simple python lst
      returns True if lst contains x, else False
   """
   lo = 0
   hi = len(lst) - 1
   while lo <= hi:
      mid = (lo + hi) // 2
      if lst[mid] == x:
         return True
      elif lst[mid] > x:
         hi = mid - 1
      else: # lst[mid] < x:
         lo = mid + 1
   else:
      return False
```

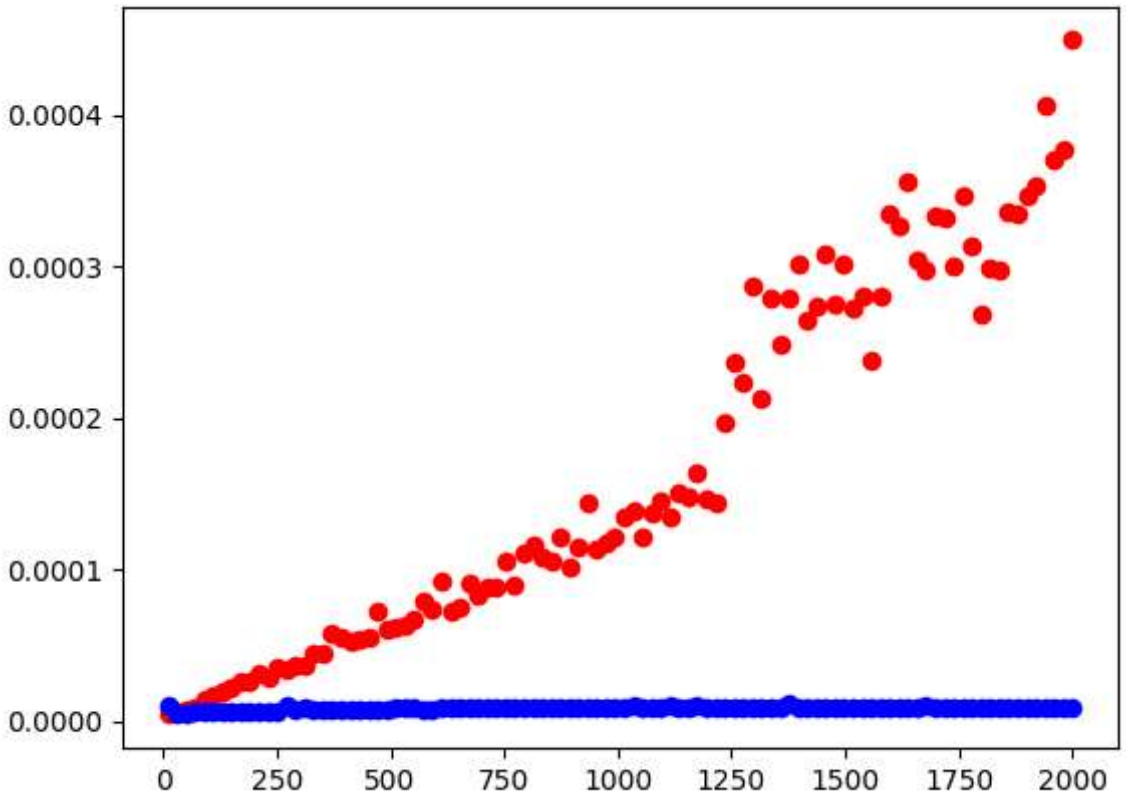
In [ ]:
```
# check your code with the following lines
myList = [1,2,3,4,5]
assert linearSearch(myList,3) == True
assert linearSearch(myList,6) == False
assert binarySearch(myList,3) == True
assert binarySearch(myList,6) == False
```

b) Komplexität bei der Suche nach einem Randwert

Visualisieren Sie den Aufwand der beiden verschiedenen Such-Algorithmen, wenn in einer Liste von aufsteigenden Zahlen, nach dem höchsten Element gesucht wird. Beispiel: Suche nach der Zahl 5 in der Liste [1,2,3,4,5]

In [ ]:
```
# runtimes when searching for an edge-value in lists of increasing size

ns = np.linspace(10, 2000, 100, dtype=int)
ts_linear = [timeit.timeit(f'linearSearch(lst, {n})',
                    setup=f'lst=list(range({n}))',
                    globals=globals(),
                    number=10)
      for n in ns]

ts_binary = [timeit.timeit(f'binarySearch(lst, {n})',
                    setup=f'lst=list(range({n}))',
                    globals=globals(),
                    number=10)
      for n in ns]

plt.plot(ns, ts_linear, 'or')
plt.plot(ns, ts_binary, 'ob')
```

Out[ ]: [<matplotlib.lines.Line2D at 0x1644424a450>]

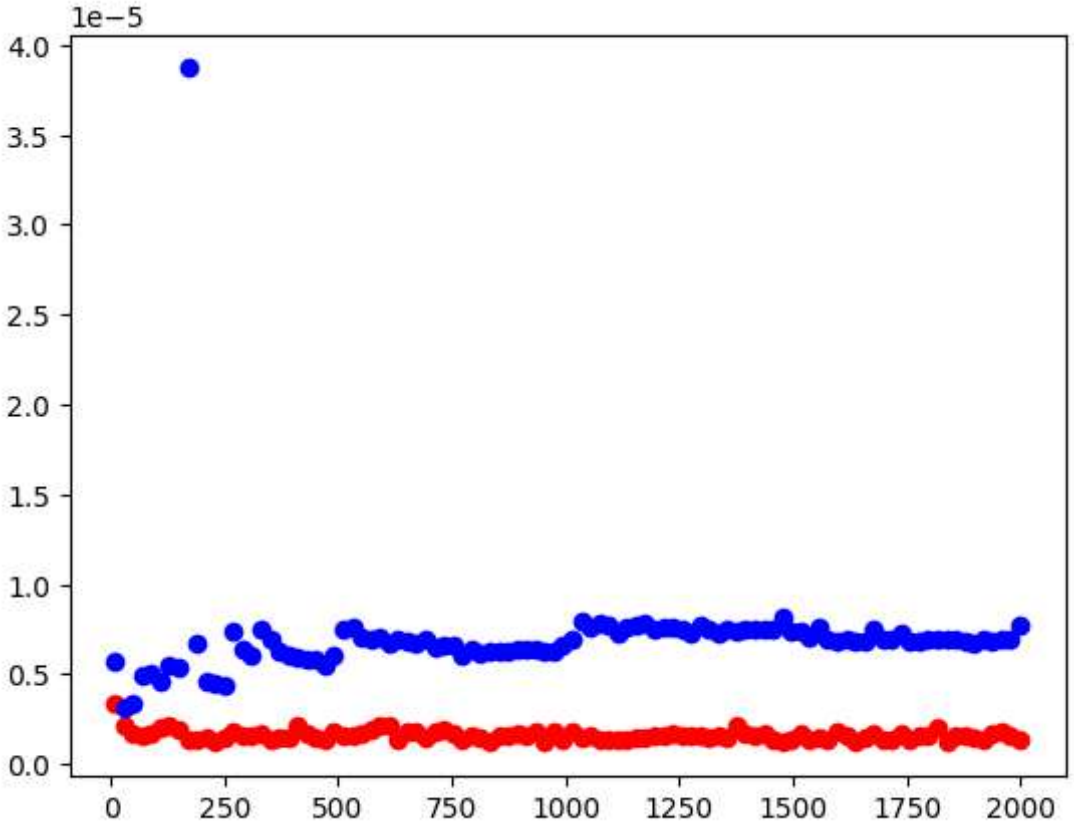


c) Beispiel für schnellere lineare Suche

Konstruieren Sie ein Experiment, bei dem der Aufwand der linearen Suche geringer ist, als der der binären Suche, selbst bei größer werdenden Listen:

In [ ]:
```
# example where linear search is faster than binary
# easiest way, search for the first element in a list

ns = np.linspace(10, 2000, 100, dtype=int)
ts_linear = [timeit.timeit(f'linearSearch(lst, 2)',
                    setup=f'lst=list(range({n}))',
                    globals=globals(),
                    number=10)
      for n in ns]

ts_binary = [timeit.timeit(f'binarySearch(lst, 2)',
                    setup=f'lst=list(range({n}))',
                    globals=globals(),
                    number=10)
      for n in ns]

plt.plot(ns, ts_linear, 'or')
plt.plot(ns, ts_binary, 'ob')
```

Out[ ]: [<matplotlib.lines.Line2D at 0x16441837d50>]



Abgabe

Bitte denken Sie daran, ihre Notebook-Datei auf den Abgabe-Server hochzuladen.

https://privacy.dhbw-stuttgart.de/wwi2022b.html