Übung 1 zu KMPS

Aufgabe 1: iterative Quicksort

```
package uebung1;

import java.util.Stack;

public class IterativeQuickSort {
   private int a[];
   public IterativeQuickSort(int arr[]) {
      a = arr;
      quickSort();
   }
   public String toString(){
      String s = "";
      for (int i = 0; i < a.length; i++) {
         s += a[i];
         s += (i != a.length-1)? ", ":".";
      }
      return s;
   }

   public void swap(int i, int j){
      int tmp = a[j];
      a[j] = a[i];
      a[i] = tmp;
   }

   public int findPivotPosition(int low, int high){
      int pivot = a[high];
      int index_bigger = low - 1;
      int pivot_index = high;
      for (int j = low; j < high; j++){
         if (a[j] <= pivot){
            index_bigger++;
            swap(index_bigger, j);
         }
      }
      pivot_index = index_bigger+1;
```

Übung 1 zu KMPS

```
        swap(pivot_index, high);
        return pivot_index;
    }

    public void quickSort(){
        Stack stack = new Stack<Integer>();
        stack.push(0);
        stack.push(a.length-1);
        while(!stack.isEmpty()){
            int high = (int) stack.pop();
            int low = (int) stack.pop();
            int p = findPivotPosition(low, high);
            if(p != 0 && (p-1) > low){
                stack.push(low);
                stack.push(p-1);
            }
            if(p != a.length-1 && (p+1) < high){
                stack.push(p+1);
                stack.push(high);
            }
        }
    }

    public static void main(String[] args) {
        int arr[] = {1, 3, 2, 0, 9, 0, 8, 6, 1, 9};
        IterativeQuickSort sort = new IterativeQuickSort(arr);
        System.out.println(sort.toString());
    }
}
```

Aufgabe 2:

Unit ist so ähnlich wie void also für ohne Rückgabe Funktion.

Aufgabe 3:

a. [1| [2| [3|[ ]]]]
b. Element::Liste -> rechtsassoziativ

Aufgabe 4:

Übung 1 zu KMPS

```
package uebung1;

public class aufgabe4 {
   private int i = 0;
   public int add(int j){
      i += j;
      return i;
   }
   public static void main(String[] args) {
      aufgabe4 a = new aufgabe4();
      System.out.println(a.add(1));  //output:1
      System.out.println(a.add(1));  //output:2
   }
}
```

Aufgabe 5:

?-square(4,A), square(A,B)

Aufgabe 6:

```
a. package uebung1;


   public class IterativeFibonacci {
       private int a[];
       private int n;
       public IterativeFibonacci(int n){
           this.n = n;
           a = new int[n];
           fibonacci();
       }
       public void fibonacci(){
           for(int i=0; i<n; i++){
               if(i == 0){
                   a[i] = 0;
               }
               else if(i == 1){
                   a[i] = 1;
               }
               else{
                   a[i] = a[i-1] + a[i-2];
               }
           }
       }
       public String toString(){
           String s = "";
           for (int i = 0; i < a.length; i++) {
               s += a[i];
```

```
                s += (i != a.length-1)? ", ":".";
            }
            return s;
        }

        public int getFibonacci(){
            return a[n-1];
        }
        public static void main(String[] args) {
            IterativeFibonacci fib = new IterativeFibonacci(10);
            System.out.println(fib);
            System.out.println(fib.getFibonacci());
        }
    }

    Zeitsaufwand: linear O(n)
    Speicheraufwand: const O(1)(ohne Array)oder linear O(n)(mit Array)
```

b.

def fib(n: Int) : Int = if (n==0 || n==1) n else fib(n-1) + fib(n-2)

Zeitsaufwand: exponentiell O(2^n)

Speicheraufwand: linear O(n)

Aufgabe 7:

def ggTRekursiv(a: Int, b: Int) : Int = if(b==0) a else ggTRekusiv(b, a%b)

Zeitsaufwand: linear O(n)

Speicheraufwand: const O(1)