

Rekursive Algorithmen

Einstiegsaufgabe 1: Das Processing-Programm *Einstiegsaufgabe1* enthält eine Operation `zeichne_rekursiv` und eine Operation `zeichne_iterativ`. Die Operation `zeichne_rekursiv` wird ausgeführt, wenn die linke Maustaste gedrückt werden. Die Operation `zeichne_iterativ` wird ausgeführt, wenn die rechte Maustaste gedrückt werden.

- Führen Sie das Programm aus und beobachten Sie die Ausgaben, die beim Klicken der linken oder rechten Maustaste erzeugt werden.
- Vergleichen Sie die Methoden `zeichne_rekursiv` und `zeichne_iterativ` miteinander.

```
void zeichne_rekursiv(int x, int y, int anzahl){
    if(anzahl <= 1)
        square(x, y, 20);
    else{
        square(x, y, 20);
        zeichne_rekursiv(x+20, y, anzahl - 1);
    }
}

void zeichne_iterativ(int x, int y, int anzahl){
    for(int i = anzahl; i > 0; i--){
        square(x, y, 20);
        x = x + 20;
    }
}
```

Abbildung 1: Die Operationen `zeichne_rekursiv` und `zeichne_iterativ`

Iterativer vs. rekursiver Ansatz

Rekursiv und iterativ beschreiben jeweils eine bestimmte, prinzipielle Arbeitsweise von Algorithmen, bei denen das wiederholte Ausführen von Befehlen eine Rolle spielt. **Iterative Algorithmen** arbeiten mit Schleifen, um Wiederholungen auszuführen. Diese Art von Algorithmen haben Sie vermutlich bereits häufiger erstellt. **Rekursive Algorithmen** bzw. **Operationen** rufen sich hingegen selbst wieder auf, um Wiederholungen umzusetzen. Damit dieser Selbstaufruf nicht endlos fortgesetzt wird, sollte der Selbstaufruf für ein kleineres Problem erfolgen, so dass das Problem irgendwann so klein ist, dass es direkt ohne einen weiteren rekursiven Aufruf gelöst werden kann. Eine Rekursion benötigt daher eine **Abbruchbedingung**. Wenn die Abbruchbedingung erfüllt ist, erfolgt kein erneuter rekursiver Aufruf. Man spricht hier auch von **Basisfall**, während der rekursive Aufruf als **Rekursionsschritt** bezeichnet wird.

Aufgabe1:

- Erläutern Sie das algorithmische Vorgehen der Operation `zeichne_rekursiv` aus der Einstiegsaufgabe 1 in Abbildung 1 mithilfe der Begriffe *rekursiver Aufruf* und *Abbruchbedingung*.
- Beschreiben Sie, inwiefern das Problem für den rekursiven Aufruf verkleinert wird. Was wäre in diesem Fall das kleinste Problem, das direkt gelöst werden kann?
- Abbildung 2 zeigt eine kürzere Schreibweise der rekursiv arbeitenden Operation `zeichne_rekursiv`. Erläutern Sie die vorgenommenen Änderungen.

```
void zeichne_rekursiv(int x, int y, int anzahl){  
    square(x, y, 20);  
    if(anzahl > 1)  
        zeichne_rekursiv(x+20, y, anzahl - 1);  
}
```

Abbildung 2: Alternative rekursive Implementierung der Operation `zeichne_rekursiv`

Gleichmächtigkeit von iterativen und rekursiven Algorithmen

Ein rekursiver Algorithmus lässt sich in einen iterativen Algorithmus, der mit Schleifen arbeitet, umwandeln und umgekehrt. Je nach Problemstellung fällt es manchmal leichter, einen rekursiven oder einen iterativen Algorithmus zu finden.

Aufgabe 2: Abbildung 3 zeigt einen Ausschnitt des Programms *geschachtelteQuadrate* zum Zeichnen geschachtelter Quadrate. Verändern Sie die Operation `zeichneQuadrate` so, dass eine Schleife statt einer Rekursion verwendet wird.

```
void mouseClicked(){  
    zeichneQuadrate(mouseX, mouseY, 60);  
}  
  
void zeichneQuadrate (int x, int y, int breite){  
    square(x, y, breite);  
    if(breite > 10){  
        zeichneQuadrate (x+5, y+5, breite - 10);  
    }  
}
```

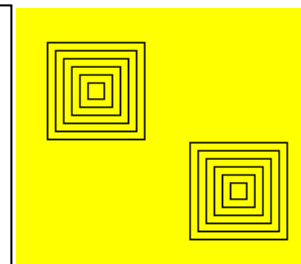


Abbildung 3: Ausschnitt des Programms zum Zeichnen geschachtelter Quadrate und exemplarische Ausgabe

Aufgabe 3: Erstellen Sie zwei Varianten eines Programms, das das Fenster mit Rechenkästchen füllt (vgl. Abb. 4). Die eine Variante soll das Problem iterativ (d. h. mithilfe von Schleifen), die andere Variante soll es rekursiv lösen.

Tipp: Erzeugen Sie die Rechenkästchen aus senkrechten und waagerechten Linien. Dabei können die Systemvariablen `width` und `height` hilfreich sein, welche die Breite und Höhe des Fensters enthalten.



Abbildung 4: Exemplarische Ausgabe zu Aufgabe 3

Aufgabe 4: Erstellen Sie ein Programm mit einer rekursiven Operation `zeichneKreise`, welche mehrere Kreise ineinander schachtelt, deren Füllung immer dunkler wird.

Rufen Sie die Operation `zeichneKreise` mit geeigneten Zufallswerten in der Methode `draw` auf, um eine Animation zu erzeugen (vgl. Abbildung 5).

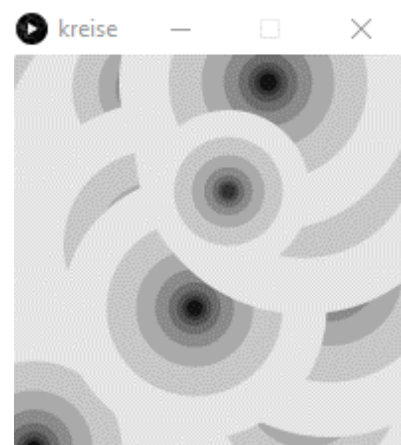


Abbildung 5: Exemplarische Ausgabe zu Aufgabe 4

Aufgabe 5*: Die Figur in Abbildung 6 ist unter dem Namen Sierpinski-Dreieck bekannt (benannt nach Waclaw Sierpiński).

Erstellen Sie ein Programm, das rekursiv Sierpinski-Dreiecke zeichnet. Dazu werden in einem gleichseitigen Dreieck die Mittelpunkte der Seiten wieder zu einem gleichseitigen Dreieck verbunden, bis eine bestimmte Rekursionstiefe erreicht ist. Diese soll der Anwender wählen können.

Hinweis: Um die rekursive Struktur zu veranschaulichen, ist in Abbildung 7 zunächst ein Sierpinski-Dreieck mit Rekursionstiefe 1 abgebildet.

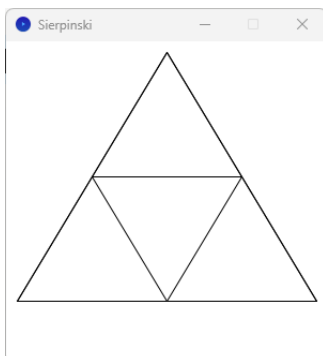


Abbildung 7: Sierpinski-Dreieck mit Rekursionstiefe 1

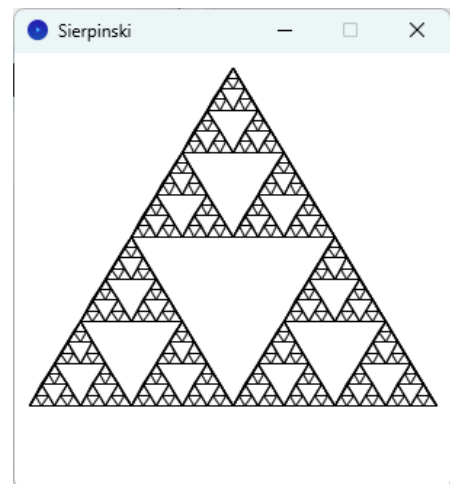


Abbildung 6: Sierpinski-Dreieck mit Rekursionstiefe 5

Rekursiver Auf- und Abstieg

Einstiegsaufgabe 2: Um den Durchschnittswert einer Reihe von Messwerten zu berechnen, muss zunächst die Summe der Messwerte gebildet werden. Abbildung 8 zeigt ein entsprechendes Programm, das die Werte in einer global definierten Reihung `messwerte` vom Typ Fließkommazahl zunächst mithilfe einer rekursiv definierten Operation `sum` addiert.

- a) Beschreiben Sie Gemeinsamkeiten und Unterschiede der rekursiv arbeitenden Operation `sum` zu den rekursiven Operationen, die Sie bisher erstellt und verwendet haben.

```
float[] messwerte = {20.4, 23.5, 24.3, 25.6, 24.8};  
void setup(){  
    float gesamtsumme = sum(messwerte.length-1);  
    float durchschnitt = gesamtsumme / messwerte.length;  
    println(durchschnitt);  
}  
float sum(int index){  
    if(index == 0) return messwerte[0];  
    else{  
        float summe = messwerte[index] + sum(index-1);  
        return summe;  
    }  
}
```

Abbildung 8: Programm zur Berechnung des Durchschnitts einer Reihe von Messwerten

- b) Erläutern Sie die Arbeitsweise der Operation `sum` mithilfe der Grafik in Abbildung 9.

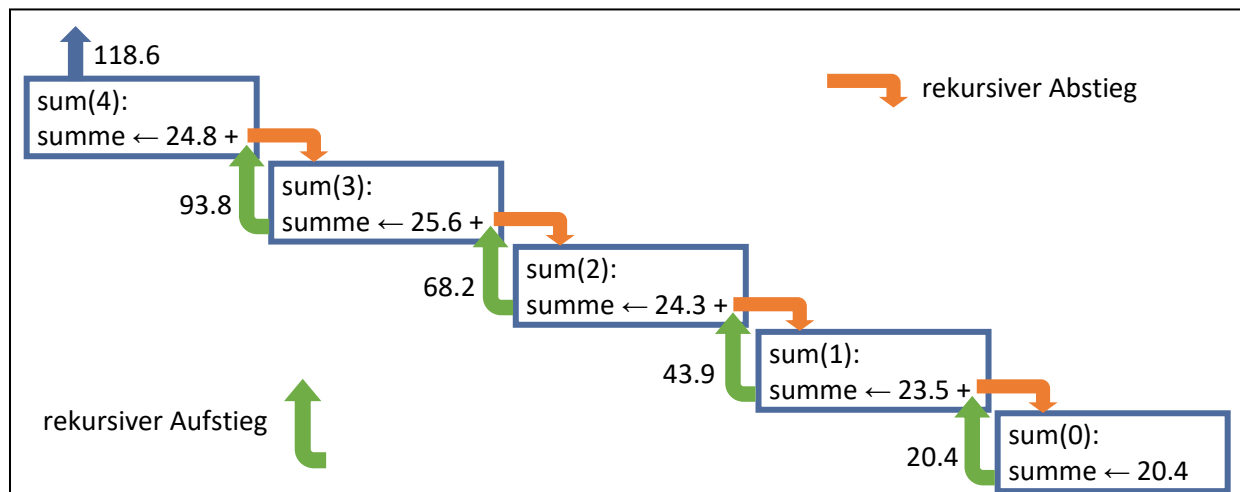


Abbildung 9: Visualisierung der Arbeitsweise der rekursiven Operation `sum`

- c) Rekursive Algorithmen gelten als speicherintensiv. Begründen Sie diese Aussage, indem Sie den Speicherbedarf der rekursiven Operation `sum` mit der iterativen Variante der Operation in Abbildung 10 vergleichen.

```
float sum_iterativ(){  
    float summe = 0;  
    for(int i = 0; i < messwerte.length; i++){  
        summe = summe + messwerte[i];  
    }  
    return summe;  
}
```

Abbildung 10: iterative Implementierung der Operation `sum`

Aufgabe 6: Einige mathematische Funktionen lassen sich rekursiv übersichtlich definieren. Ein Beispiel ist die Fakultätsfunktion $n!$, die für alle natürlichen Zahlen n und die Zahl 0 definiert ist.

$$n! = \begin{cases} 1, & \text{für } n = 0 \text{ oder } n = 1 \\ n * (n-1)! & \text{für } n > 1 \end{cases}$$

- Implementieren Sie eine entsprechende rekursive Operation:
`fakultaet(n: Ganzzahl): Ganzzahl`
- Skizzieren Sie den rekursiven Ab- und Aufstieg für den Aufruf `fakultaet(4)` ähnlich wie in Abbildung 9.
- Je nach Datentyp für den Rückgabewert (`int` bzw. `long`) liefert die Operation nur für $n \leq 12$ bzw. $n \leq 20$ sinnvolle Werte. Begründen Sie.

Aufgabe 7: Ergänzen Sie in dem Programm aus Einstiegsaufgabe 2 eine rekursiv arbeitende Operation, die das Maximum der Messwerte bestimmt.

Aufgabe 8:

- Die folgende fehlerhafte Implementierung einer Operation `summeN(n: Ganzzahl): Ganzzahl` in Abbildung 11, welche die Werte von 1 bis n addieren soll, erzeugt die Fehlermeldung in Abbildung 12. Erläutern Sie und korrigieren Sie die Implementierung der Operation geeignet.

```
int summeN(int n){  
    if(n == 1) return 1;  
    else return n-1 + summeN(n);  
}
```

Abbildung 11: fehlerhafte Implementierung der Operation `summeN`

A `StackOverflowError` means that you have a bug that's causing a function to be called recursively (it's calling itself and going in circles), or you're intentionally calling a recursive function too much, and your code should be rewritten in a more efficient manner.
`StackOverflowError`

Abbildung 12: Fehlermeldung, die von der Operation in Abbildung 11 ausgelöst wird.

- Führen Sie die korrigierte Operation `summeN` für die Werte 3.000 und 30.000 aus. Welches Problem tritt dabei auf? Erläutern Sie.
- Erstellen Sie eine iterative Implementierung für die Operation `summeN`. Testen Sie auch diese für die Eingaben 3.000 und 30.000.

Aufgabe 9: Eine Bank bietet für ein Festgeldkonto eine jährliche Verzinsung von 3% an. Erstellen Sie eine rekursive Operation `berechneGuthaben(startbetrag: Fließkommazahl, n: Ganzzahl): Fließkommazahl`, die für einen Startbetrag, der zu Beginn eingezahlt wird, das Guthaben nach n Jahren berechnet. Testen Sie Ihre Operation mit verschiedenen Startwerten und Jahren n .

Das Prinzip Teile und Herrsche

Eine Strategie beim Entwerfen rekursiver Algorithmen ist das Prinzip *Teile und Herrsche*. Die Idee dieser Strategie ist, ein Problem bei jedem rekursiven Aufruf in zwei Teilprobleme zu zerlegen, die möglichst nur noch halb so groß sind, wie das ursprüngliche Problem. Dies geschieht so lange, bis das Problem so klein ist, dass es direkt gelöst werden kann. Anschließend wird beim rekursiven Aufstieg die Lösung aus den Lösungen der Teilprobleme zusammengesetzt.

Das Teilen erfolgt also in der Phase des rekursiven Abstiegs. Das Herrschen bezieht sich auf das Lösen der kleinsten Probleme und das Zusammensetzen der Lösung aus Teillösungen.

Abbildung 13 zeigt ein Beispiel für eine rekursive Operation, die die n -te Potenz einer Zahl x nach dem Prinzip *Teile und Herrsche* berechnet.

```
int potenz(int x, int n) {  
    if (n == 0) return 1;  
    if (n == 1) return x;  
    int p = potenz(x, n / 2);  
    if (n % 2 == 0) {  
        return p * p;  
    } else {  
        return x * p * p;  
    }  
}
```

Abbildung 13: rekursive Operation *potenz*, die nach dem Prinzip *Teile und Herrsche* arbeitet

Aufgabe 10:

- Visualisieren Sie den rekursiven Ab- und Aufstieg der Operationsaufrufe `potenz(3, 8)` und `potenz(3, 10)`.
- Beschreiben Sie, wie die Operation `potenz` das Prinzip *Teile und Herrsche* umsetzt.
- Bestimmen Sie die maximale Rekursionstiefe in Abhängigkeit des Exponenten n .
Hinweis: Als Rekursionstiefe bezeichnet man die maximale Anzahl der rekursiven Selbstaufrufe bis der Basisfall erreicht bzw. die Abbruchbedingung erfüllt ist.

Aufgabe 11: In einer sortierten Reihung kann effizient nach Werten gesucht werden.

- Spiele Sie mit Ihrem Gegenüber einige Runden das Spiel *Zahlenraten*: Bei dem Spiel *Zahlenraten* denkt sich eine Person eine Zahl zwischen 0 und 100. Die andere Person versucht die Zahl mit möglichst wenig Versuchen zu erraten. Nach jedem Fehlversuch erhält die ratende Person einen Hinweis, ob die gesuchte Zahl kleiner oder größer ist.
- Tauschen Sie sich über Ihr Vorgehen aus. Welche Strategie haben Sie verwendet?
- Entwickeln Sie basierend auf Ihren Überlegungen zu b) einen rekursiven Algorithmus in Form eines Struktogramms, der in einer global definierten, sortierten Reihung `zahlen` vom Inhaltstyp `Ganzzahl` effizient nach dem Index einer als Parameter übergebenen Zahl sucht.
- Visualisieren Sie jeweils den Ablauf Ihres Algorithmus für die Suche nach den Werten 14 bzw. 23 in der Reihung `zahlen`: [2, 5, 10, 12, 14, 17, 20, 23].

Aufgabe 12: Entwickeln Sie eine Operation, die rekursiv nach dem Prinzip *Teile und Herrsche* den kleinsten Wert in einer unsortierten, global definierten Reihung `zahlen` bestimmt.

Aufgabe 13:

- a) Informieren Sie sich über die Fibonacci-Folge.
- b) Geben Sie die rekursive Definition zur Berechnung der n-ten Fibonacci-Zahl an.
- c) Implementieren Sie eine rekursive Operation `fib(n: Ganzzahl) : Ganzzahl`, welche die n-te Fibonacci-Zahl berechnet.
- d) Veranschaulichen Sie die Arbeitsweise der Operation `fib` für die Berechnung der 5-ten Fibonacci-Zahl.
- e) Begründen Sie, dass die rekursive Berechnung der n-ten Fibonacci-Zahl ineffizient ist.

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](https://creativecommons.org/licenses/by-nc-sa/4.0/). Von der Lizenz ausgenommen ist das InfSII-Logo.

Für die korrekte Ausführbarkeit der Quelltexte in diesem Arbeitsblatt wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.