Serie 7

Aufgabe 1:

Schreiben Sie eine rekursive und eine nicht-rekursive Funktion, welche das n-te Element der Recaman-Folge berechnet. Die Recaman-Folge ist definiert durch:

- $a_0 = 0$
- $a_n = a_{n-1} - n$, wenn $a_{n-1} - n > 0$ und $a_{n-1} - n$ nicht bereits in der Folge vorkommt
- $a_n = a_{n-1} + n$, sonst

Welche der beiden Implementierungen kann größere Werte von n berechnen ohne einen Fehler zu erzeugen? Begründen Sie Ihre Antwort.

Aufgabe 2:

Schreiben Sie eine Funktion, die zwei bereits aufsteigend sortierte Listen, vereinigt, sodass die resultierende Liste ebenfalls aufsteigend sortiert ist. Geben Sie diese Liste zurück.

Achtung: die Listen müssen nicht die selbe Länge haben. Die Funktion soll nicht die eingebauten Sortierfunktionen verwenden.

Aufgabe 3:

Merge-Sort: Schreiben Sie eine rekursive Implementierung des Merge-Sort Algorithmus. Dieser funktioniert nach dem *divide and conquer* Prinzip wie folgt: Gegeben sei eine Liste der Länge n für $n \in \mathbb{N}$.

1. Erstelle zwei Unterlisten, die jeweils ca. die Hälfte der Elemente der ursprünglichen Liste enthalten (falls n ungerade ist, enthält eine Liste ein Element mehr).
2. Sortiere beide Listen mithilfe des Merge-Sort Algorithmus.
3. Füge die sortierten Listen wieder zu einer sortierten Liste zusammen, i.e. **Aufgabe 2** (Englisch: *merge*)

Bonus 1: Stoppen Sie die Laufzeit ihres Algorithmus für Listen der Länge $n = 2^k$ mit $k = 10, 11, \ldots$. Wie ist das Verhältnis von Laufzeit und n.

Aufgabe 4:

Nach einer alten Legende gibt es in Hanoi einen Tempel, in dem sich folgende rituelle Anordnung befindet: Es handelt sich um 3 Stäbe auf denen 64 goldene Scheiben mit

paarweise verschiedenen Durchmesser gestapelt werden. Als der Tempel erbaut wurde, wurden diese Scheiben der Größe nach aufsteigend auf den ersten Stab gestapelt. Die Aufgabe der Priester in diesem Tempel besteht darin, diese Scheiben systematisch unter Zuhilfenahme des zweiten Stabes so umzustapeln, dass sich diese am Schluss in der anfänglichen Anordnung auf dem dritten Stab befinden. Dabei sind folgende Regeln zu beachten:

1. Die Scheiben dürfen nur einzeln verschoben werden.
2. In jedem Schritt wird die oberste Scheibe eines Stapels auf einen der anderen Stapel gesetzt.
3. Eine Scheibe darf nie auf einer anderen Scheibe kleineren Durchmessers zu liegen kommen

Das Ende der Welt, so sagt die Legende, ist durch denjenigen Zeitpunkt vorherbestimmt, an dem die Priester diese Aufgabe erfüllt haben werden. Wir bezeichnen mit $n \in \mathbb{N}$ die Anzahl der Scheiben ($n = 64$ in der Legende, am Computer eher $n = 3, 4, \ldots$). Ein rekursiver Algorithmus, der dieses Problem löst, lässt sich wie folgt formulieren: Um die obersten $m \leq n$ Scheiben von Stab i auf Stab j zu verschieben, verschiebt man

1. die obersten $m - 1$ Scheiben von Stab i auf Stab $k \in \{i, j\}$,
2. dann die grösste Scheibe von Stab i auf Stab j,
3. und schliesslich die $m - 1$ in Schritt (a) auf Stab k verschobenen Scheiben auf Stab j.

Nutzen Sie für die Implementierung eine geschachtelte Liste `[[],[],[]]`. Jede Unterliste steht für einen der Stäbe. Geben Sie nach jedem Schritt aus, welcher Zug ausgeführt wurde. Beispiel: `"Bewege Scheibe von Stab 3 zu Stab 2"`.

Aufgabe 5:

Für eine differenzierbare Funktion $f : \mathbb{R} \to \mathbb{R}$ kann man die Ableitung $f'(x)$ an einem festen Punkt $x \in \mathbb{R}$ durch den einseitigen Differenzenquotienten

$$\Phi(h) := \frac{f(x + h) - f(x)}{h},$$

für $h > 0$ approximiert werden. Schreiben Sie eine Funktion `diff(f,h0,eps)`, die die Folge $\Phi(h_n)$ für $h_n = 2^{-n} h_0$ berechnet bis gilt

$$|\Phi(h_n) - \Phi(h_{n+1})| \leq \epsilon |\Phi(h_n)|.$$

Aufgabe 6:

Schauen Sie sich nochmal das Beispiel, **Finde den kürzesten Weg** in der Vorlesung an.

Nehmen Sie das Beispiel als Gerüst und erweitern Sie den Code so, dass auch diagonale Schritte möglich sind. Nutzen Sie dabei, dass diagonale Nachbarn $\sqrt{2}$ voneinander

entfernt sind. Finden und Visualisieren Sie den kürzesten Weg mit dieser Variante.

- Erweitern Sie die Funktion `update()` aus der Vorlesung so, dass auch diagonale Nachbarn berücksichtigt werden.
- Erweitern Sie die Funktion `find_shortest_path()` so, dass auch diagonale Schritte gefunden werden.

Sie können die Funktion `print_maze` zur Visualisierung verwenden (Wenn Sie stattdesssen die Funktion `print_maze` aus der Vorlesung verwenden wollen, müssen Sie zuerst das Modul `matplotlib` installieren).

In [1]:
```
n=10
maze = [[0]*n for _ in range(n)]
walls = [(2,8),(0,9),(1,7),(2,2),(2,3),(2,4),(2,5),
         (2,6),(2,7),(3,2),(4,2),(7,2),(6,2),(8,2),
         (4,7),(5,1),(5,2),(5,3),(5,4),(5,5),(5,6),
         (5,7),(6,2),(6,6),(7,6),(3,5),(7,4),(8,4),
         (9,4),(8,6),(8,7),(8,8),(5,0)]
for (x,y) in walls:
    maze[x][y]=1

start = (8,1)
goal = (3,4)

# Visualize maze in console as text
def print_maze(maze, start=None, goal=None, path=None, dist=None):
    n = len(maze)
    print(*"#"*(n+2), sep="")
    for y in range(n-1,-1,-1):
        row = "#"
        for x in range(n):
            if maze[x][y] == 1:
                row += "#"  # wall
            elif start and (x, y) == start:
                row += "S"  # start
            elif goal and (x, y) == goal:
                row += "G"  # goal
            elif path and (x, y) in path:
                row += "*"  # path
            elif dist and dist[x][y] < float('inf'):
                row += str(int(dist[x][y])) if dist[x][y] < 10 else "+"
            else:
                row += " "
        row += "#"
        print(row)
    print(*"#"*(n+2), sep="")
```