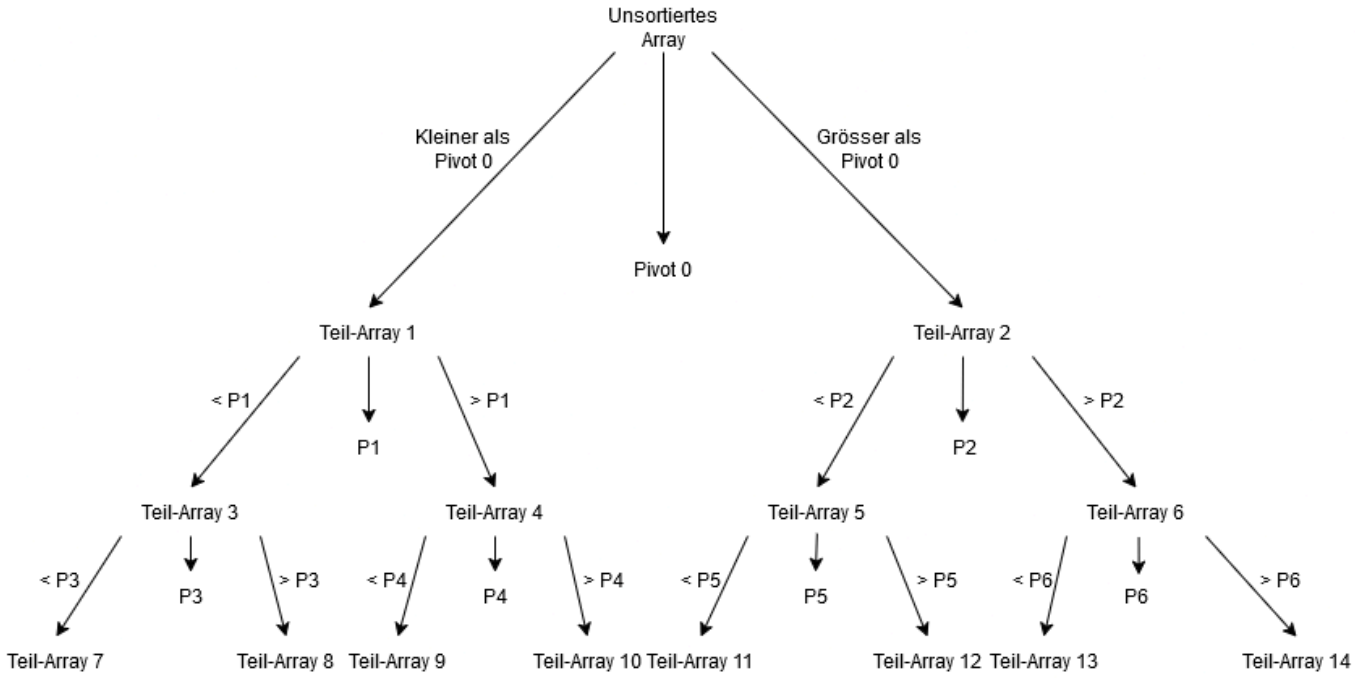
Quicksort

Quicksort (englisch quick ‚schnell' und to sort ‚sortieren') ist ein schneller, rekursiver, nicht-stabiler Sortieralgorithmus, der nach dem Prinzip Teile und herrsche arbeitet. Er wurde ca. 1960 von C. Antony R. Hoare in seiner Grundform entwickelt und seitdem von vielen Forschern verbessert. Der Algorithmus hat den Vorteil, dass er über eine sehr kurze innere Schleife verfügt, was die Ausführungsgeschwindigkeit stark erhöht.

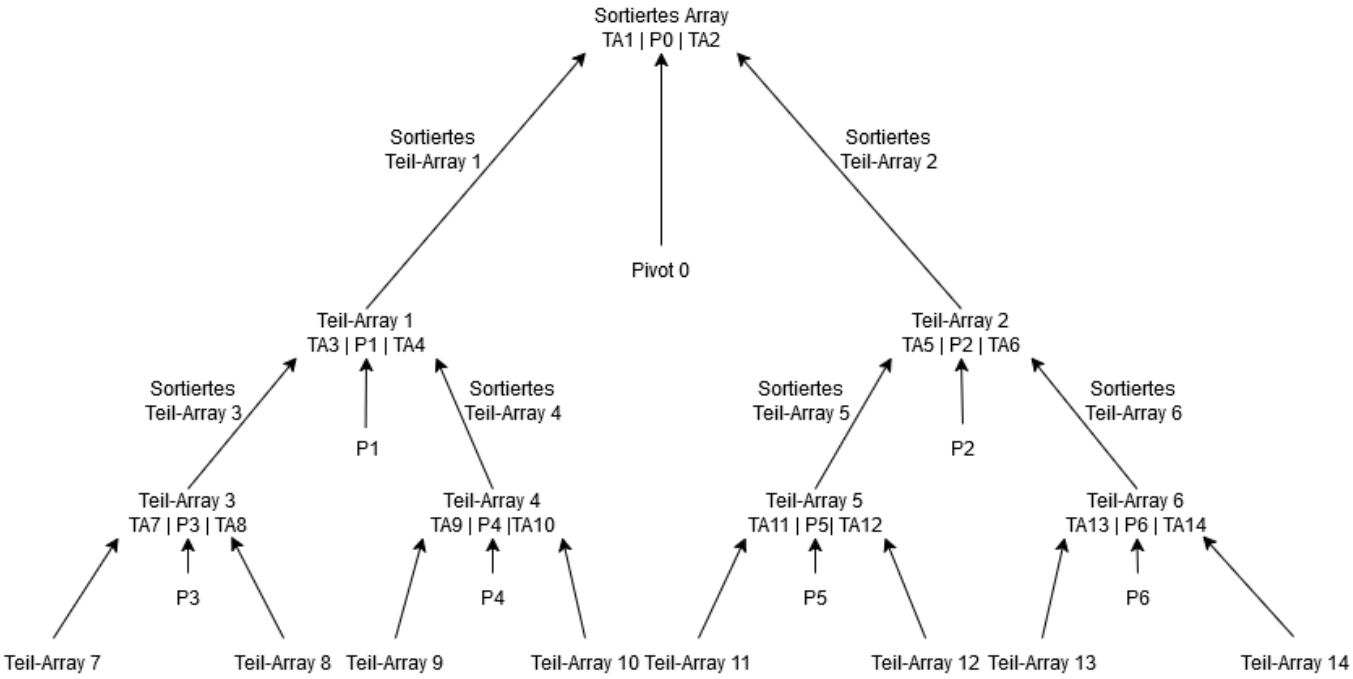
- *Allgemeine Informationen (Ursprung und Beschrieb) Quicksort Algorithmus*
  - *Wikipedia Site zu Quicksort*
  - *https://de.wikipedia.org/wiki/Quicksort*

Funktionsprinzip

Zunächst wird das zu sortierende Array in zwei Teil-Arrays („linkes" und „rechtes" Teil-Array) getrennt. Dazu wählt Quicksort ein sogenanntes Pivotelement aus dem Array aus. Alle Elemente, die kleiner als das Pivotelement sind, kommen in das linke Teil-Array, und alle, die größer sind, in das rechte Teil-Array. Für beide dieser Teil-Arrays wird wiederum jeweils ein Pivotelement definiert und das Teil-Array anhand diesem erneut sortiert. Dies wird solange wiederholt, bis die einzelnen Teil-Arrays eine Länge von 1 oder 0 aufweisen. Diese Selbstaufrufe werden als Rekursion bezeichnet.



Im Anschluss werden die erstellten Teil-Arrays wieder zusammengesetzt. Auf Grund deren Sortierung und der Pivotelemente (Array wird vor oder nach Pivotelement eingefügt, jenachdem ob bei Teilen Grösser oder kleiner als Pivotelement) erhält man nach Abschluss der Zusammenführung ein perfekt sortiertes Array.



- *Funktionsprinzip Quicksort Algorithmus / Quick Sort Sortierverfahren mit Beispiel (deutsch)*
  - *Erklärvideo, wie der Quicksort Algorithmus funktioniert.*
  - *https://www.youtube.com/watch?v=eNUM23f6g-s*

Anwendungsfälle

1. Sortierung von Daten Allgemeine Datensortierung: Quicksort wird oft in vielen Programmiersprachen und Bibliotheken verwendet, um Arrays oder Listen zu sortieren.

2. Datenbank-Operationen Indizierung und Abfragen: In Datenbanken wird Quicksort häufig für das Sortieren von Datensätzen verwendet, um schnelle Abfragen und Indizierungen zu ermöglichen.

3. Suchalgorithmen Schnelle Suche durch sortierte Daten: Quicksort wird oft verwendet, um Daten zu sortieren, die dann mit binärer Suche durchsucht werden können. Einmal sortierte Daten ermöglichen schnellere Suchoperationen (O(log n) statt O(n) in unsortierten Listen).

4. Datenverarbeitung in parallelen und verteilten Systemen Verteilte Systeme: In Systemen, die Daten über mehrere Maschinen verteilen, kann Quicksort verwendet werden, um parallel Daten auf verschiedenen Knoten zu sortieren und zu kombinieren.

- *Anwendungsfälle von Quicksort*
  - *Artikel in dem Funktion und Anwendungsfälle von Quicksort beschrieben werden.*
  - *https://datascientest.com/de/quicksort-beschreibung-und-bedeutung-dieses-sortieralgorithmus*
- *Anwendungsfälle von Quicksort*
  - *Vorgeschlagene Anwendungsfälle durch ChatGPT (Einabe: Hi ChatGPT. Kannst du mir Anwendungsfälle für Quicksor nennen?).*
  - *ChatGPT*

Vor- und Nachteile

Vorteile von Quicksort:

1. Effizienz (im Durchschnitt):
   - Durchschnittliche Zeitkomplexität O(n log n): Quicksort hat eine ausgezeichnete durchschnittliche Laufzeit, was ihn zu einem der effizientesten Sortieralgorithmen für allgemeine Zwecke macht.

2. Gute Performance bei großen Datensätzen:
   - Quicksort funktioniert besonders gut bei großen Datensätzen, weil die rekursive Teilung der Liste und das effiziente Vergleichen von Elementen dazu führen, dass der Algorithmus schnell konvergiert.

3. Einfache Implementierung:
   - Quicksort ist relativ einfach zu implementieren und benötigt wenig Code. In vielen Programmiersprachen wird er direkt oder als Grundlage für die eingebaute Sortierfunktion verwendet.

4. Flexibilität:
   - Der Pivot-Selektionsmechanismus kann angepasst werden, um verschiedene Varianten von Quicksort zu erzeugen, die auf unterschiedliche Datensätze besser abgestimmt sind. Zum Beispiel kann man den Pivot zufällig auswählen oder als Median der drei Elemente wählen, um die Effizienz zu verbessern.

5. Parallelisierbarkeit:
   - Quicksort lässt sich gut parallelisieren. In verteilten Systemen oder bei Multi-Core-Prozessoren kann jeder rekursive Aufruf auf verschiedenen Kernen ausgeführt werden, was zu einer weiteren Leistungssteigerung führt.

Nachteile von Quicksort:

1. Schlechte Worst-Case-Leistung ($O(n^2)$):
   - Im schlechtesten Fall hat Quicksort eine Zeitkomplexität von $O(n^2)$, was auftritt, wenn das Pivot immer das größte oder kleinste Element ist (z. B. bei bereits sortierten Daten oder umgekehrt sortierten Daten). Dies führt zu einer schlechten Performance, da die Liste nicht effizient geteilt wird.

2. Rekursive Natur (Stack Overflows):
   - Quicksort ist ein rekursiver Algorithmus, und bei sehr großen Eingabedaten oder schlecht gewählten Pivot-Elementen kann es zu Stack Overflow-Fehlern kommen, wenn die Rekursion zu tief wird.

3. Nicht stabil:
   - Quicksort ist nicht stabil, was bedeutet, dass gleichwertige Elemente nach der Sortierung ihre ursprüngliche Reihenfolge nicht beibehalten. In Szenarien, in denen die Reihenfolge gleichwertiger Elemente wichtig ist (z. B. bei der Sortierung von Objekten mit mehreren Kriterien), ist Quicksort weniger geeignet.

4. Unregelmäßige Performance bei schlecht strukturierten Daten:
   - Wenn die Daten sehr unregelmäßig oder bereits unsortiert sind, kann Quicksort durch die schlechte Wahl des Pivots ineffizient werden und im schlimmsten Fall eine $O(n^2)$-Leistung erreichen. Dies kann durch den Einsatz spezieller Techniken zur Wahl des Pivots gemildert werden, aber es bleibt ein potenzielles Problem.

- *Vor- und Nachteile von Quicksort*

- *Artikel in dem Funktion, sowie Vor- und Nachteile von Quicksort beschrieben werden.*
- *https://www.studysmarter.de/schule/informatik/algorithmen-und-datenstrukturen/quicksort/*

Testing

Für unseren Algorithmus haben wir folgende Testfälle identifiziert:

1. Leeres Array
   - Ein leeres Array ist per Definition bereits sortiert.
   - Bsp:
     - Input: []
     - Erwarteter Output: []

2. Array mit einem Element
   - Ein Array mit nur einem Element ist bereits sortiert.
   - Bsp:
     - Input: [42]
     - Erwarteter Output: [42]

3. Bereits sortiertes Array
   - Ein bereits sortiertes Array muss nicht nochmals sortiert werden
   - Bsp:
     - Input: [1, 2, 3, 4, 5]
     - Erwarteter Output: [1, 2, 3, 4, 5]

4. Rückwärts sortiertes Array
   - Ein Array, das in absteigender Reihenfolge vorliegt, sollte korrekt aufsteigend sortiert werden.
   - Bsp:
     - Input: [5, 4, 3, 2, 1]
     - Erwarteter Output: [1, 2, 3, 4, 5]

5. Array mit doppelten Elementen
   - Doppelte Elemente sollten in der endgültigen sortierten Liste korrekt auftauchen.
     - Input: [3, 1, 2, 3, 4, 3]
     - Erwarteter Output: [1, 2, 3, 3, 3, 4]

6. Array mit negativen Zahlen
   - Der Algorithmus sollte auch mit negativen Zahlen umgehen können und sie korrekt sortieren.
     - Input: [0, -1, 3, -5, 2]
     - Erwarteter Output: [-5, -1, 0, 2, 3]

7. Array mit allen gleichen Elementen
   - Ein Array, in dem alle Elemente gleich sind, sollte unverändert bleiben, da es bereits sortiert ist.
     - Input: [7, 7, 7, 7, 7]
     - Erwarteter Output: [7, 7, 7, 7, 7]

8. Array mit gemischten positiven und negativen Zahlen
   - Ein Array, das sowohl positive als auch negative Zahlen enthält, sollte korrekt sortiert werden.

        - Input: [3, -2, 5, -1, 0, 4, -3]
        - Erwarteter Output: [-3, -2, -1, 0, 3, 4, 5]

9. Array mit Zahlen in zufälliger Reihenfolge
    - Eine zufällig gemischte Liste sollte korrekt sortiert werden.
        - Input: [8, 3, 1, 7, 0, 10, 2]
        - Erwarteter Output: [0, 1, 2, 3, 7, 8, 10]

10. Array mit vielen doppelten Elementen
    - Der Algorithmus sollte korrekt mit mehrfach vorkommenden Elementen umgehen.
        - Input: [4, 5, 4, 4, 6, 4, 5, 5]
        - Erwarteter Output: [4, 4, 4, 4, 5, 5, 5, 6]

11. Array mit Floats
    - Der Quicksort-Algorithmus sollte auch mit Gleitkommazahlen umgehen können.
        - Input: [2.1, 3.5, 1.2, 5.7, 0.5]
        - Erwarteter Output: [0.5, 1.2, 2.1, 3.5, 5.7]

12. Eingabe in Array muss Integer oder Float sein - Bei der Eingabe von anderen Elementen als Zahlen soll eine Fehlermeldung geworfen werden und der User aufgefordert, Zahlen einzugeben
    - Input: [D, $, &, 1, 7]
    - Fehlermeldung: "Als Eingabe dürfen nur Zahlen verwendet werden!"

In dieser Arbeit wurden die Testfälle mit folgendem Code abgehandelt:

```
import unittest
from quicksort import quicksort

class TestSort(unittest.TestCase):

    def testEmpty(self):
        test_input = []
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [])
        self.assertEqual(error, None)

    def testOneElement(self):
        test_input = [42]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [42])
        self.assertEqual(error, None)

    def testRandom(self):
        test_input = [8, 3, 1, 7, 0, 10, 2]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [0, 1, 2, 3, 7, 8, 10])
        self.assertEqual(error, None)

    def testSorted(self):
        test_input = [1, 2, 3, 4, 5]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [1, 2, 3, 4, 5])
        self.assertEqual(error, None)
```

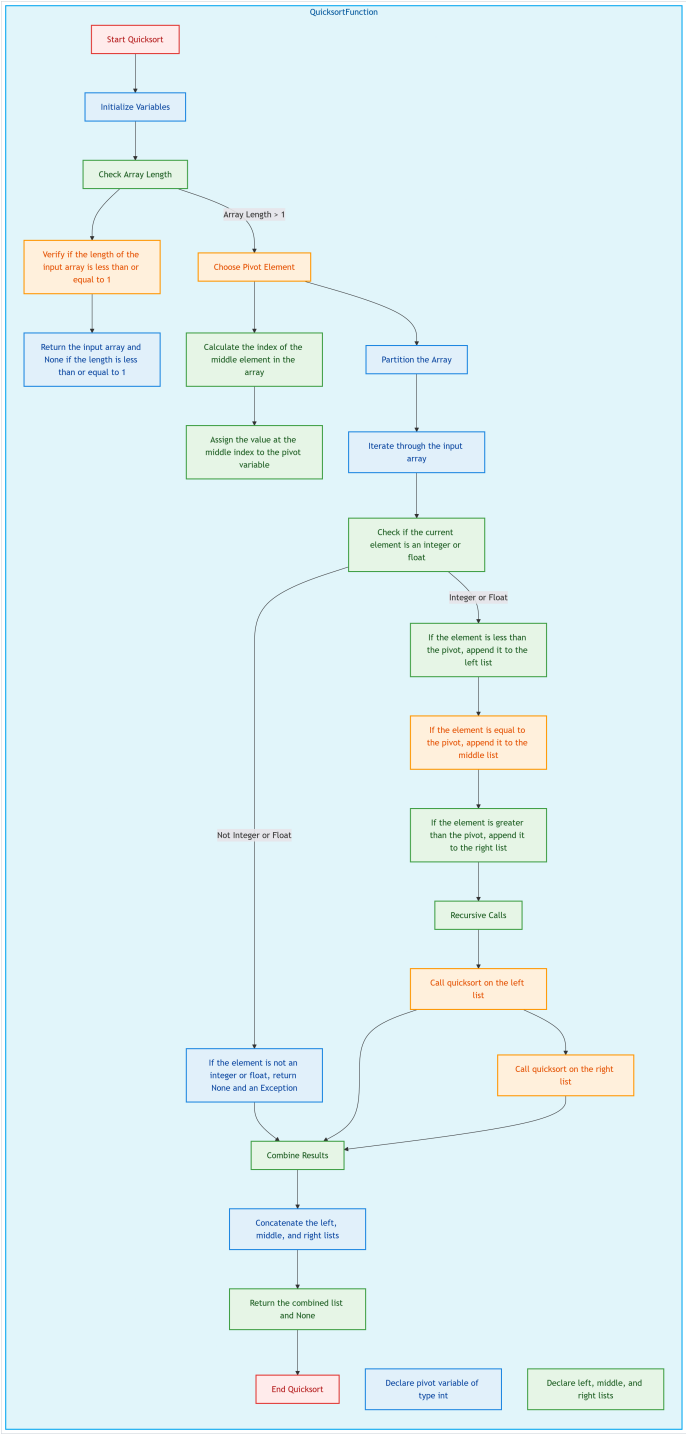
```
    def testReverseSorted(self):
        test_input = [5, 4, 3, 2, 1]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [1, 2, 3, 4, 5])
        self.assertEqual(error, None)

    def testDubbledElements(self):
        test_input = [3, 1, 2, 3, 4, 3]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [1, 2, 3, 3, 3, 4])
        self.assertEqual(error, None)

    def testNegativeElements(self):
        test_input = [0, -1, 3, -5, 2]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [-5, -1, 0, 2, 3])
        self.assertEqual(error, None)

    def testAllTheSame(self):
        test_input = [7, 7, 7, 7]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [7, 7, 7, 7])
        self.assertEqual(error, None)

    def testManyDoubled(self):
        test_input = [4, 5, 4, 4, 6, 4, 5, 5]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [4, 4, 4, 4, 5, 5, 5, 6])
        self.assertEqual(error, None)

    def testFloats(self):
        test_input = [2.1, 3.5, 1.2, 5.7, 0.5]
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, [0.5, 1.2, 2.1, 3.5, 5.7])
        self.assertEqual(error, None)

    def testMixedTypes(self):
        test_input = ['D', '$', '&', 1, 7]
        test_output, error = quicksort(test_input)
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, None)
        self.assertEqual(error, Exception("Als Eingabe dürfen nur Zahlen verwendet werden!"))

    def testListOfStrings(self):
        test_input = ['D', '$', '&', 'f', '5']
        test_output, error = quicksort(test_input)
        self.assertEqual(test_output, None)
        self.assertEqual(error, Exception("Als Eingabe dürfen nur Zahlen verwendet werden!"))
```

- *Eigene Testfälle*
- *Vorschlagene Testfälle für Quicksort*
  - *Ergänzen unserer Testfälle mit Vorschlägen aus ChatGPT (Hi ChatGPT. Kannst du mir mögliche Testfällefür Quicksort nennen?)*
  - *ChatGPT*

Python Implementierung

Anhand der, durch die Recherche gewonnenen, Erkenntnisse zum Funktionsprinzip des Quicksort Algorithmus, entschieden wir uns diesen wie folgt in unserem Python -Code abzubilden:

1. Definition Funktion Quicksort
    - Array als Eingabe
    - Definition Variable Pivot, die später Pivot-Wert speichert
    - Initialisierung Liste left, Liste middle und Liste right
2. Definition Basisfall
    - Wenn Liste leer oder nur ein ELement enthält ist sie bereits sortiert
    - Das Array wird ohne Fehlermeldung oder Sortierung wieder ausgegeben
3. Auswahl Pivot-Element
    - Pivot-Element wird als das mittlere Element (Median) des Arrays ausgewählt.
    - len(array) // 2 berechnet dabei den Index des mittleren Elements.
4. Partitionierung der Liste
    - Der Code durchläuft jedes Element im Array, überprüft ob die enthaltenen Elemente int oder float sind
        - Wenn ein Element nicht vom Typ int oder float ist, wird die Meldung "Als Eingabe dürfen nur Zahlen verwendet werden!" ausgegeben
    - Elemente werden gemäss Pivot-Element einsortiert
        - Wenn das Element kleiner als das Pivot ist, wird es zur Array left hinzugefügt.
        - Wenn es gleich dem Pivot ist, wird es zur Array middle hinzugefügt.
        - Wenn es größer ist, geht es in die Array right.
5. Rekursiver Aufruf von Quicksort
    - Nachdem das Array in left, middle und right aufgeteilt wurde, wird die Funktion rekursiv auf left und right angewendet.
6. Ergebnis zusammenführen
    - Nachdem die rekursiven Aufrufe abgeschlossen sind, werden die sortierten Arrays für left, middle und right wieder zusammengefügt.
    - Das Ergebnis ist ein perfekt sortiertes Array

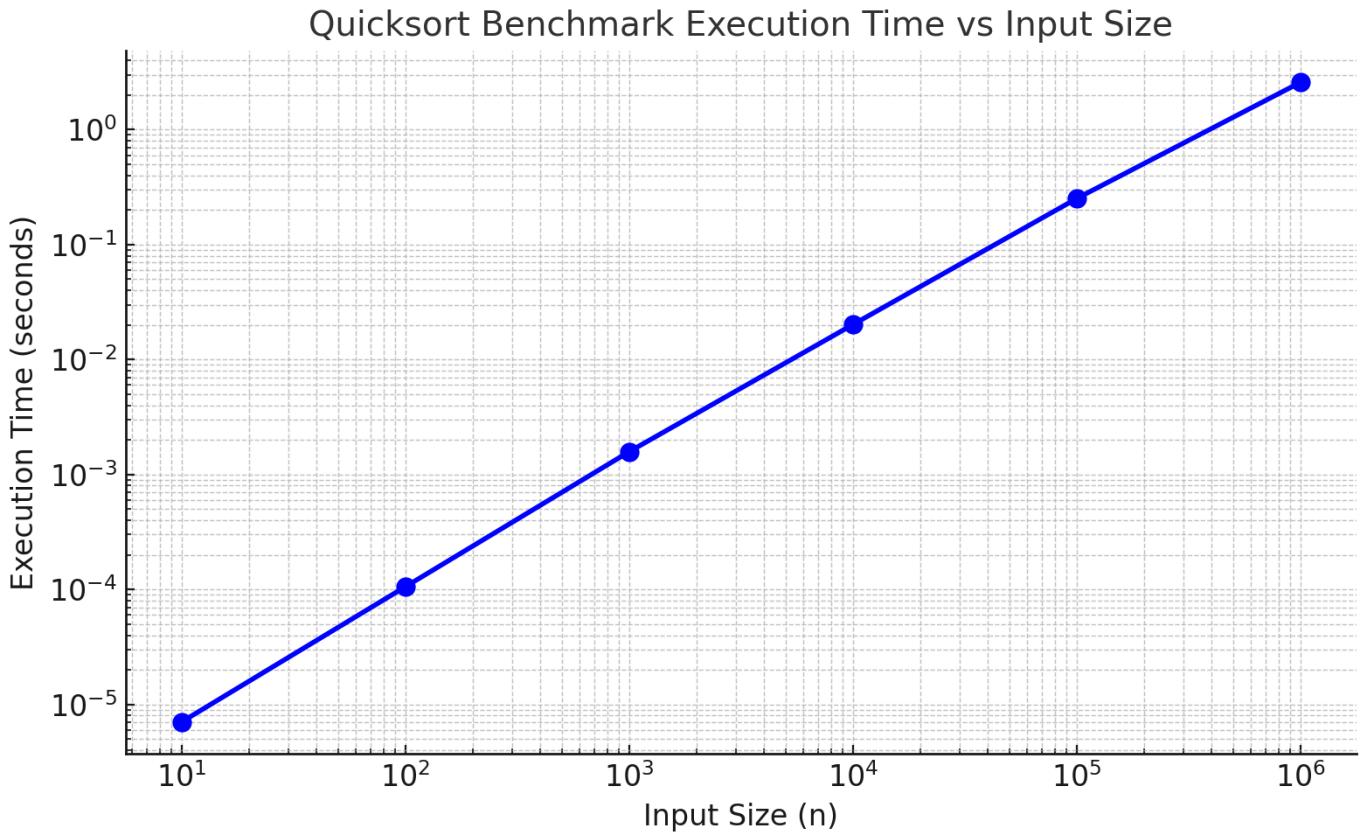



```python
def quicksort(array: list):
    pivot: int
    left: list = []
    middle: list = []
    right: list = []

    if len(array) <= 1:
        return array, None

    pivot = array[(len(array) // 2 )]

    for x in array:
        if isinstance(x, (int, float)):
            if x < pivot:
```

```
                left.append(x)
            elif x == pivot:
                middle.append(x)
            else:
                right.append(x)
        else:
            return None, Exception("Als Eingabe dürfen nur Zahlen verwendet werden!")

    left, leftErr = quicksort(left)
    right, rightErr = quicksort(right)
    return (left +  middle + right), None
```

Komplexitätsanalyse

Die durchgeführte Komplexitätsanalyse ergab einen Wert von O(n), gemäss BigO-Notation. Dieser Wert erschien den Projektbeteiligten sureal. Dies da die Recherche zu Quicksort ergab, dass im Optimalfall ein Wert von O(n log n) erreicht wird. Trotz Prüfung der Komplexitätsanalyse und des verwendeten Codes, fanden die Projektbeteilgten keinen Fehler, möchten aber darauf hinweisen, dass das erhaltene Ergebniss in Sachen Komplexitätsanalyse möglicherweise fehlerhaft ist.



Fazit