

Binary Insertion Sort (Binary-Sort)


Grundidee

Binary Insertion Sort ist eine **Optimierung von Insertionsort**. Statt die Einfügeposition linear zu suchen, wird eine **binäre Suche** verwendet.

Binary-Sort reduziert die Anzahl der Vergleiche, behält aber die gleiche Anzahl an Verschiebungen bei!

- Verschiebungen bleiben gleich
 - Weniger Vergleiche
 - Gleiche asymptotische Laufzeit
-

Voraussetzungen

-  keine (intern wird ein sortierter Teil genutzt)
 - funktioniert auf **beliebigen vergleichbaren Elementen**
-

Laufzeiten & Eigenschaften

Eigenschaft	Wert
Best Case	$O(n \log n)$
Average Case	$O(n^2)$
Worst Case	$O(n^2)$
Speicherbedarf	$O(1)$
In-place	ja
Stabil	ja

Hinweis: Die Anzahl der **Vergleiche** sinkt auf $O(n \log n)$, die Anzahl der **Verschiebungen** bleibt $O(n^2)$.

Schritt-für-Schritt-Beispiel

Ausgangsliste:

[5, 3, 4, 1]

Schritt 1

- sortierter Teil: [5]
- füge 3 per binärer Suche ein → [3, 5]

Schritt 2

- sortierter Teil: [3, 5]
- füge 4 ein → [3, 4, 5]

Schritt 3

- sortierter Teil: [3, 4, 5]
- füge 1 ein → [1, 3, 4, 5]

Ergebnis:

[1, 3, 4, 5]

5 Besonderheiten / Prüfungsrelevante Hinweise

- Reduziert Vergleiche, nicht Verschiebungen
 - Gut bei teuren Vergleichsoperationen
 - Oft Prüfungsfrage: *Warum bleibt $O(n^2)$?*
-

6 Vor- und Nachteile

Vorteile

- stabil
- weniger Vergleiche als Insertionsort
- in-place

Nachteile

- weiterhin quadratische Laufzeit
 - kein Vorteil bei vielen Verschiebungen
-

Merksatz für die Prüfung

7 Python-Implementierung

```
In [1]: def binary_search(arr, key, start, end):
        while start < end:
            mid = (start + end) // 2
            if arr[mid] < key:
                start = mid + 1
            else:
                end = mid
        return start

def binary_insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        pos = binary_search(arr, key, 0, i)

        j = i
        while j > pos:
            arr[j] = arr[j - 1]
            j -= 1

        arr[pos] = key

    return arr
```

BST (Binary Search Tree) Sort

Eine weitere Variante, die binäre Suche nutzt, ist der **BST-Sort**. Hierbei werden die Elemente in einen binären Suchbaum eingefügt und anschließend in-order traversiert, um die sortierte Reihenfolge zu erhalten.

```
In [2]: class Node:
        def __init__(self, value):
            self.value = value
            self.left = None
            self.right = None

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, value):
        if self.root is None:
            self.root = Node(value)
        else:
            self._insert_recursive(self.root, value)
```

```

def _insert_recursive(self, current, value):
    if value < current.value:
        if current.left is None:
            current.left = Node(value)
        else:
            self._insert_recursive(current.left, value)
    else:
        if current.right is None:
            current.right = Node(value)
        else:
            self._insert_recursive(current.right, value)

def inorder(self):
    result = []
    self._inorder_recursive(self.root, result)
    return result

def _inorder_recursive(self, node, result):
    if node is not None:
        self._inorder_recursive(node.left, result)
        result.append(node.value)
        self._inorder_recursive(node.right, result)

def binary_tree_sort(arr):
    bst = BinarySearchTree()

    for value in arr:
        bst.insert(value)

    return bst.inorder()

data = [5, 3, 4, 1]
sorted_data = binary_tree_sort(data)
print(sorted_data)

```

[1, 3, 4, 5]



Schrittweise Erklärung (für die Prüfung)

Einfügen der Elemente:

```

    5
   /
  3
   \
    4
   /
  1

```

Inorder-Traversierung:

links → wurzel → rechts → 1, 3, 4, 5

Prüfungs-Merksatz

Binary Sort mit einem Suchbaum fügt Elemente per binärer Suche ein und erzeugt durch Inorder-Traversierung eine sortierte Folge.

Abgrenzung zur vorherigen Binary Insertion Sort

Variante Idee Laufzeit Speicher Binary Insertion Sort (Array) Binäre Suche + Verschieben
 $O(n^2)$ $O(1)$ Binary Sort (BST / Tree Sort) Binäre Suche im Baum \emptyset $O(n \log n)$ $O(n)$

 Sehr typische Prüfungsfrage!