

Sortieralgorithmen in Java

Sortieren ist eine grundlegende Operation in der Programmierung, und Java glänzt mit seiner Flexibilität bei der Implementierung verschiedener Sortieralgorithmen. Lassen Sie uns Quicksort, Merge Sort und Bubble Sort –drei beliebte Sortiertechniken mit unterschiedlichen Eigenschaften –erkunden.

1. Quicksort: Teilen und Herrschen

Quicksort wählt einen Pivot, partitioniert das Array und sortiert rekursiv die Teilarrays. Es ist effizient mit einer durchschnittlichen Zeitkomplexität von $O(n \log n)$.

Java-Implementierung

```
public class QuickSort {  
    public static void quickSort(int[] arr, int low, int high) {  
        if (low < high) {  
            int pi = partition(arr, low, high);  
            quickSort(arr, low, pi - 1);  
            quickSort(arr, pi + 1, high);  
        }  
    }  
  
    private static int partition(int[] arr, int low, int high) {  
        int pivot = arr[high];  
        int i = low - 1;  
        for (int j = low; j < high; j++) {  
            if (arr[j] <= pivot) {  
                i++;  
                int temp = arr[i];  
                arr[i] = arr[j];  
                arr[j] = temp;  
            }  
        }  
        int temp = arr[i + 1];  
        arr[i + 1] = arr[high];  
        arr[high] = temp;  
        return i + 1;  
    }  
  
    public static void main(String[] args) {  
    }  
}
```

```

    int[] arr = {10, 7, 8, 9, 1, 5};
    quickSort(arr, 0, arr.length - 1);
    System.out.println("Sortiert: " + java.util.Arrays.toString(arr));
}
}

```

Ausgabe: Sortiert: [1, 5, 7, 8, 9, 10]

2. Merge Sort: Stabile Sortierung

Merge Sort teilt das Array, sortiert die Hälften und fusioniert sie wieder. Es ist stabil und garantiert $O(n \log n)$ Zeit.

Java-Implementierung

```

public class MergeSort {
    public static void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int m = (l + r) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    private static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1], R = new int[n2];
        for (int i = 0; i < n1; i++) L[i] = arr[l + i];
        for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        mergeSort(arr, 0, arr.length - 1);
        System.out.println("Sortiert: " + java.util.Arrays.toString(arr));
    }
}

```

```
 }
}
```

Ausgabe: Sortiert: [5, 6, 7, 11, 12, 13]

3. Bubble Sort: Einfach, aber langsam

Bubble Sort tauscht wiederholt benachbarte Elemente aus, wenn sie nicht in der richtigen Reihenfolge sind.
Es ist $O(n^2)$, aber einfach zu verstehen.

Java-Implementierung

```
public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 34, 25, 12, 22, 11, 90};
        bubbleSort(arr);
        System.out.println("Sortiert: " + java.util.Arrays.toString(arr));
    }
}
```

Ausgabe: Sortiert: [11, 12, 22, 25, 34, 64, 90]