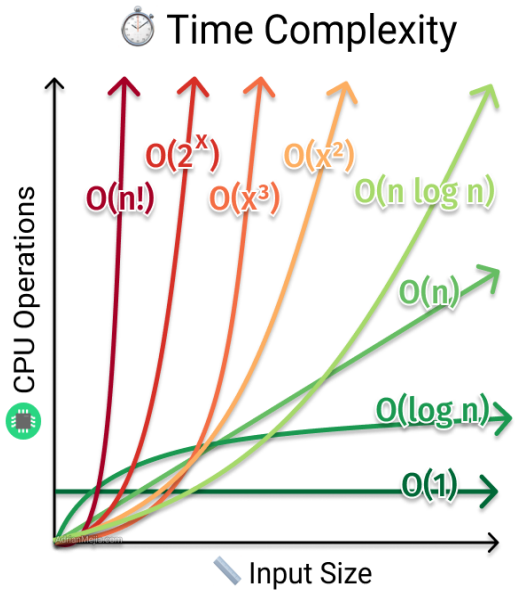
> 👉 Top Secret 👈

I) *Laufzeitkomplexität*

- *beschreibt, wie lange ein eine SQL-Abfrage im Verhältnis zur Größe der Eingabedaten läuft*
- *Je schneller die Laufzeit im Verhältnis zur Eingabedatenmenge wächst, desto komplexer ist der Algorithmus*



I ***a***) *Warum ist Laufzeitkomplexität wichtig?*

- **Performance**:

  > Eine Abfrage mit hoher Laufzeitkomplexität kann bei großen Datenmengen sehr lange dauern oder zu einem Absturz führen

- **Skalierbarkeit**:

  > Wenn die Datenmenge wächst, sollten Abfrage idealerweise nicht proportional langsamer werden

- **Ressourcenverbrauch**:

  > Langsame Abfragen verbrauchen mehr Serverressourcen (CPU, Speicher) und können andere Benutzer beeinträchtigen

II) *Beispiele* von Laufzeitkomplexitäten

- $\mathcal{O}_{(1)}$:

KONSTANTE ZEIT/KOMPLEXITÄT : **DIE LAUFZEIT HÄNGT NICHT VON DER DATENMENGE AB**

- ***Beispiel 1***:

  Eine Abfrage, die nur die erste Zeile einer Tabelle ausgibt

  ```
  SELECT * FROM customers WHERE customer_id = 123;
  ```

- ***Beispiel 2***:

  Zählen aller Zeilen in einer Tabelle mit einem Index auf der gesamten Tabelle

  ```
  SELECT COUNT(*) FROM customers;
  ```

- $\mathcal{O}_{(n)}$:

LINEARE KOMPLEXITÄT : **DIE LAUFZEIT IST PROPORTIONAL ZUR DATENMENGE**

- ***Beispiel 1***:

  Eine Abfrage, die jede Zeile einer Tabelle durchläuft

  ```
  SELECT * FROM orders;
  ```

- ***Beispiel 2***:

  Berechnung einer Summe über alle Werte einer Spalte

  ```
  SELECT SUM(amount) FROM orders;
  ```

- $\mathcal{O}_{(n^2)}$:

QUADRATISCHE KOMPLEXITÄT : **DIE LAUFZEIT WÄCHST QUADRATISCH MIT DER DATENMENGE**

- ***Beispiel 1***:

  Verschachtelte Schleifen (vereinfacht, in der Praxis oft ineffizient)

```
SELECT * FROM customers c1, customers c2
WHERE c1.city = c2.city;
```

- ***Beispiel 2***:

  Berechnung aller möglichen Kombinationen ohne Indizes

```
SELECT * FROM products, colors;
```

- $\mathcal{O}_{(\log n)}$:

LOGARITHMISCHE KOMPLEXITÄT : **DIE LAUFZEIT WÄCHST LOGARITHMISCH MIT DER DATENMENGE**

- ***Beispiel***:

  Binäre Suche auf einem sortierten Index (vereinfacht)

```
-- Annahme: Ein Index auf customer_name ist vorhanden und
sortiert
SELECT * FROM customers
WHERE customer_name >= 'Mustermann'
ORDER BY customer_name
FETCH FIRST 1 ROW ONLY;
```

- $\mathcal{O}_{(n \log n)}$:

SUPERLINEARE KOMPLEXITÄT : **LIEGT ZWISCHEN $\mathcal{O}(N)$ UND $\mathcal{O}(N^2)$**

- ***Beispiel***:

  Optimierte Sortieralgorithmen wie Quicksort

```
SELECT * FROM customers ORDER BY last_name;
```

- $\mathcal{O}_{(2^n)}$:

EXPONENTIELLE KOMPLEXITÄT : **DIE LAUFZEIT VERDOPPELT SICH, WENN DIE DATENMENGE UM EINE EINHEIT GRÖSSER WIRD**

- *Beispiel*:

  > Bilden aller Paare einer Menge, Türme von Hanoi als rekursiver Algorithmus

---

- $\mathcal{O}_{(n!)}$:

  FAKTORIELLE KOMPLEXITÄT : **DIE LAUFZEIT WÄCHST MIT DER FAKULTÄT DER DATENMENGE**

- *Beispiel*:

  > Problem des Handlungsreisenden

---

made by

jxi-Enki