Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

HNSW

Hierarchical Navigable Small World

Von der Brute-Force-Suche zum logarithmischen Index

Karsten Keßler
20.02.2026

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

---

1. Das Problem: Warum brauchen wir HNSW?

Stellen Sie sich vor, Sie haben eine Vektordatenbank mit 1 Million Produktvektoren, jeder mit 384 Dimensionen. Ein Nutzer sucht nach „Schiff“ – dieser Suchbegriff wird ebenfalls in einen 384-dimensionalen Vektor umgewandelt. Nun muss der ähnlichste Produktvektor gefunden werden.

1.1 Brute Force: Der naive Ansatz

Die einfachste Lösung: Jeden einzelnen der 1 Million Vektoren mit dem Query-Vektor vergleichen und den mit der höchsten Kosinus-Ähnlichkeit zurückgeben. Das funktioniert, ist aber viel zu langsam:

Bei **n = 1.000.000** Vektoren benötigt man **1.000.000 Vergleiche**. Die Laufzeitkomplexität ist **O(n)** – linear mit der Datenmenge. Bei doppelt so vielen Daten dauert es doppelt so lang.

1.2 Was bedeutet die O-Notation?

Die O-Notation (auch „Big-O“ oder Landau-Notation) beschreibt, wie sich die Laufzeit eines Algorithmus verhält, wenn die Datenmenge n wächst – nicht die exakte Anzahl der Schritte, sondern die Wachstumsordnung.

Notation	Bedeutung	Beispiel	n = 1 Mio.
O(1)	Konstant	HashMap-Lookup	1 Schritt
O(log n)	Logarithmisch	B-Tree, HNSW	≈20 Schritte
O(n)	Linear	Brute Force	1.000.000 Schritte
$O(n^2)$	Quadratisch	Nested Loop Join	10^{12} Schritte

Kernaussage: O(log n) bedeutet, dass bei einer Verdoppelung der Datenmenge nur **ein einziger Schritt** mehr benötigt wird. Bei 1 Million Vektoren sind das nur ca. 20 Vergleiche statt 1 Million.

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

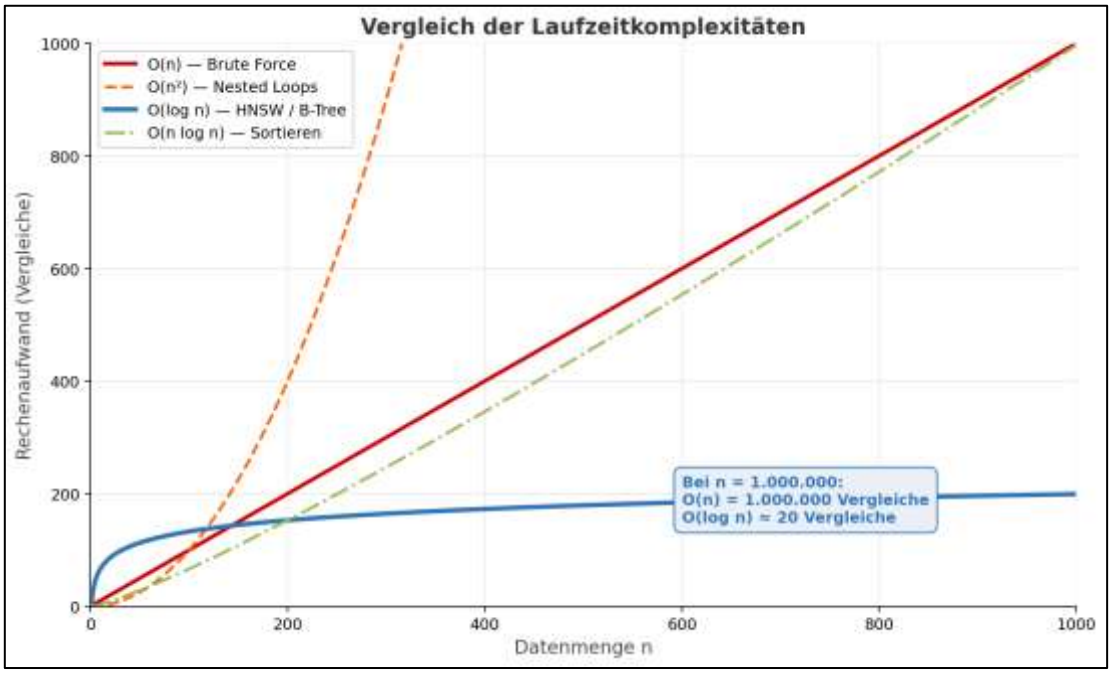


Abbildung 1: Vergleich der Laufzeitkomplexitäten. O(log n) wächst so langsam, dass die Kurve fast flach erscheint.

2. Die Lösung: HNSW in sechs Schritten

HNSW (Hierarchical Navigable Small World) ist ein Indexverfahren, das die Vektorsuche von O(n) auf O(log n) beschleunigt. Das Verfahren lässt sich in sechs aufeinander aufbauende Schritte zerlegen:

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

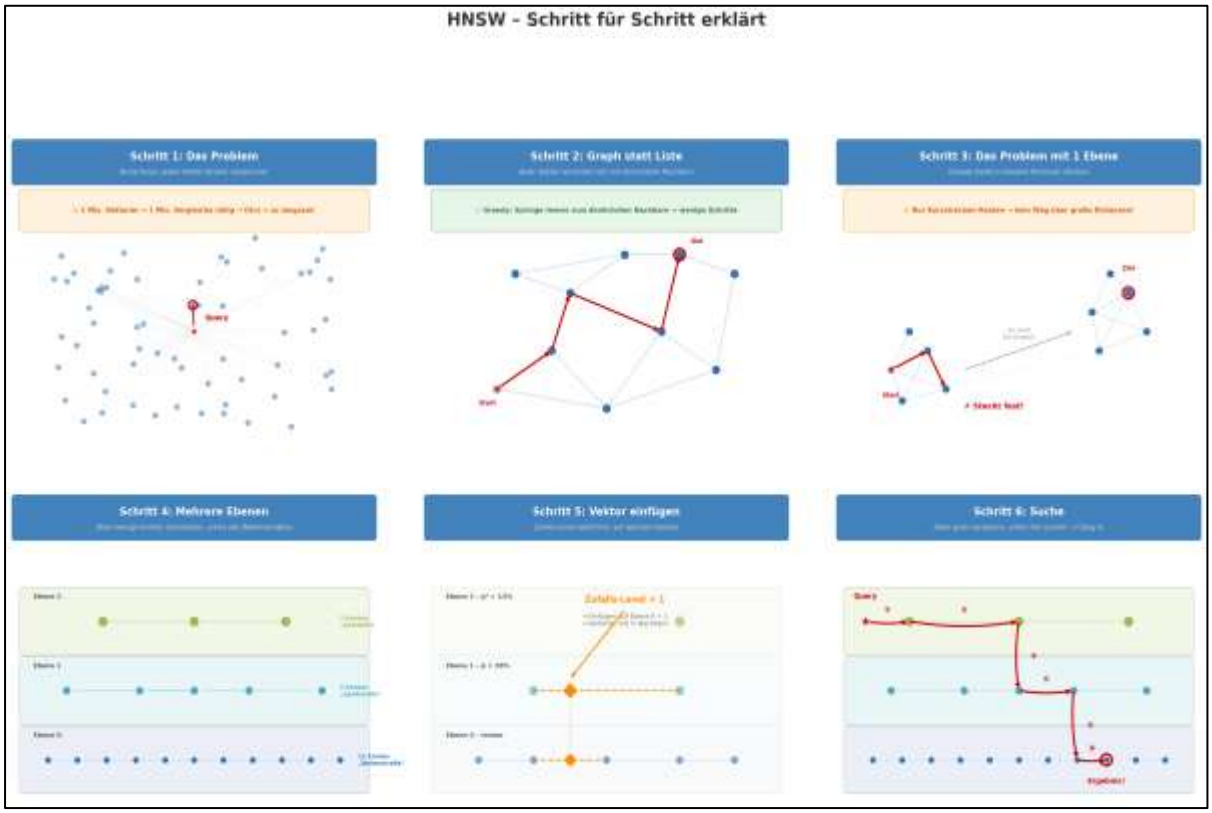


Abbildung 2: HNSW in sechs Schritten – vom Problem zur Lösung.

Schritt 1: Das Problem erkennen

Brute Force vergleicht den Query-Vektor mit jedem gespeicherten Vektor. Bei n Vektoren sind das n Berechnungen der Kosinus-Ähnlichkeit – O(n).

Schritt 2: Graph statt Liste

Statt alle Vektoren linear zu durchsuchen, verbindet HNSW jeden Vektor mit seinen ähnlichsten Nachbarn als Kanten in einem Graphen. Die Suche startet bei einem beliebigen Knoten und springt immer zum Nachbarn, der dem Query-Vektor am ähnlichsten ist (Greedy-Suche). Das reduziert die Anzahl der Vergleiche drastisch.

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

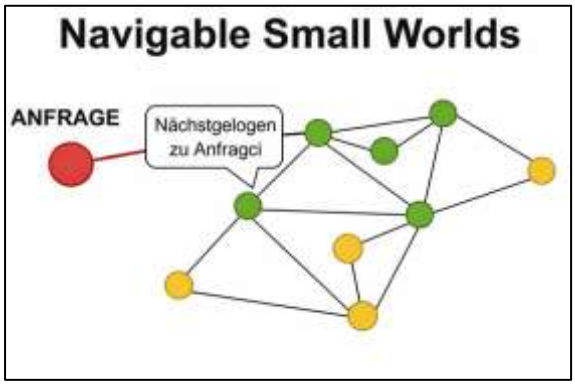


Abbildung 2a: Greedy-Suche.

Schritt 3: Das Problem mit einer Ebene

Ein einfacher Graph hat nur Kurzstrecken-Verbindungen. Wenn das Ziel weit entfernt liegt, bleibt die Greedy-Suche in einem lokalen Minimum stecken – sie findet keinen Nachbarn, der näher am Ziel liegt, obwohl es bessere Treffer im Graphen gibt. Es fehlen „Abkürzungen" über große Distanzen.

Schritt 4: Mehrere Ebenen – die Hierarchie

Hier kommt das „Hierarchical" ins Spiel. HNSW baut mehrere Ebenen übereinander – wie ein Straßennetz:

Ebene 0 ist das vollständige Netz: Alle Vektoren sind enthalten, die Kanten gehen nur zu nahen Nachbarn (Nebenstraßen). **Ebene 1** enthält nur eine Teilmenge. Da es weniger Knoten gibt, überbrücken die Kanten automatisch größere Distanzen (Landstraßen). **Ebene 2+** enthält noch weniger Knoten – die Kanten überbrücken riesige Distanzen (Autobahnen).

Schritt 5: Vektor einfügen – der Zufalls-Level

Beim Einfügen eines neuen Vektors wird per Zufall ein Level gezogen. Die Wahrscheinlichkeit sinkt exponentiell:

$P(\text{Ebene } l) = p^l$ mit $p = 1/\ln(m)$

Jeder Vektor ist **garantiert auf Ebene 0**. Aber nur mit Wahrscheinlichkeit $p \approx 0{,}36$ auch auf Ebene 1, mit $p^2 \approx 0{,}13$ auf Ebene 2, usw. Das ergibt automatisch die Pyramidenstruktur: unten viele Knoten (fein), oben wenige (grob).

Auf jeder Ebene wird der neue Vektor mit seinen m nächsten Nachbarn verbunden. Der Parameter m (typisch: 16) bestimmt die Anzahl der Kanten pro Knoten.

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

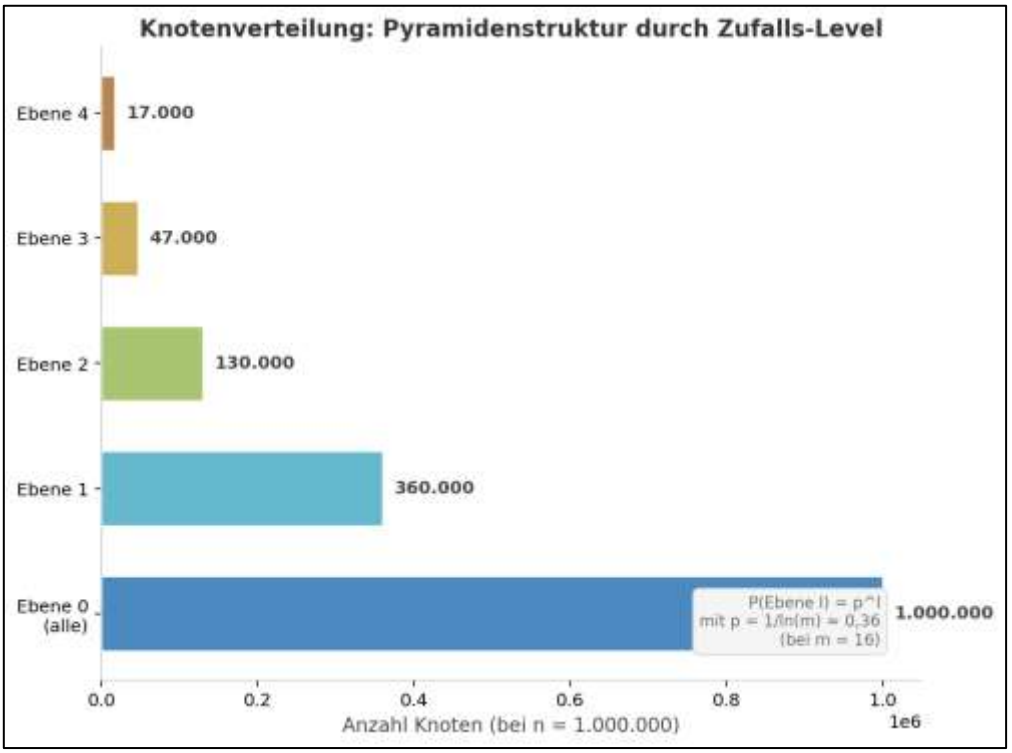


Abbildung 3: Knotenverteilung bei 1 Million Vektoren. Ebene 0 enthält alle, höhere Ebenen exponentiell weniger.

Schritt 6: Suche – der komplette Ablauf

Wenn eine Suchanfrage kommt (z.B. „Schiff“), passiert Folgendes:

1. Der Query-Text wird in einen 384-dimensionalen Vektor umgewandelt (Embedding).
2. Die Suche startet auf der obersten Ebene bei einem zufälligen Einstiegsknoten.
3. Greedy-Navigation: Sprung zum jeweils ähnlichsten Nachbarn auf dieser Ebene (große Sprünge).
4. Wenn kein Nachbar näher ist: Abstieg eine Ebene tiefer beim selben Knoten.
5. Schritte 3–4 wiederholen auf jeder Ebene (immer feinere Sprünge).
6. Auf Ebene 0: Feinsuche mit ef Kandidaten → Top-K Ergebnisse zurückgeben.

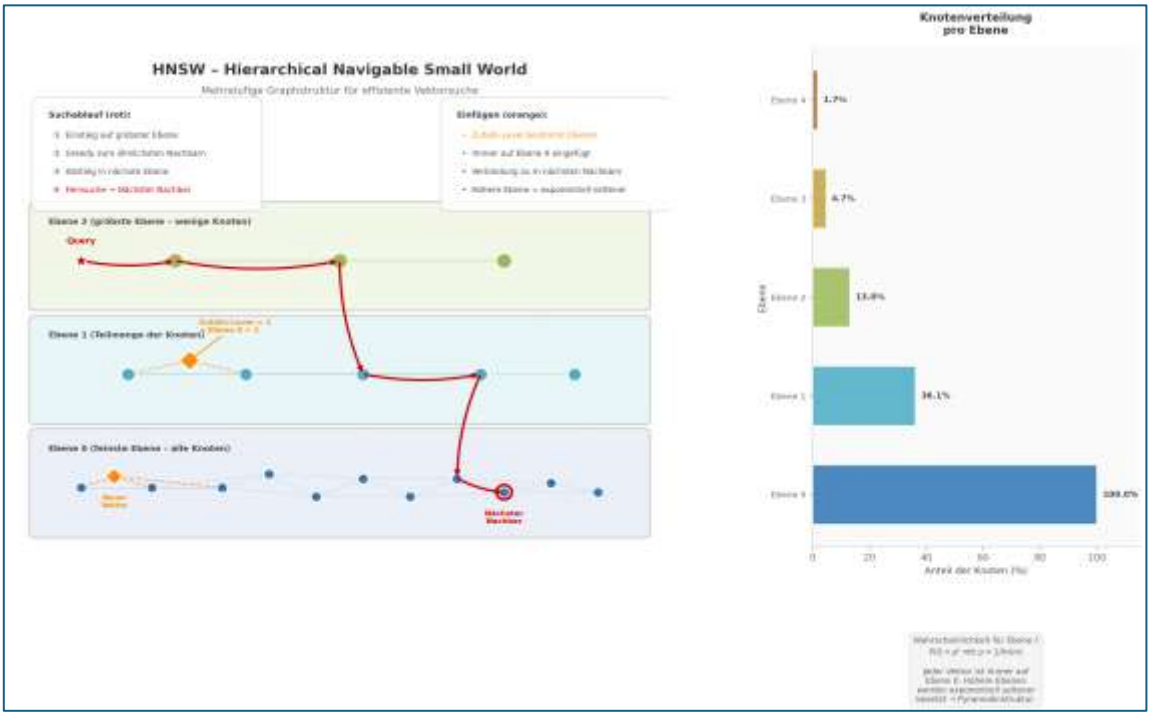
Die Anzahl der Ebenen ist log(n). Auf jeder Ebene werden nur wenige Greedy-Sprünge gemacht. Das ergibt insgesamt: **O(log n) Vergleiche**.

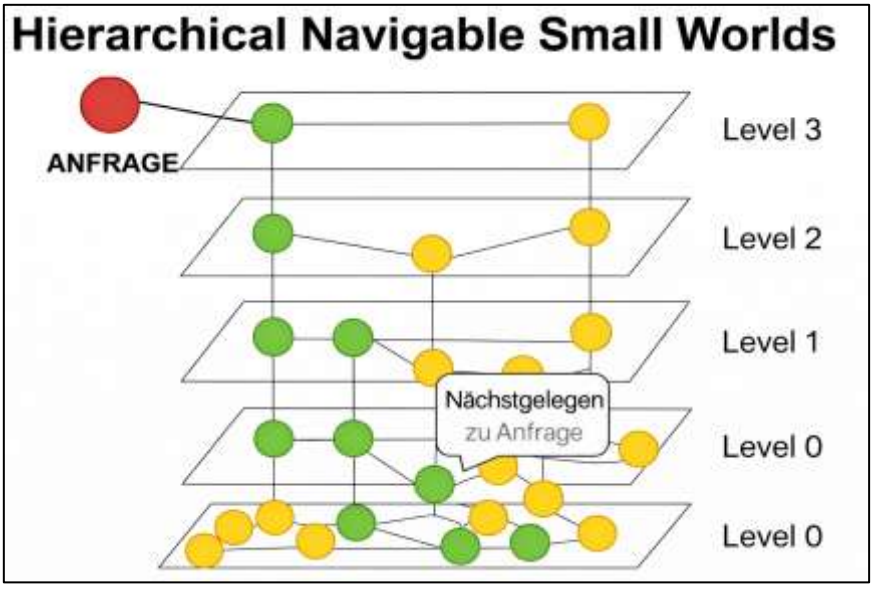
Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

---

3. Gesamtübersicht: HNSW-Architektur

Das folgende Diagramm zeigt die vollständige HNSW-Architektur: Links die dreistufige Graphstruktur mit Suchpfad (rot) und Einfüge-Logik (orange), rechts die Knotenverteilung pro Ebene.





Abbildungen 4a und 4b: HNSW-Gesamtarchitektur mit Suchpfad, Einfüge-Logik und Knotenverteilung.

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

3.1 Konfigurationsparameter

Parameter	Typischer Wert	Bedeutung
m	16	Anzahl der Kanten pro Knoten. Höher = bessere Qualität, mehr Speicher.
ef_construct	128	Suchbreite beim Indexaufbau. Höher = besserer Index, längerer Aufbau.
ef	64	Suchbreite zur Queryzeit. Höher = genauer, aber langsamer.
mL	1/ln(m)	Steuert die Wahrscheinlichkeit höherer Ebenen (Pyramidenform).

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

4. Vergleich: HNSW und B-Tree – zwei Wege zu O(log n)

Man kennt B-Tree-Indizes aus den relationalen Datenbanken. HNSW folgt dem gleichen Grundprinzip – hierarchische Halbierung des Suchraums – aber für einen völlig anderen Datentyp.

4.1 Das gemeinsame Prinzip

Sowohl B-Tree als auch HNSW erreichen O(log n), weil sie den Suchraum auf jeder Ebene halbieren (oder zumindest drastisch verkleinern). Die Anzahl der Ebenen ist proportional zu log(n), und auf jeder Ebene wird nur eine konstante Anzahl von Vergleichen durchgeführt.

Analogie: Eine binäre Suche in einem sortierten Array mit 1 Million Elementen braucht nur $\log_2(1.000.000) \approx 20$ Schritte. B-Tree und HNSW erweitern dieses Prinzip auf komplexere Datenstrukturen.

4.2 Die Unterschiede

Aspekt	B-Tree (MySQL)	HNSW (Qdrant)
Datentyp	Skalare Werte (Zahlen, Strings)	Hochdimensionale Vektoren (384+ Dim.)
Vergleich	Exakt: = , < , >	Ähnlichkeit: cos(A,B)
Struktur	Balancierter Baum	Mehrstufiger Graph
Ergebnis	Exakt: Wert gefunden oder nicht	Approximativ: ähnlichster Nachbar
Ebenen-Aufbau	Deterministisch (Splits)	Zufällig (probabilistisch)
Navigation	Vergleich: links/rechts	Greedy: ähnlichster Nachbar
Komplexität	O(log n) exakt	O(log n) approximativ (ANN)
Einsatz	WHERE price < 100	Semantische Suche: „Schiff“

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

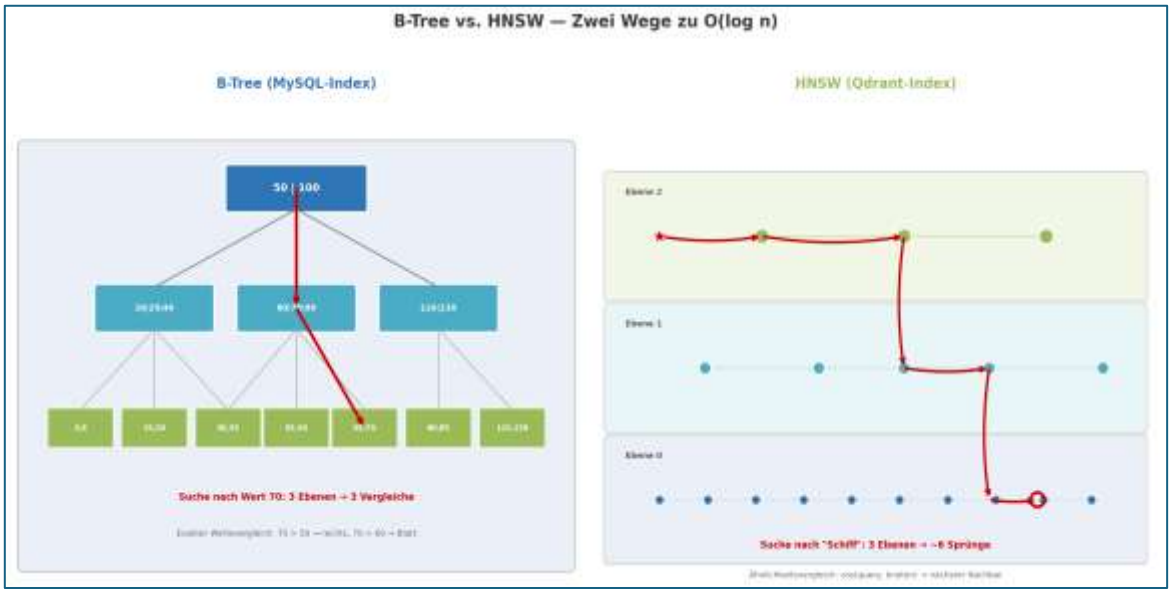


Abbildung 5: B-Tree vs. HNSW – gleiche Komplexität, unterschiedliche Datentypen und Vergleichsoperationen.

4.3 Warum O(log n) bei HNSW?

Die Ebenen im HNSW halbieren (näherungsweise) die Knotenmenge. Die Anzahl der Ebenen ist deshalb proportional zu log(n):

Anzahl Ebenen = log1/p(n) ≈ ln(n) / ln(1/p)

Bei n = 1.000.000 und p ≈ 0,36 ergibt das ca. 13–15 Ebenen. Auf jeder Ebene werden maximal ef Nachbarn geprüft (konstant). Die Gesamtkomplexität ist daher:

O(log n × ef) = O(log n) (da ef eine Konstante ist)

Der entscheidende Punkt: HNSW liefert **approximative** Ergebnisse (Approximate Nearest Neighbor, ANN). Das bedeutet, dass nicht garantiert immer der mathematisch exakt nächste Vektor gefunden wird. In der Praxis sind die Ergebnisse jedoch sehr genau – bei dramatisch höherer Performance.

Karsten Keßler
tutor@kkessler.de
tutor.kkessler.de
14.02.2026

5. Zusammenfassung

Konzept	Kernaussage
O-Notation	Beschreibt das Wachstumsverhalten. O(log n) = bei Verdoppelung nur +1 Schritt.
Brute Force	Vergleicht jeden Vektor → O(n). Zu langsam für große Datenmengen.
Graph-Suche	Vektoren als Nachbar-Graph → Greedy-Navigation in wenigen Schritten.
Hierarchie	Mehrere Ebenen: oben grob (Autobahn), unten fein (Nebenstraße).
Zufalls-Level	Exponentiell abnehmende Wahrscheinlichkeit → Pyramidenstruktur.
Parameter m	Kanten pro Knoten. Trade-off: Qualität vs. Speicher.
Parameter ef	Suchbreite. Trade-off: Genauigkeit vs. Geschwindigkeit.
B-Tree-Analogie	Gleiche Idee (O(log n)), aber für exakte Werte statt Vektoren.
ANN	Approximate Nearest Neighbor – nicht exakt, aber sehr genau und schnell.

Merksatz: HNSW ist für Vektoren das, was ein B-Tree-Index für Zahlen ist – eine hierarchische Datenstruktur, die den Suchraum auf jeder Ebene halbiert und so logarithmische Suchzeit ermöglicht.