Wurzel und Wurzelziehen

Dennis Komm, Jakub Závodný @ ClevAlg Kurs

20.11.2019

Wurzel

Wie berechnet man die Wurzel von einer Zahl N?

Wurzel

Wie berechnet man die Wurzel von einer Zahl N?

```
sqrt(N)
```

Wurzel

Wie berechnet man die Wurzel von einer Zahl N?

ohne der eingebauten `sqrt()` Funktion?

Wurzel

Einfacher = Wie berechnet man die **ganzzahlige** Wurzel von einer Zahl N?

Wurzel

Einfacher = Wie berechnet man die **ganzzahlige** Wurzel von einer Zahl N?

Das heisst, die grösste ganze Zahl w, so dass $w * w \leq N$?

Wurzel

Einfacher = Wie berechnet man die **ganzzahlige** Wurzel von einer Zahl N?

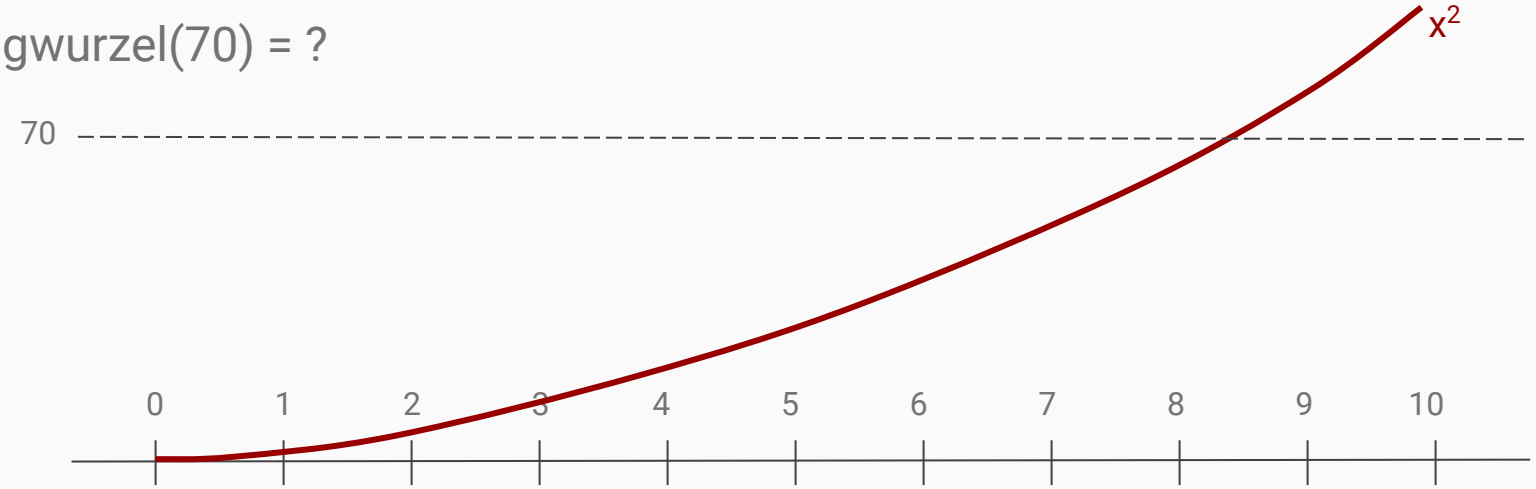
Das heisst, die grösste ganze Zahl w, so dass $w * w \leq N$?

gwurzel(70) soll 8 zurückgeben, weil $8 * 8 \leq 70$ aber schon $9 * 9 \not\leq 70$.

Wurzel

gwurzel(N) ist die grösste ganze Zahl w, so dass w * w ≤ N

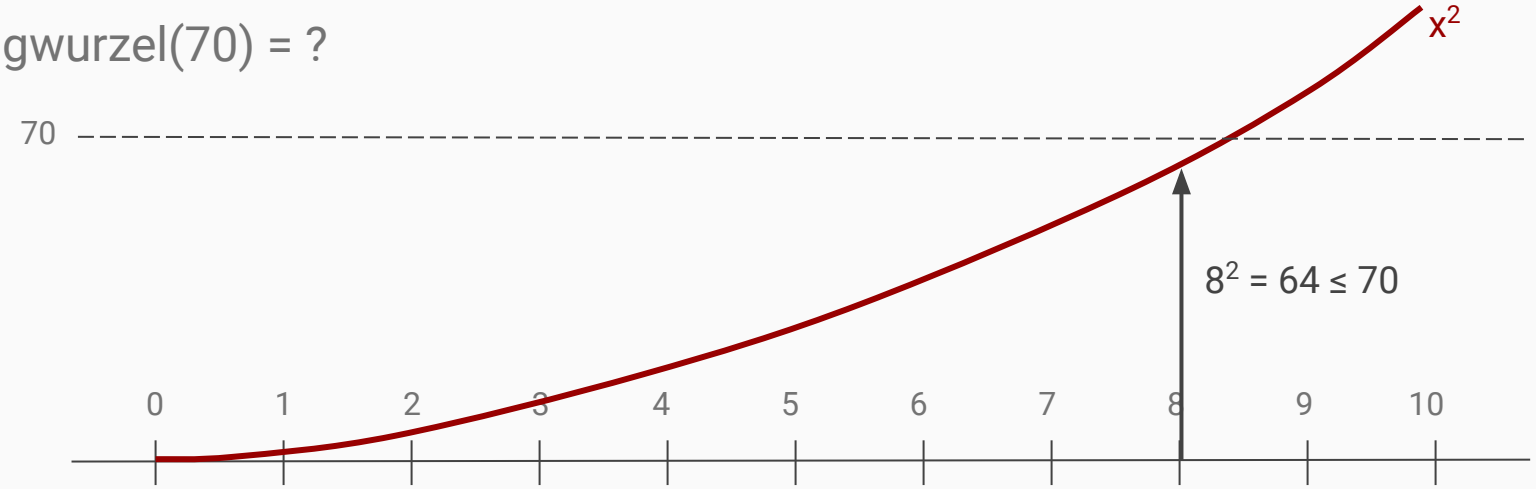
gwurzel(70) = ?



Wurzel

gwurzel(N) ist die grösste ganze Zahl w, so dass w * w ≤ N

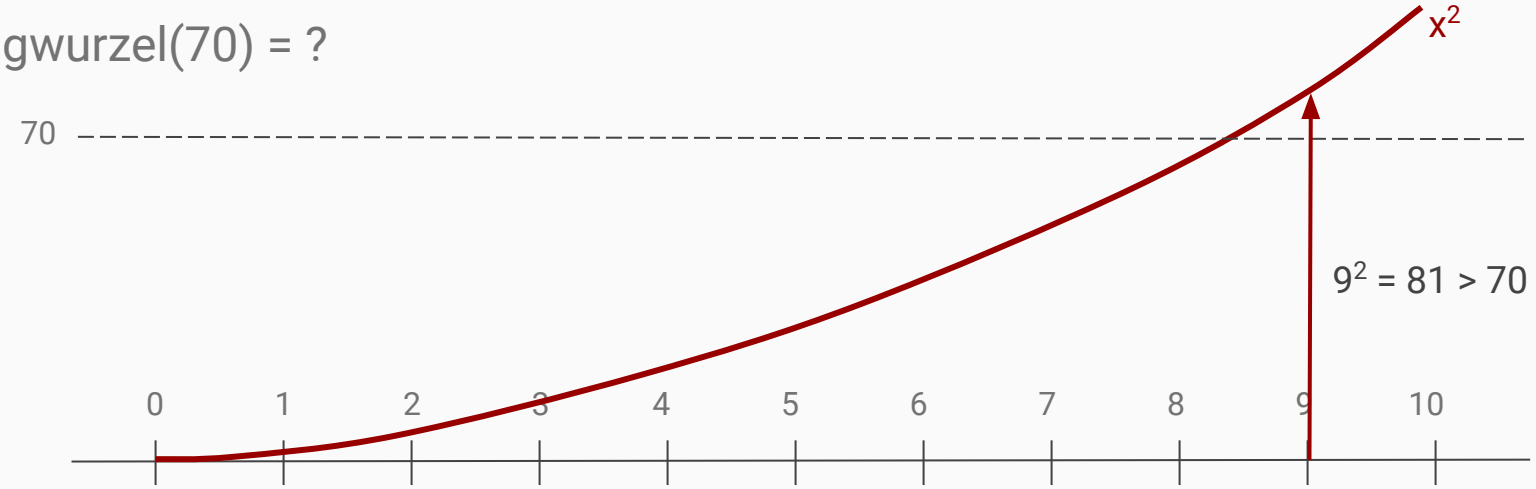
gwurzel(70) = ?



Wurzel

gwurzel(N) ist die grösste ganze Zahl w, so dass w * w ≤ N

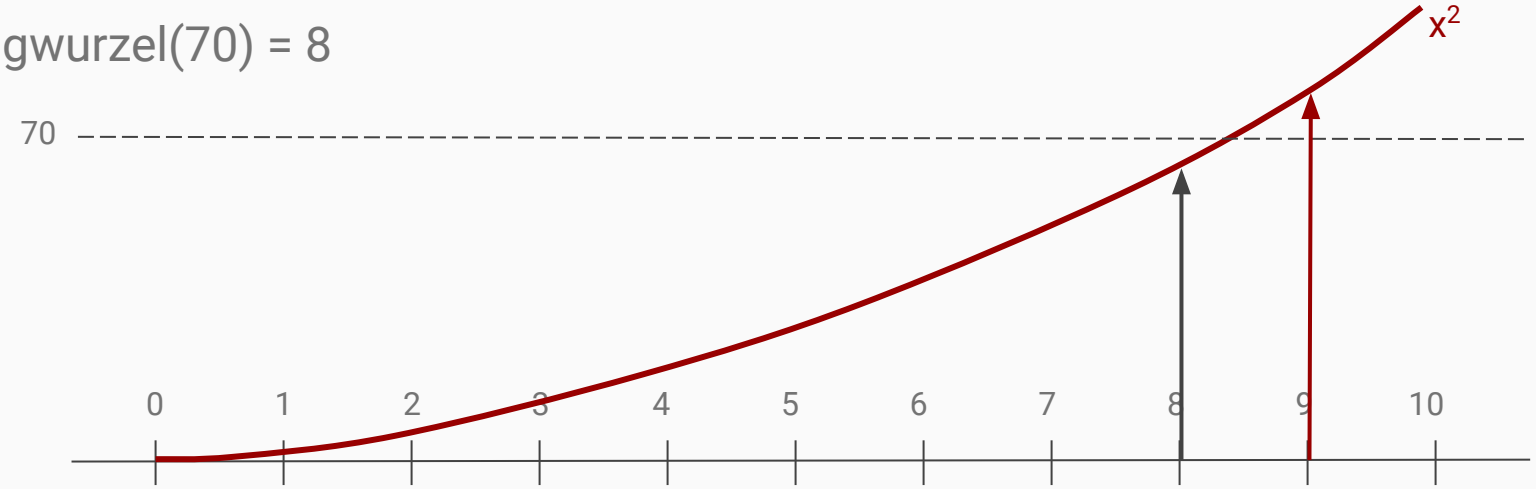
gwurzel(70) = ?



Wurzel

gwurzel(N) ist die grösste ganze Zahl w, so dass w * w ≤ N

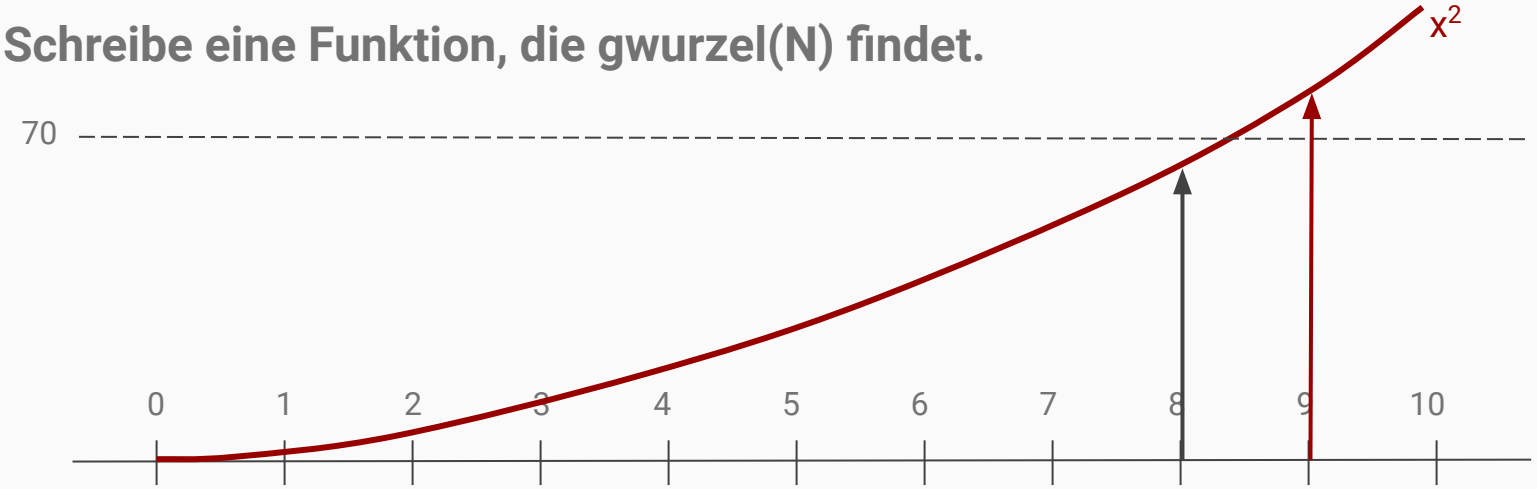
gwurzel(70) = 8



Wurzel

gwurzel(N) ist die grösste ganze Zahl w, so dass w * w ≤ N

Schreibe eine Funktion, die gwurzel(N) findet.



Wurzelziehen: Aufgaben

gwurzel(N) ist die grösste ganze Zahl w, so dass w * w ≤ N

1) Schreibe eine Funktion, die gwurzel(N) findet.

Wie schnell kann sie gwurzel(123456789012345) berechnen?
Was ist ihre Laufzeit für allgemeine N?
Geht es noch schneller? (*Hinweis: Ja.*)

2) Schreibe eine Funktion, die wurzel(N) findet. (Mit Genauigkeit ±0.000001)

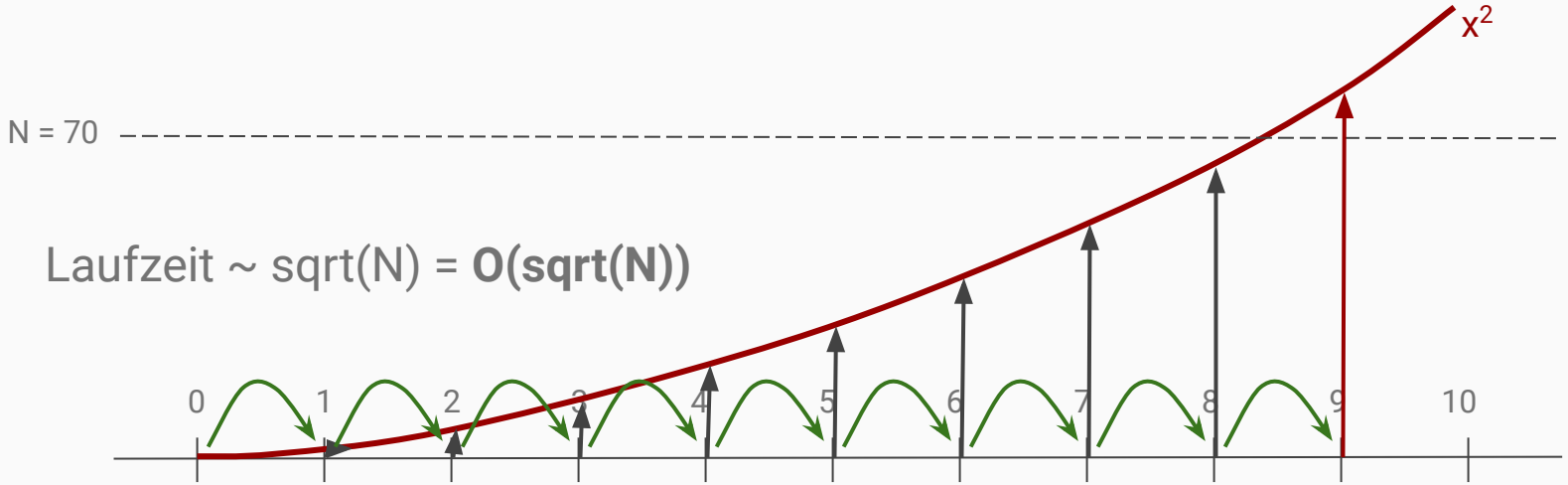
Hinweis: Mit diesen Befehlen kannst du Zeit messen:

`import time` ← *nur einmal ausführen, um das Modul* `time` *zu importieren*

`time.clock()` ← *gibt die aktuelle Zeit auf einer Stoppuhr zurück*

Algorithmus #1: Naiv

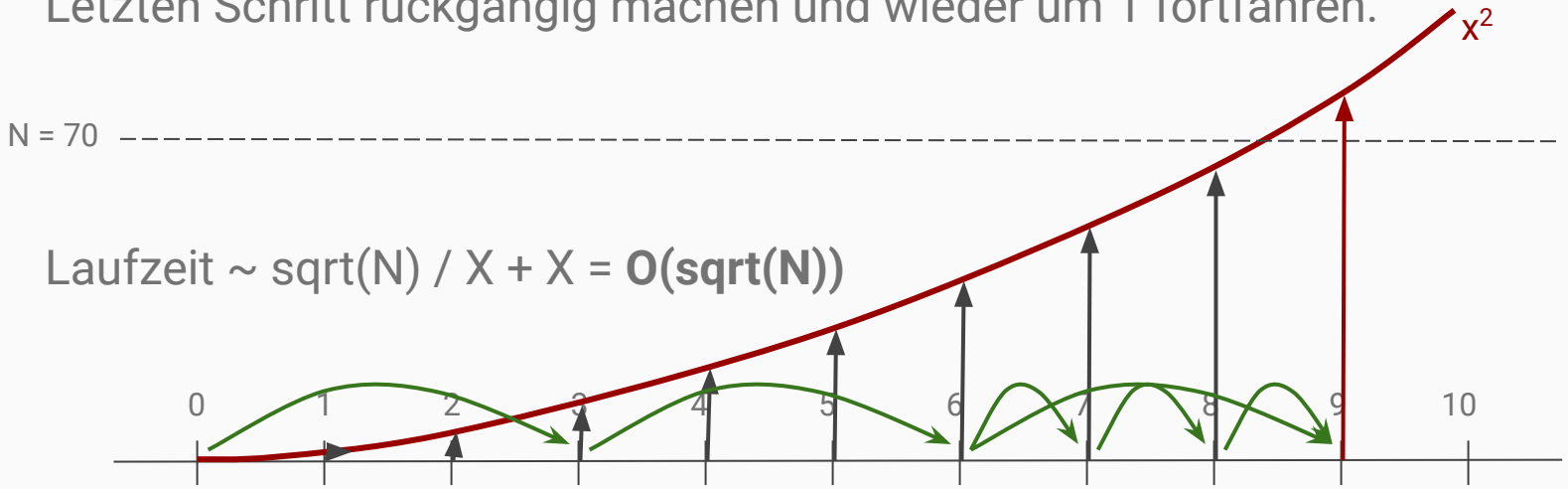
Auf w = 0 beginnen, alle Zahlen probieren, bis w * w ⩽ N.



Laufzeit ~ sqrt(N) = **O(sqrt(N))**

Algorithmus #2: Grösserer Schritt

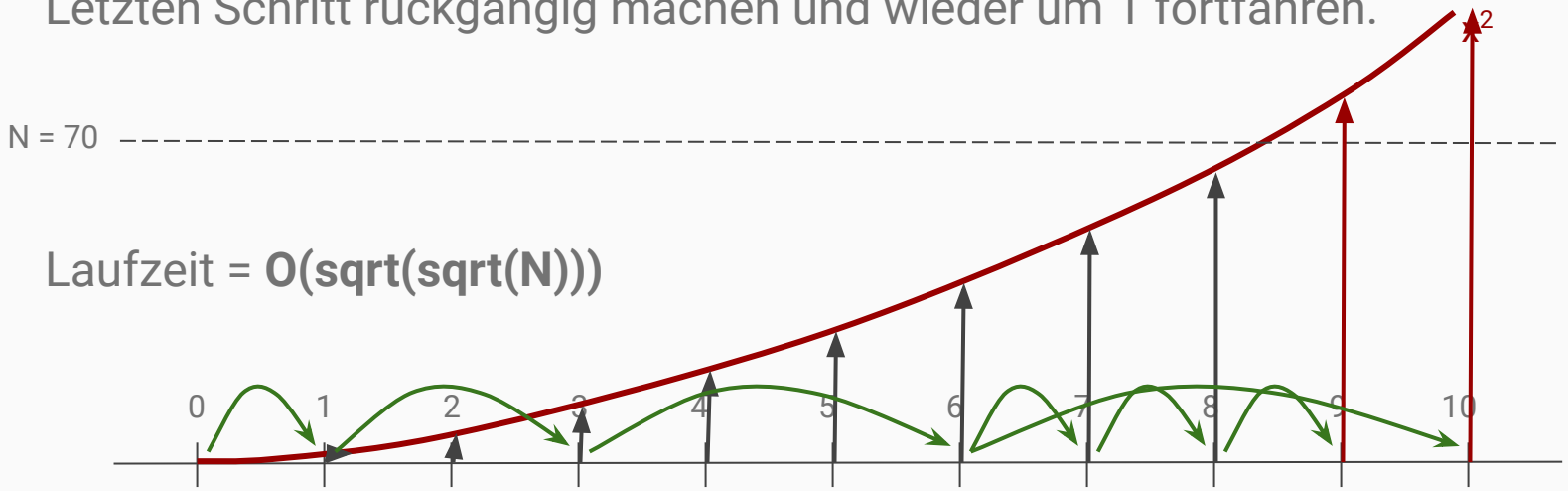
Von w = 0, w immer um fixen Schritt X erhöhen, bis w * w ⩽ N.
Letzten Schritt rückgängig machen und wieder um 1 fortfahren.



Laufzeit ~ sqrt(N) / X + X = **O(sqrt(N))**

Algorithmus #3: Zunehmender Schritt

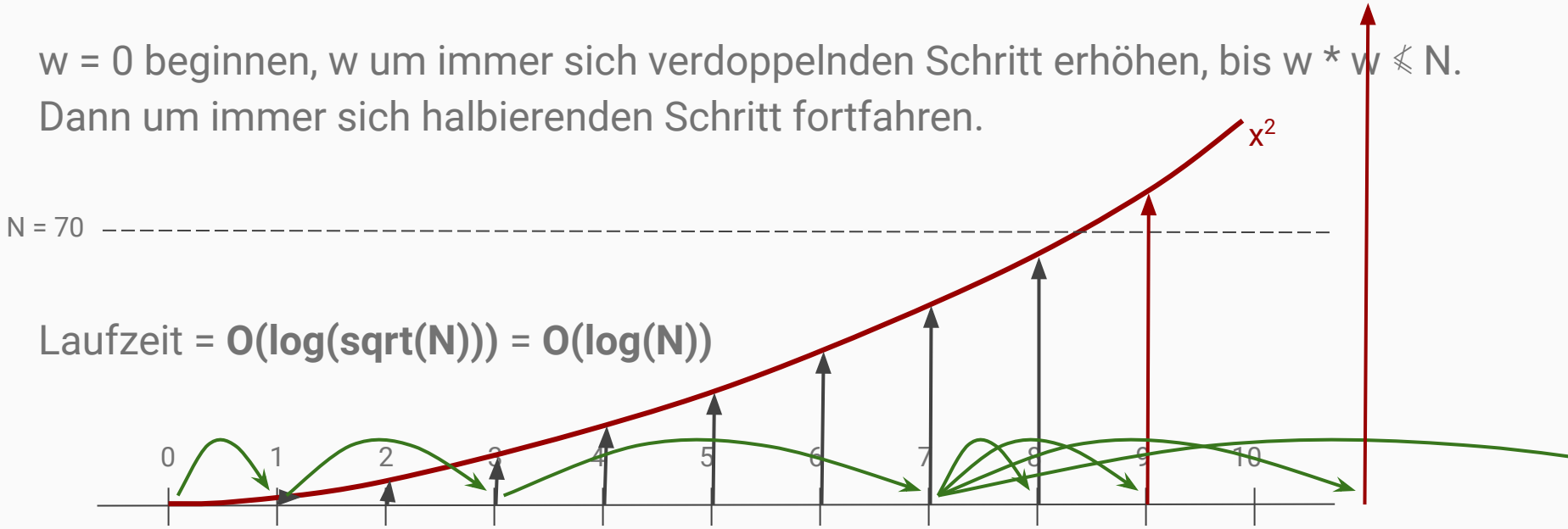
Von w = 0, w um Schritt 1, 2, 3, 4, ... erhöhen, bis w * w ⩽ N.
Letzten Schritt rückgängig machen und wieder um 1 fortfahren.



Laufzeit = **O(sqrt(sqrt(N)))**

Algorithmus #4: Verdoppelnder und Halbierender Schritt

w = 0 beginnen, w um immer sich verdoppelnden Schritt erhöhen, bis w * w ⩽ N.
Dann um immer sich halbierenden Schritt fortfahren.

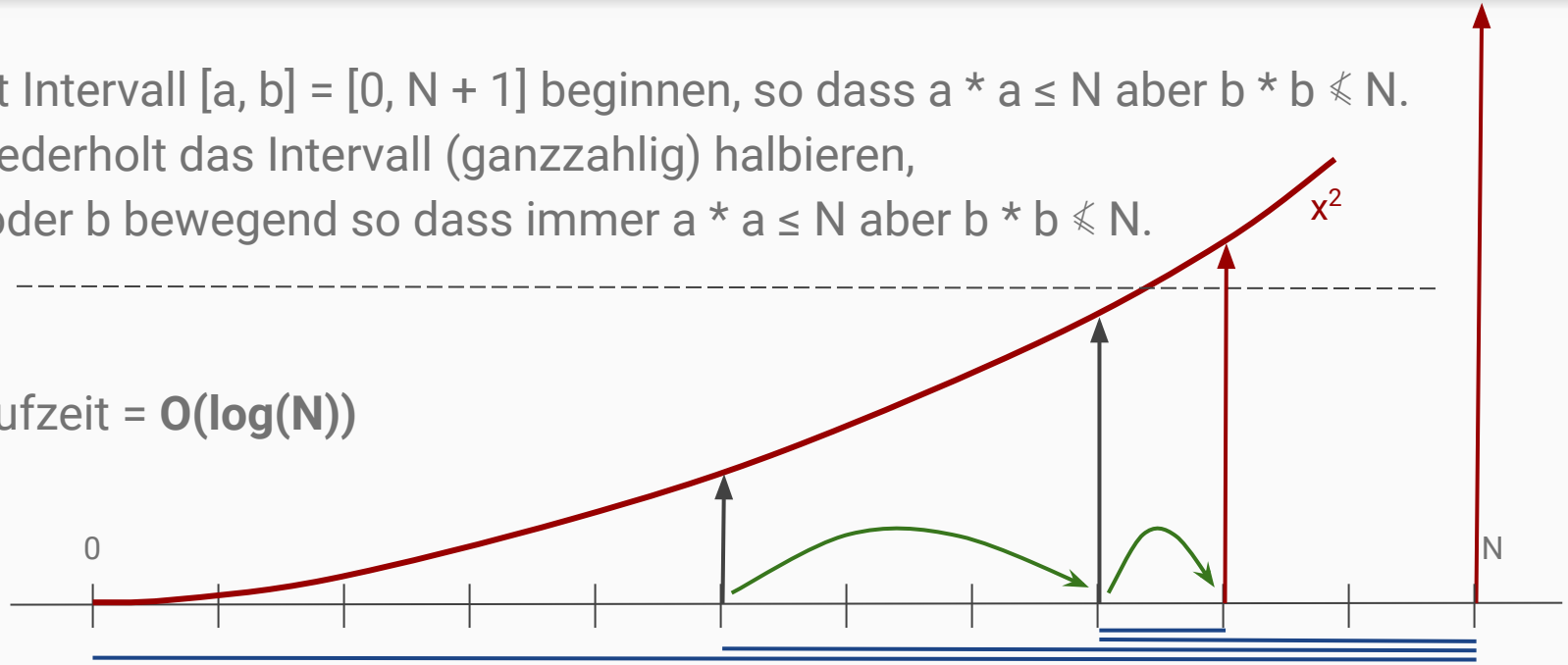


Laufzeit = **O(log(sqrt(N))) = O(log(N))**

Algorithmus #5: Binäre Suche

Mit Intervall [a, b] = [0, N + 1] beginnen, so dass a * a ≤ N aber b * b ⩽ N.
Wiederholt das Intervall (ganzzahlig) halbieren,
a oder b bewegend so dass immer a * a ≤ N aber b * b ⩽ N.

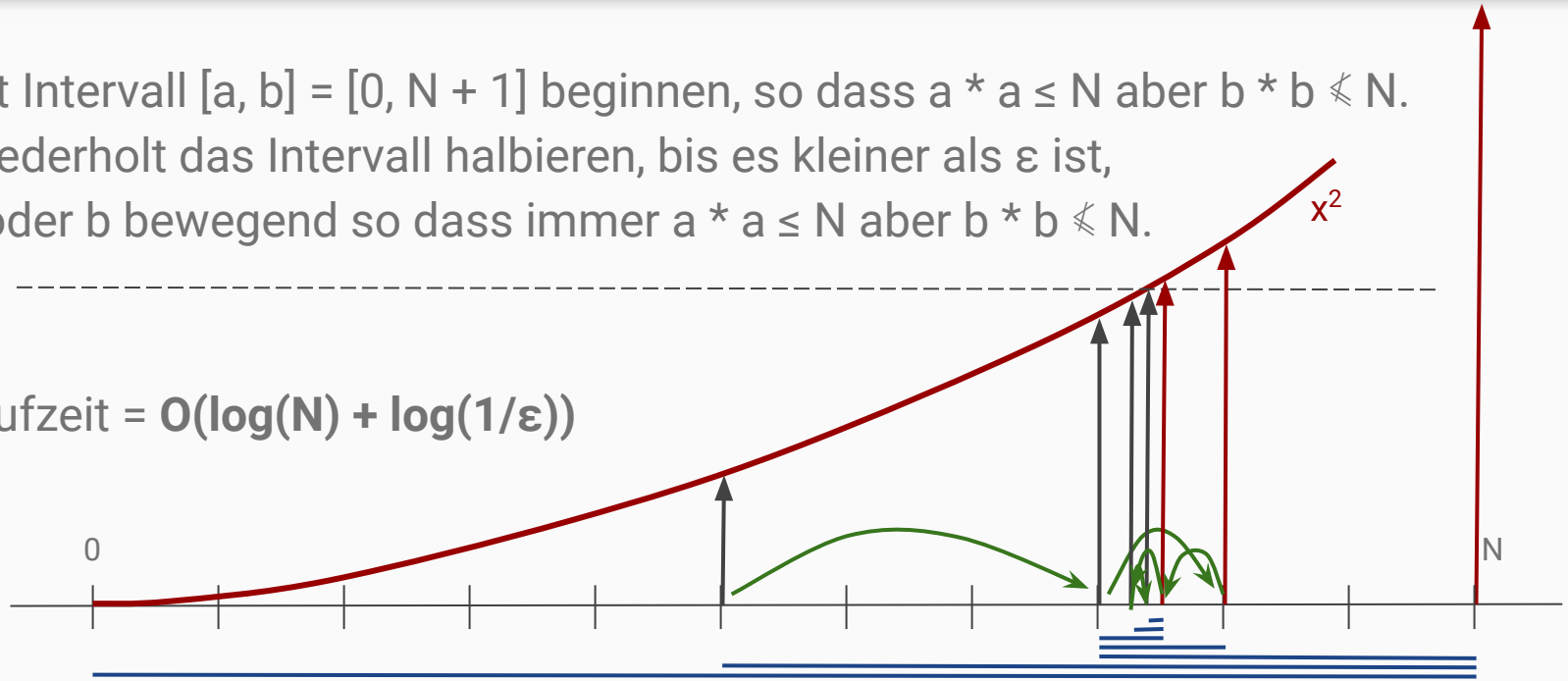
Laufzeit = **O(log(N))**



Algorithmus #5b: Binäre Suche mit beliebiger Genauigkeit ε

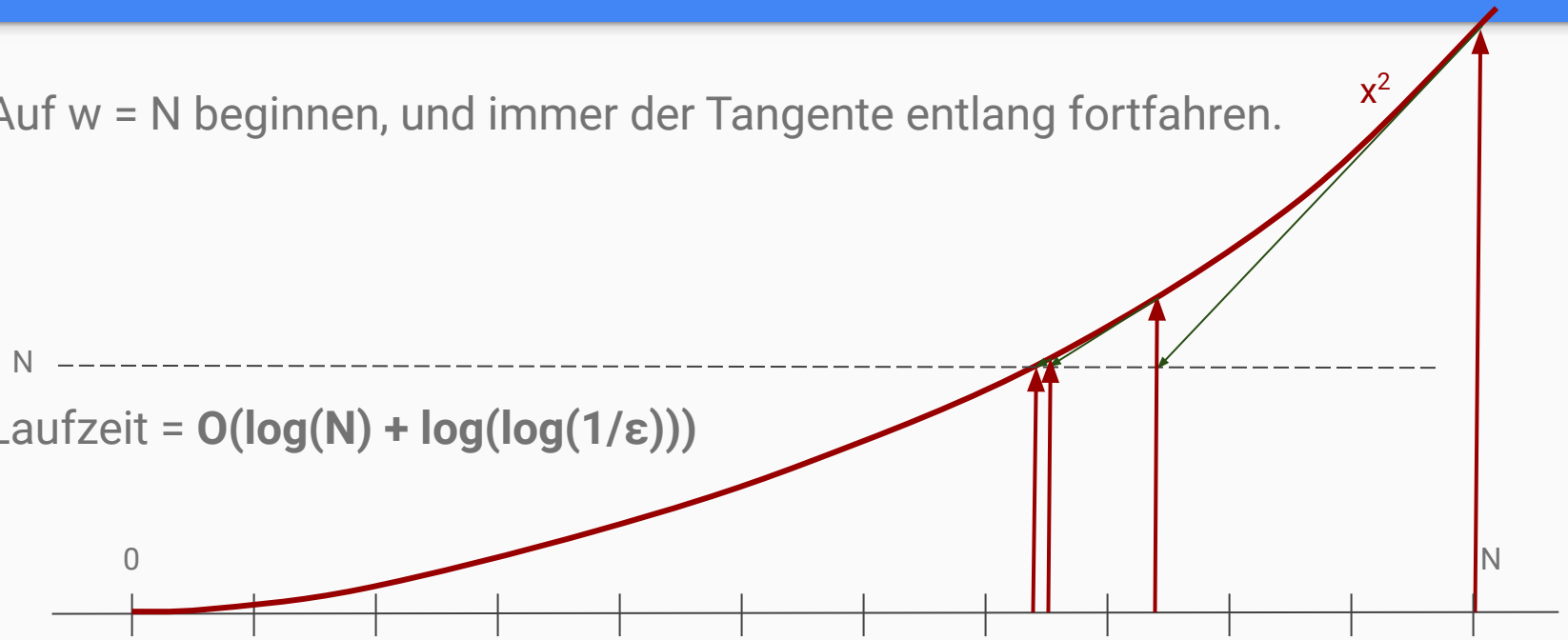
Mit Intervall [a, b] = [0, N + 1] beginnen, so dass a * a ≤ N aber b * b ⩽ N.
Wiederholt das Intervall halbieren, bis es kleiner als ε ist,
a oder b bewegend so dass immer a * a ≤ N aber b * b ⩽ N.

Laufzeit = **O(log(N) + log(1/ε))**



Algorithmus #6: Newton-Verfahren mit beliebiger Genauigkeit ε

Auf w = N beginnen, und immer der Tangente entlang fortfahren.



Laufzeit = **O(log(N) + log(log(1/ε)))**