

Complexity Theory in a Nutshell

CS460 Databases for Data Scientists



Why?

- Complexity theory
 - essential means of analysis in computer science
 - describes the behavior of an algorithm
 - often not known to non-computer scientists
- Or: what the hell does $O(N^2)$ mean?


What?

- Measure the performance of algorithms
 - how much time does it need? → time complexity
 - how much memory does it need? → memory complexity
- It's not about *absolute* numbers
 - that would be: it takes 21 seconds
- It's about *relative* numbers
 - relative to, e.g., no. of rows
- It's about *scaling*
 - i.e.: what happens if I double the number of rows?



Depends on
hardware etc.

First Example

- Reading N customer records from disk
 - N is a variable
 - each record takes a time t
 - i.e., the total time is $N*t$
- t may vary
 - e.g., by buying a hard disk twice as fast 
 - thus, we usually do not consider t
 - we say: the complexity of reading N customers is $O(N)$
- $O(N) \leftrightarrow$ linear scaling
 - i.e., double the number of customers, double the time
 - the actual hard disk speed does not matter here $\rightarrow O(0.5*N) = O(N)$

Second Example

- Storing the pairwise distances between N cities
 - we need to store $0.5 * N * N$ distances
 - each distance needs b bytes $\rightarrow 0.5 * b * N * N$
- Again
 - we may tweak the constant factor b
 - e.g., using more/less decimal digits
 - we already know that constant factors do not change the complexity
- $O(N^2) \leftrightarrow$ quadratic complexity
 - twice as many cities \rightarrow four times as many distances to store
 - that is not affected by 0.5 nor by b !

“Calculating” with Complexities

- Constant factors are neglected
 - $O(N) = O(2 * N) = O(1,000 * N)$
- The highest complexity dominates the overall complexity
 - $O(N + N^2) = O(N^2)$
- $O(1)$ denotes constant complexity
 - i.e., it is independent of problem size
 - e.g.: add a new record to a table
 - in theory, that should take an equal amount of time
 - irrelevant of the size of the table

Further Notes

- There might be more than one variable
 - e.g., storing a table with N records and C columns uses $O(N \cdot C)$ memory
- Complexity often depends on the solution, not the problem
 - example: storing who is sitting in which office

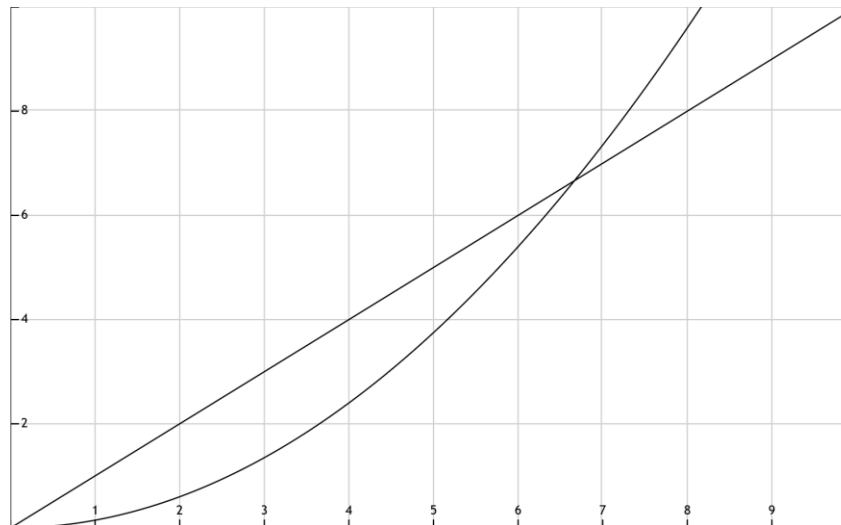
Person	Room
Peter	B0.01
Mary	B0.04
John	B0.02
Julia	B0.03

Person	B0.01	B0.02	B0.03	B0.04
Peter	yes	no	no	no
Mary	no	no	no	yes
John	no	yes	no	no
Julia	no	no	yes	no

- Storage and time complexity may be different
 - sometimes, we have to trade them off against each other

Comparison of Complexities

- Complexities can be compared
 - $O(1) < O(\log n) < O(n) < O(n * \log n) < O(n^2) < O(n^c) < O(c^n)$
- Complexity helps analyzing scalability
 - e.g., assessing suitability for larger problems
 - e.g., choosing between different variants



Complexity and Worst Case Behavior

- Complexity describes the worst case behavior
 - think: what happens for very big data?
 - think: what happens in very degraded cases?
- Example for big scales
 - Approach A takes $0.00001 * N^2$, approach B takes $10,000 * N$
 - Unless your N gets very large, you will use A, although $O(N^2) > O(N)$
- Example for degraded cases
 - Storing the ratings of C customers and I items is $O(C * I)$
 - However, the actual number is much lower
 - Each customer only rates a very small fraction of C

Questions?

