Basics
Big O Notation

Torsten Houwaart & Martin Raden

Lehrstuhl für Bioinformatik
Institut für Informatik
Albert-Ludwigs-Universität Freiburg

Course Bioinformatics I

built on September 20, 2018
Algorithms are the basis of bioinformatics

Design goals of algorithms:
- Correctness - Cool! Proofable!
- Termination - Otherwise you wouldn’t get a result!
- Efficiency - Naïve solutions are trivial!

For large data sets, efficiency depends upon
- internal representation of data (data structure)
- the applied algorithm

Functionally equivalent algorithms often exhibit drastic differences in efficiency (complexity)

Examples: searching & sorting
Example: Searching

- Problem: find an entry within a (sorted) list of length \( n \)

- How to search for an entry? Different strategies ..
  - Naïve = Sequentially! How many comparisons on average?

  \[
  C_{avg}(n) = \frac{1}{n} \cdot \sum_{i=1}^{n} i = \frac{n + 1}{2}
  \]

- Binary search:
Example: Binary Search

For binary tree $b = 2$

The height of the above tree is answer to the following question: How many times we divide problem of size $n$ by $b$ until we get down to problem of size $1$?

The other way of asking same question:

When $\frac{n}{b^1} = 1$ [in binary tree $b = 2$]

i.e. $n = b^x$ which is $\log_b n$ [by definition of logarithm]

(Figure from Algorithms and Data Structures, Georgios Gousios, TU Delft)
Example: Searching

- Naïve solution vs binary search

- Sorted input:
  - naïve solution: on average \((n + 1)/2\) comparisons
  - binary search: on average \(\log n\)

- What about *unsorted* input?
  - naïve solution: as before (no sorting needed)
  - sort entries \((n \log n) + \text{binary search (log } n \text{ per search)}\)
Time complexity: asymptotic analysis

- Needed: classification of algorithm complexity

- Use upper bounds ("worst case"): Big O Notation

- Given:
  \[ n = \text{length of arbitrary input} \text{ (eg. size of phone book or length of DNA string)} \]
  \[ f(n) = \text{maximal runtime for input length } n \text{ (eg. in \#operations)} \]
  \[ = \text{strictly positive for large } n \]

- Wanted: classification ala "\(f(n)\) is in complexity class . . ."
Definition Big O Notation

- Complexity class \( = \) group of similar (runtime) functions
  - named \( O(g) \) via “simplest” / reference function \( g \) within
  - e.g. \( O(n) \) with \( g(n) = n \) covers all linear functions
    like \( f(n) = 0.2n + 10 \) or \( f(n) = 123n \)

- Class \( O(g) \), all functions of a specific complexity \( g \):

\[
O(g) = \{ f \mid \exists c > 0 : \exists n_0 > 0 : \forall n \geq n_0 : f(n) \leq c \cdot g(n) \}
\]

- eg. \( O(n) = \) linear functions (and simpler)

![Graph showing \( f(n) \) vs \( g(n) \) and \( cg(n) \) with \( n_0 \) and \( c \) values calculated]

\( f(n) = 0.2n + 10 \) in \( g(n) = n ? \)

\[
\forall n \geq n_0 : 0.2n + 10 \leq cn
\]

\[
0.2 + \frac{10}{n} \leq c \quad \rightarrow \quad n_0 = 1, \quad c = 10.2
\]
Important complexity classes

- Classes in increasing complexity:
  
  **cheap**
  
  - $O(1)$ constant costs
  - $O(\log n)$ logarithmic increase
  - $O(n)$ linear increase
  - $O(n \log n)$
  - $O(n^2)$ quadratic increase
  - $O(n^3)$ cubic increase
  - $O(2^n)$ exponential increase

- **costly**

  $O(f + h) = O(f) + O(h) = O(\max(f, h))$
  
  eg. sort + binsearch = $O(n \log n) + O(\log n) = O(n \log n)$
How to assess runtime function \( f(n) \)?

- Given input of length \( n \)

- Get number of computation steps \( S \)
- Identify complexity class \( s(n) \) of \( S \) (ie. \( S \) as a function of \( n \))

- For each computation step:
  - Get maximal number of operations \( K \) per step
  - Identify complexity class \( k(n) \) of \( K \) (ie. \( K \) as a function of \( n \))
  - If based on substeps: analyze each and take maximum

- Product \( s(n)k(n) \) yields overall complexity \( f(n) \)
  - eg. binsearch: \( s(n) = \log n, k(n) = 1 \) : \( O(\log n) \)
  - eg. sorting: \( s(n) = n, k(n) = \log n \) : \( O(n \log n) \)