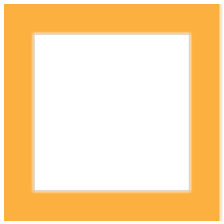
Introduction
Algorithms and Datastructures

Marjahan Begum and Anders Kalhauge



cphbusiness

Spring 2017

Outline

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Marjahan Begum

mbeg@cphbusiness.dk (91 81 25 23)

- PhD in Computer Science Education
- 10 years experience in reseaching learning
- Main interests
  - Machine learning and algorithms
  - Programming

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Anders Kalhauge

aka@cphbusiness.dk (21 72 44 11)

- 26 years experience as IT consultant in the private sector
- 16 years teaching computer science for students and private companies
- Main interests
  - Programming and programming languages
  - Development of large scale systems
  - Software architecture

Overview of the course

cphbusiness
COPENHAGEN BUSINESS ACADEMY

We have decided on four major topics:

Introduction **2 weeks** Introduction to algorithms and complexity. Basic sorting algorithms.

Data Structures **4 weeks** Basic data structures and searching algorithms. Heaps, heap sorting, and priority queues.

Graphs **4 weeks** Graph types, directed and weighted graphs. Implementation of graph data structures. Algorithms for graphs including searching for spanning trees and shortest path.

Application **5 weeks** Application of algorithms including scheduling, text mining, and big data.

Objectives

At the end of the course the students:

- Have experience with a representative selection of algorithms and data structures
- Knows what's inside the abstract data types of programming framework
- Knows how compare algorithms and data structures' time complexity
- Knows how to use algorithms on big data.

At the end of the course the students can:

- Use relevant algorithms in own applications
- Calculate time and space complexity (big O)
- Handle big and faulty data

Exam

The exam is oral but as part of the exam a written test is performed in the end of the course. For the oral part, the student will prepare a (app. ten minutes) presentation of the solution of one of the major assignments. Further discussions will be based on the presentation, but can include all aspects of the curriculum.

In order to be approved for the exam:

- All five major assignments must be handed in
- Attendance at the questionare
- At least 80 study points must be obtained

Study Points

- Attendance: 20
- Hand in of major assignments (8 per assignment): 40
- Hand in of minor assignments (4 per assignment): 12
- Attendance at the questionaire: 28

Exercise 1

- Create temporary groups of four or five
- Follow instructions . . .

Sum of array

```
int sumOfarray(int[] list) {                    // time units
  int total = 0;                                //   1
  for (int i = 0; i < list.length; i++)         //   2*n
     total = total + list[i];                   //   2*n
  return total;                                 //   1
  }
```

Total unit of time:

$$O(1 + 2n + 2n + 1) = O(2 + 4n) = O(n)$$

ThreeSum example

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
int count(int[] a) {
  int n = a.lenght;
  int count = 0;
  for (int i = 0; i < n; i++)
      for (int j = i + 1; j < n; j++)
          for (int k = j + 1; k < n; k++)
              if (a[i] + a[j] + a[k] == 0) count++
  return count;
  }
```

To measure the efficiency of algorithms, We calculate their time and space complexity.

The notation for that is big-O, or order of growth:

$$O(order)$$

Properties of Big-O

Big-O is a measure of complexity, not of actial running time or space consumption.

In other words, big-O is a measure of the quality of the algorithm, not of the computer.

Big-O gives a picture of what happens if we scale the problem by running the algorithm on larger number (normally n) of data.

Therefore the running time (a constant c) of the individual steps

are not relevant:

$$O(c\,x) = O(x)$$

Contstant growth

The algorithm uses the same time or space no matter how much much data is involved.

- Pushing an element on a stack
- Returning the size of an array
- Calculating a single step

$$O(1)$$

Remember:

$$O(cx) = O(x) \rightarrow O(7) = O(1000000) = O(1)$$

The algorithm separates the problem in two (or more) equally sized problems and solve those.

- Binary search
- Binary tree insertions and deletions

$$O(\log n)$$

$$\log_a n = \frac{\log_b n}{\log_b a} = c \log_b n, c = \frac{1}{\log_b a}$$

Therefore:

$$O(\log_{10} n) = O(\log_2 n) = O(\log n)$$

The algorithm uses one step per data element. Remeber that one step can take as much time as needed as log as the time is constant.

- Search for maximum in unordered data
- Copying an array
- Bucket sort or sorting using a Trie

$$O(n)$$

Again:

$$O(7n) = O(6,022^{23}n) = O(n)$$

Linearitmic growth

The algorithm separates the problem in two (or more) equally sized problems and solve those. The algorithm involves doing something with each element in the data.

- Merge sort and Heap sort
- Best case Quick sort

$$O(n \log n)$$

And of course

$$O(1000n \log_{10} n) = O(n \log n)$$

Quadratic growth

The algorithm works on (almost) any pair of elements in the data. Normally done by a double nestet loop.

- Selection and Insertion sort
- Bubble sort

$$O(n^2)$$

Consider:

$$n + (n-1) + (n-2) + \cdots + 1 + 0 =$$

$$(n+0) + (n-1+1) + \cdots + (\frac{n}{2} + \frac{n}{2}) = \frac{n(n-1)}{2}$$

$$O(\frac{n(n-1)}{2}) = O(n^2)$$

Cubic growth

The algorithm works on (almost) any triple of elements in the data. Normally done by a triple nestet loop.

- Tree Sum algorithm

$$(n^3)$$

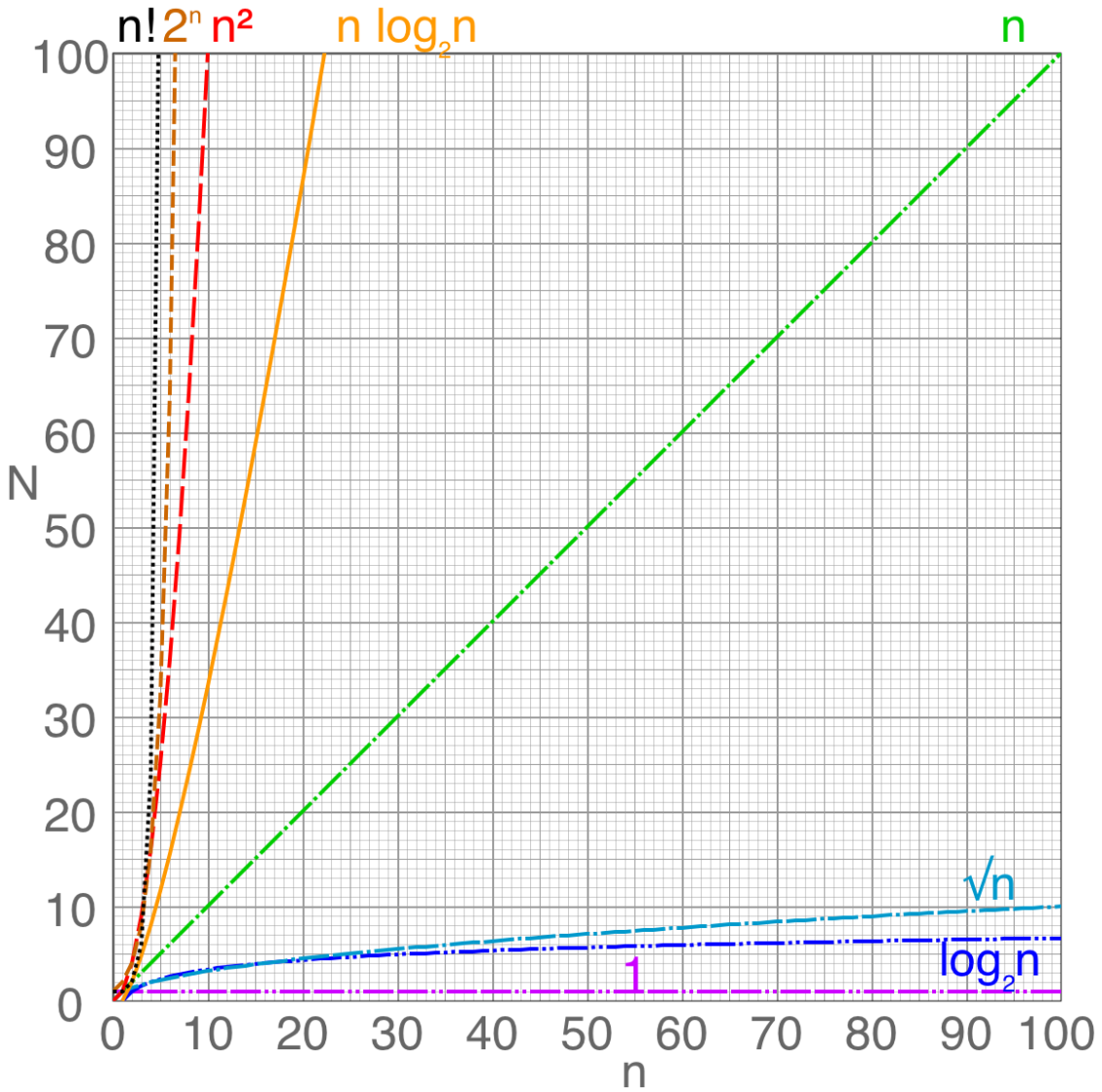
Exponential growth

The algorithm checks does an exhaustive search.

- Travelling salesman ($(n2^n)$)
- Matrix chain multiplication (brute force)

$$O(2^n)$$

Overview



Insertion Sort

To sort a stack of cards using insertion sort.

- take an unsorted stack of cards
- take one card from the unsorted stack and put in a new sorted stack
- while there are still cards in the unsorted stack
  - take a card from the unsorted stack
  - while the new picked card is of higher rank than the top card in the ordered stack
    - flip the top card of the order stack into a third temporary stack (without spoiling the order)
  - put the card taken from the unordered stack on top of the ordered stack
  - replace the flipped cards on top of the oredered stack.

Selection Sort

To sort a stack of cards using selection sort

- take an unsorted stack of cards
- while there are still cards in the unsorted stack
  - Pick the top card in the unsorted stack
  - while there are still cards in the unsorted stack
    - compare the card picked with the top card in the unsorted stack
    - put the card with the lowest rank in a new unordered stack, keep the other card on the hand
  - put the card you have in your hand on the top of the ordered stack (is empty the first time)
  - use the new unsorted stack as unsorted stack

Exercise 2 - Sorting Array

Create a class called `SortingAlgorithms`. This class should have an array of integer as a datafield and array size. The constructor should be used to create an array of given size.
`SortingAlgorithms` class should have three methods.

- One method for filling the array with random integers which you can call from the constructor.
- One method for implementing inserstion sort,
- One method for selection sort.

In the main method you create three objects with array size of 100, 1000, and 10000. Identify the elapsed times using `Stopwatch` [1] class when selection and insertion methods are called.

[1] see page 175 in Algorithms book