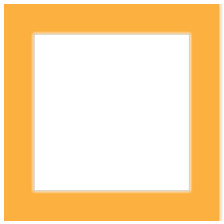
Algorithms & Data Structures
Introduction

Jacob Trier Frederiksen & Anders Kalhauge (assisting)



cphbusiness

Spring 2019

Outline – Week 5

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Introduction
- Your Instructors
- Course Overview

Warming Up
- Basic Sorting

Analysis of Algorithmic Complexity
- Sum of an Array
- Three-Sum-Zero Over an Array
- Computational Complexity
- $\mathcal{O}$ (order) notation

Sorting
- Shuffling Cards: Another Warm-up

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Jacob Trier Frederiksen

jtf@cphbusiness.dk (+4536154500, front desk)

- PhD in Computational Astrophysics
- 10 years experience in research, higher learning
- 3 years experience in business intelligence & machine learning
- Main interests
  - Computational modelling
  - Machine learning
  - Artificial Intelligence

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

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.

Course Overview – Objectives

cphbusiness
COPENHAGEN BUSINESS ACADEMY

At the end of the course the student will:

- Have experience with a representative selection of algorithms and data structures
- Know what is inside the abstract data types of the programming framework
- Know how compare algorithmsťime complexity, and associated data structures
- Know how to use algorithms on lage data sets

At the end of the course the students can:

- Select and utilize relevant algorithms in own applications
- Calculate time and space complexity (big $\mathcal{O}$)
- Have basic knowledge of data compression
- Handle big and faulty data

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:

- Major hand-in assignments must be completed (no less than 80% as minumum)!
- Participation in the written test (no study points, but mandatory for exam eligibility)
- Participation in giving a lesson in small groups.
- At least 80% of the study points must be obtained

Course Overview – Study Points

cphbusiness
COPENHAGEN BUSINESS ACADEMY

- Hand in of five major assignments (20 per assignment): 100
- Giving a lesson on an assigned or chosen/approved topic in small groups: 10

Basic Sorting

Here follows two basic sorting algorithms that we will be using in just a few minutes...

Basic Sorting – 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.

Basic Sorting – Insertion Sort

cphbusiness
COPENHAGEN BUSINESS ACADEMY

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.

Q: what did you find out? Discuss in class...

Basic Sorting – 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

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

Q: what did you find out? Discuss in class...

Sum of an Array

```
// We will measure the following in time units only,
// neglecting any movement of data and excess storage
// needs.
int sumOfarray(int[] list) {
  int total = 0;                              // ~ 1
  for (int i = 0; i < list.length; i++)       // ~ N
     total = total + list[i];                 // ~ 1
  return total;
  }
```

Total units of time consumption:

$$\mathcal{O}(1 + N + 1) \approx \mathcal{O}(1 + N) \approx \mathcal{O}(N)$$

Three-Sum-Zero Over an Array

cphbusiness
COPENHAGEN BUSINESS ACADEMY

```
// We will measure the following in time units only,
// neglecting any movement of data and excess storage
// needs.
int count(int[] a) {
  int n = a.length;                              // ~ 1
  int count = 0;                                 // ~ 1
  for (int i = 0; i < n; i++)                    // ~ N
   for (int j = i + 1; j < n; j++)               // ~ N
    for (int k = j + 1; k < n; k++)              // ~ N
     if (a[i] + a[j] + a[k] == 0) count++        // ~ 1
  return count;
  }
```

Total units of time consumption:

$$\mathcal{O}(1 + 1 + N + N^2 + N^3 + 1) \approx \mathcal{O}(N^3)$$

When accessing an array of length N while testing for the Two-Zero-sum, what is the complexity of that operation in algorithmic terms?

Task: We need to test whether any two values, chosen from N values, sum to zero.

Discussion: go to the web and find (a) formula(e) which expresses this particular complexity.

An series of rational numbers holds the answer – namely:

$$N + (N-1) + (N-2) \ldots = \sum_{k=1}^{N} \ldots$$

Q: So what about the Two-sum – what is the complexity?

When accessing an array of length N while testing for the Two-Zero-sum, what is the complexity of that operation in algorithmic terms?

Task: We need to test whether any two values, chosen from N values, sum to zero.

Discussion: go to the web and find (a) formula(e) which expresses this particular complexity.

An series of rational numbers holds the answer – namely:

$$N + (N-1) + (N-2) \ldots = \sum_{k=1}^{N} k = \frac{1}{2}(N^2 + N) \ .$$

A: The Two-Sum-Zero is $\mathcal{O}(N^2)$.

$\mathcal{O}$ (order) notation

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

The notation for that is $\mathcal{O}$, or order of growth:

$$\mathcal{O}\left(N^q\right),$$

where q is a constant, $q \in \mathbb{R}$.

$\mathcal{O}$ is a measure of complexity, not of actial running time or space consumption.

In other words, $\mathcal{O}$ is a measure of the cost of the algorithm, and has no bearing on the size or speed of the computer.

$\mathcal{O}$ 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:

$$\mathcal{O}(c\,N^q) \approx \mathcal{O}(N^q)$$

$\mathcal{O}(1)$, Constant 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

$$\mathcal{O}(1)$$

Remember:

$$\mathcal{O}(cx) \approx \mathcal{O}(x) \rightarrow \mathcal{O}(7) \approx \mathcal{O}(1000000) \propto \mathcal{O}(1)$$

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

- Binary search
- Binary tree insertions and deletions

$$\mathcal{O}(\log N)$$

No matter what log you choose:

$$\mathcal{O}(\log_{10} N) \sim \mathcal{O}(\log_2 N) \sim \mathcal{O}(\log N)$$

Q: prove, or dig out (i.e. from the web) a proof of this.

$\mathcal{O}(N)$, Linear growth

The algorithm uses one step per data element. Remember: each computational step can take as much time as needed, as long as that step requires a constant time, i.e. 10^{-9}s (a GHz machine).

- Search for maximum in unordered data
- Copying an array
- Bucket Sort
- Sorting using a Trie (project!)

$$\mathcal{O}(N)$$

Again:

$$\mathcal{O}(7N) \approx \mathcal{O}(6,022 \times 10^{23} N) \sim \mathcal{O}(N)$$

$\mathcal{O}(N \log N)$, Linearitmic growth

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

- Tree Sort
- Merge Sort and Heap Sort
- Quick Sort (best case)

$$\mathcal{O}(N \log N)$$

And of course

$$\mathcal{O}(1000N \log_{10} N) \sim \mathcal{O}(N \log N)$$

$\mathcal{O}(N^2)$, Quadratic growth

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

- Selection Sort
- Insertion Sort
- Bubble sort

$$\mathcal{O}(N^2)$$

Consider:

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

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

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

$\mathcal{O}(N^3)$, Cubic growth

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

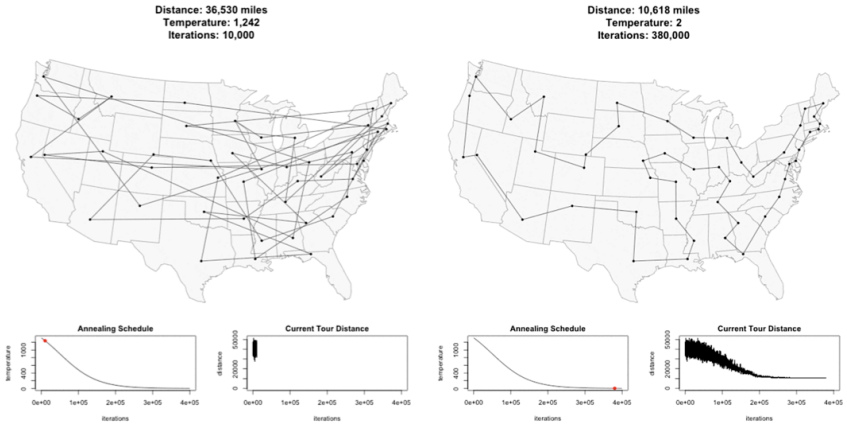
- Three-sum Algorithm

$$(N^3)$$

$\mathcal{O}(2^N)$, Exponential growth

The algorithm checks does an exhaustive search.

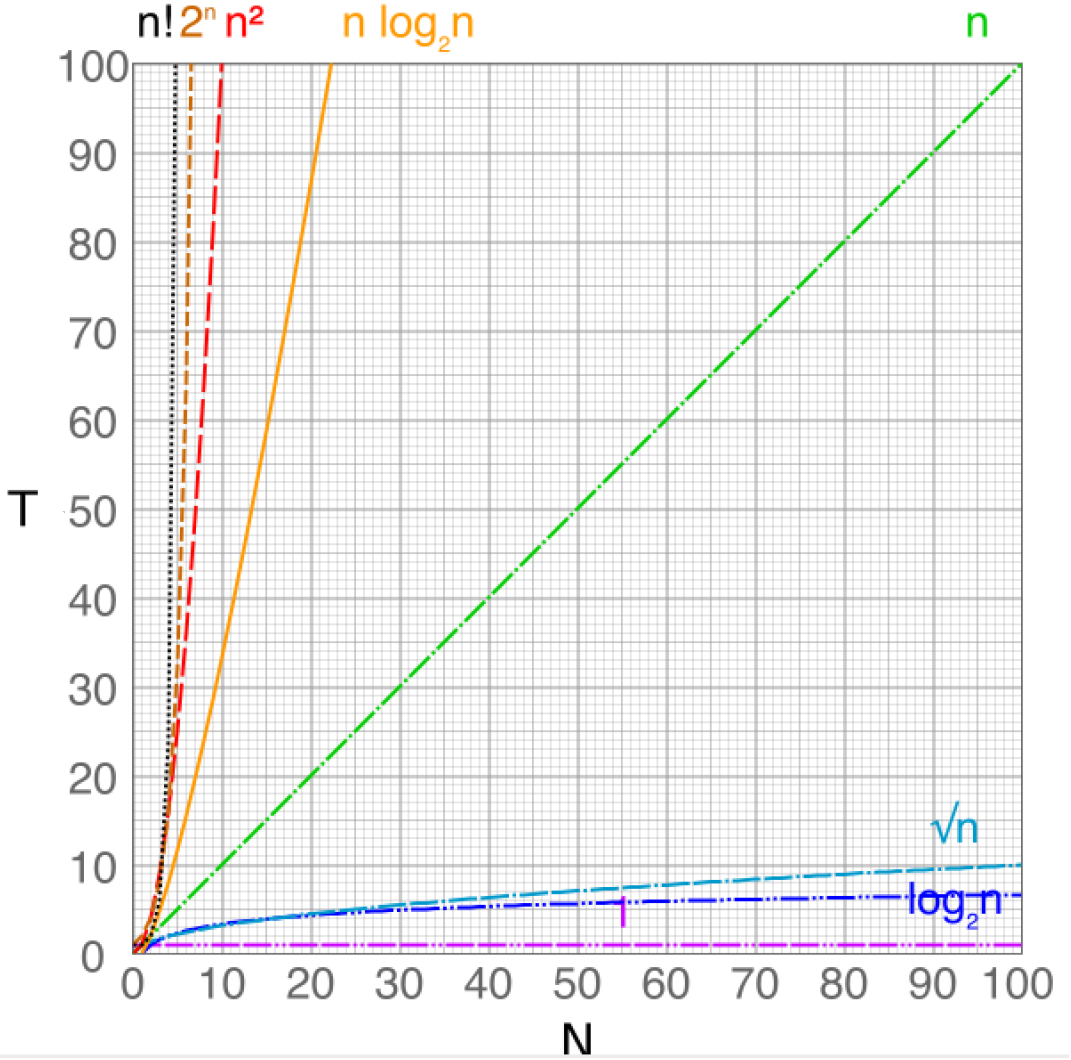
- Matrix chain multiplication (brute force, but for AI CNNs, case is much better . . .)
- Travelling salesman ($(N \cdot 2^N) \sim 2^N$)



$$\mathcal{O}(2^N)$$

$\mathcal{O}(f(N))$ – Overview

cphbusiness
COPENHAGEN BUSINESS ACADEMY



Shuffling Cards: Another Warm-up

cphbusiness
COPENHAGEN BUSINESS ACADEMY

We will find the recipe for another basic sorting algorithms, and test it manually.

1. Team up in groups of three.
2. Get your deck of cards/suit.
3. Shuffle **really** well (TRUE random is very hard, by the way)!
4. Google the `Tree-Sort` algorithm and adopt it for card games.
5. Get ready, set, go sort . . .
6. . . . leading to a manual test of simple application of a Tree-based algorithm. How many moves, how much time?

.

Shuffling Cards: Another Warm-up

cphbusiness
COPENHAGEN BUSINESS ACADEMY

We will find the recipe for another basic sorting algorithms, and test it manually.

1. Team up in groups of three.
2. Get your deck of cards/suit.
3. Shuffle **really** well (TRUE random is very hard, by the way)!
4. Google the `Tree-Sort` algorithm and adopt it for card games.
5. Get ready, set, go sort ...
6. ...leading to a manual test of simple application of a Tree-based algorithm. How many moves, how much time?

Q: what did you find out? Discuss in class...

Exercise 1 - 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 Insertion Sort,
- One method for Selection Sort.
- One method for an improved Three-Sum-Zero algorithm (book, `ThreeSumFast`, p.190), and time it, comparing with the `ThreeSum` for various sized arrays.

In the main method you create three objects with array sizes of 10^2, 10^3, 10^4, 10^5, and 10^6. Time your code using `Stopwatch` [1] class.

[1] see page 175 in Algorithms book