**Algorithm Design & Analysis: Efficiency and Complexity**

What is efficiency for software?

* Run as quickly as possible
* Respond as quickly as possible
* Use as little memory as possible
* Use as little network bandwidth as possible
* Use as little power as possible
* There could be more ways to measure efficiency

What exactly are we trying to optimise?

Scope of the solution matters, maybe when scaling up, our solution might not be efficient.

Linear complexity can be applicable to most algorithms, but not to all.

We can improve:

* Algorithms
* Data representation

Same algorithm on different machines produces result in different time. But, it is not important for measuring efficiency of the program.

**Space complexity** is linked to physical resources. It can be provisioned. Time complexity is much harder.

If time to solve problems takes too much computing power – we are talking about exponential complexinty.

Typical HD video

* 1920 x 1980 pixels

Solution

Slide 10

2D image, if we double the resolution. We will need n^2 pixels, as we work with two dimensions

For 3D – n^3 pixels.

**Time Complexity**

What is an algorithm? (Quantifiable, we are able to agree on the terms).

Criteria for algorithm efficiency can be conflicting: depends on what the algorithm does. (for nuclear reactor, we will need proof of correctness and clarity, etc.)

We are looking for the algorithm itself, not its implementation in a particular language.

What do we mean by operations?

Size is the size of the input.

We aim at worst-case analysis. (E.g. sorting algorithms – we look at situation, where the list is randomized, or sorted in an opposite way).

Linear search (slide #9)

Worst case scenarios: we look for the last value in the list, or for the value that is not in the list.

T(n) = n (time to search for an element in an array equals to the size of the array).

Matrix-Vector Multiplication

T(n) = 2n^2 (we do not consider assigning as an operation).

**Big-O Notation**

T(n) = 4n^3 + 1 = O(n^3)

T(n) = 1000 = O(1) (constant time complexity)

T(n) = n^2 + n + 1 = O(n^2)

How to determine the complexity?

* Count the number of loops…

Log Linear – consists of two loops, one is linear, another is logarithmic

Exponential complexity should be avoided!

N = 10 2^10 = 1024

N = 20 2^20 = 10^6

N = 30 2^30 = 10^9

Polynomial O(n) O(n^2) O(n^3) – tractable, meaning it’s fine.

Exponential O(2^n), O(C^n) – not tractable, we should avoid this level of complexity.

For any computing systems 65536 steps is not significant.

One million elements is not that surprising, as we might need to process millions of elements.

Slide #33

We can sort the elements on the list beforehand.

Log\_2(1000)

How to compute overall complexity?

Sequential complexity (loops/methods do not call each other): take “maximum” of complexities. **O(n) + O(n^2) = O(n^2)**

**Function/Method Calls**

O(n) x O(log\_2\_n) = O(n\* log n) The base of the log is assumed to be 2.

**Traveling Salesperson Problem (TSP)**

**NP class**

P = NP?