Introduction to intelligent systems

Algorithms

Mikkel N. Schmidt

Technical University of Denmark,
DTU Compute, Department of Applied Mathematics and Computer Science.

Overview

1. Algorithms
2. Algorithmic complexity
3. Divide and conquer / recursion
4. Software tools
5. Tasks

Feedback group

- Viet Hoang Nguyen
- Lukas Peter Dyhr
- Philip Kierkegaard
- Ali Mohammed Fathi Afif

Learning objectives

- I Algorithmic complexity.
- I Understand an algorithm from description or Python code.
- II Time complexity function.
- II Best, average and worst case complexity.
- II Big-O notation.

I Understand the concepts and definitions, and know their application. Reason about the concepts in the context of an example. Use correct technical terminology.

II As above plus: Read, manipulate, and work with technical definitions and expressions (mathematical and Python code). Carry out practical computations. Interpret and evaluate results.

Algorithms

What is an algorithm

- Unambiguous specification of how to solve a problem
- Expressed in a well-defined formal language
- Starts from initial state and input
- Proceeds through a finite number of steps
- Eventually terminates and produces an output

en.wikipedia.org/wiki/Algorithm

Levels of description

High level description Describes algorithm in normal language, ignoring implementation detail.

Implementation description Detailed description of exactly which actions must be performed by the computer.

Sipser, Introduction to the theory of computation, 2006

Example: Finding the largest number in a list

1. Assume the first number is the largest number
2. For each remaining number in the set: if this number is larger than the current largest number, consider this number to be the largest number
3. When there are no numbers left in the set to iterate over, consider the current largest number to be the largest number of the list

Example list

99 83 125 12 5 256 31 192

en.wikipedia.org/wiki/Algorithm

Exercise: An algorithm for sorting

1. Write down the numbers below on 8 small pieces of paper
2. Lay them in a random sequence on the table
3. Sort them starting with the smallest, and take notice of exactly which procedure you use
4. Write down a high level description of your sorting algorithm
5. Randomize the order of the numbers again, and follow your written procedure to the letter to sort the numbers again

Example list

99 83 125 12 5 256 31 192

Prepare to present your algorithm to the class

A simple sorting algorithm

1. Find the smallest number on the list
2. Remove the smallest number from the list and append it to the list of sorted numbers
3. Repeat the above steps until the list is empty

Example list

99 83 125 12 5 256 31 192

Good algorithms

Knuth: *"...we want good algorithms in some loosely defined aesthetic sense. One criterion ... is the length of time taken to perform the algorithm. (...) [Other] criteria are adaptability of the algorithm to computers, its simplicity and elegance, etc"*

Chaitin: *"a program is elegant, by which I mean that it's the smallest possible program for producing the output that it does"*

`en.wikipedia.org/wiki/Algorithm`

Algorithmic complexity

Time complexity

The time compexity function, $T(n)$:

- The “runtime” of an algorithm that operates on an input of length n.
- Can e.g. measure the number of required computational operations.

Best, worst, and average case

Best case Minmal complexity for the most favorable input.

Worst case Maximal complexity for the least favorable input.

Average case Typical complexity (for some definition of typical input).

Algorithmic complexity

- Classify algorithms according to their performance
- Time function $T(n)$ measures runtime
- Big-O notation expresses *runtime complexity*
- Considers only the highest order term of $T(n)$
- Upper bound on growth rate

Definition

The computational complexity is

$$T(n) \in O(f(n))$$

if and only if there exists a constant c such that $T(n) < c\,f(n)$ for all $n > n_0$
We say $f(n)$ is an asymptotic upper bound for $T(n)$.

Sipser, Introduction to the theory of computation, 2006

Simplification

The term in $T(n)$ that grows most quickly will eventually dominate all other terms.

We can make the following simplifications

- Only keep the fastest growing term.
- Omit any multiplicative constants.

For example $T(n) = 4x^3 + 5x^2 + 10$ can be simplified to $T(n) \in O(n^3)$.

Example

An algorithm with time complexity

$$T(n) = 1\,000\,000n + n^2$$

is still $O(n^2)$ because for $n > 1\,000\,000$ the term n^2 is largest.

Algorithmic complexity

Constant time, $O(1)$ Same amount of computation regardless of input size
Example: Access a specific element in a list

Logarithmic time, $O(\log n)$ Computation proportional to logarithm of input size
Example: Binary search (find element in sorted list)

Linear time, $O(n)$ Computation proportional to the input size
Example: Find minimum element in a list

Quadratic time, $O(n^2)$ Computation proportional to the square of the input size
Example: Selection sort

Factorial time, $O(n!)$ Computation proportional to the factorial of the input size
Example: Tabulate all permutations of a list

Example lists

Unsorted	99	83	125	12	5	256	31	192
Sorted	5	12	31	83	99	125	192	256

A simple sorting algorithm

Algorithm

1. Find the smallest number on the list
2. Remove the smallest number from the list and append it to the list of sorted numbers
3. Repeat the above steps until the list is empty

A simple sorting algorithm

Algorithm

1. Find the smallest number on the list
2. Remove the smallest number from the list and append it to the list of sorted numbers
3. Repeat the above steps until the list is empty

Complexity

(Number of comparisons needed to sort a list of n numbers)

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

$$T(n) \in O(n^2)$$

Exercise: Merge sort

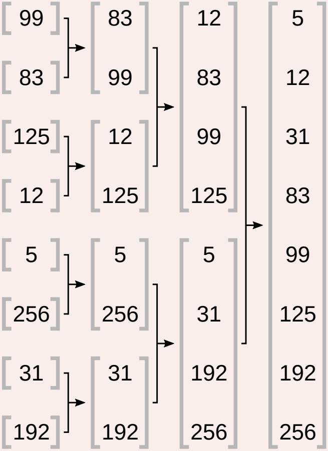
Algorithm

At all times, maintain a set of sorted sublists
Initially each element is a sorted sublist

1. Merge each pair of sublists to form a new sorted sublist
2. Repeat until all sublists have been merged

Question

- How many operations (comparisons) are required (in the worst case) to sort a list of 8 items?
- What is the algorithmic complexity of merge sort?
  Assume for simplicity that the number of elements is a power of two, $n = 2^\ell$.



Exercise: Merge sort

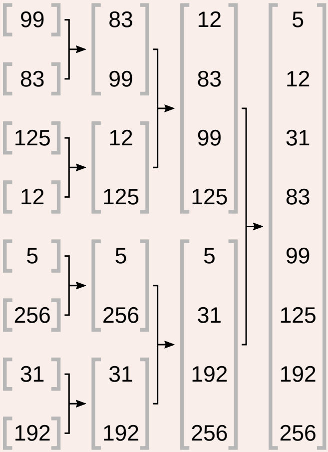
Algorithm

At all times, maintain a set of sorted sublists
Initially each element is a sorted sublist

1. Merge each pair of sublists to form a new sorted sublist
2. Repeat until all sublists have been merged

Question

- How many operations (comparisons) are required (in the worst case) to sort a list of 8 items?
- What is the algorithmic complexity of merge sort?
  Assume for simplicity that the number of elements is a power of two, $n = 2^\ell$.



Solution

$$T(n) = \frac{n}{2}\cdot 1 + \frac{n}{4}\cdot 3 + \frac{n}{8}\cdot 7 + \cdots = n(\frac{1}{2} + \frac{3}{4} + \frac{7}{8} + \cdots) < n\ell \quad T(8) = 17$$

$$n = 2^\ell \Leftrightarrow \ell = \log_2(n), \quad T(n) \in O(n \log n)$$

Divide and conquer / recursion

Divide an conquer

Divide and conquer

1. Divide the problem into smaller sub-problem
2. Solve sub-problems (recursively) until solved
3. Combine the sub-problems to get the solution to the full problem

Find a peak

Input A sequence of n numbers, $x_1, x_2, \ldots, x_n$.

Objective Find a peak, defined as a position $i \in (1, \ldots, n)$ in the sequence such that

$$x_{i-1} \leq x_i \geq x_{i+1}$$

or at the edges a peak is defined as

$$x_1 \geq x_2 \quad \text{and/or} \quad x_{n-1} \leq x_n.$$

Based on a problem discussed in MIT 6.006 Introduction to Algorithms

Find a peak

Input A sequence of n numbers, $x_1, x_2, \ldots, x_n$.

Objective Find a peak, defined as a position $i \in (1, \ldots, n)$ in the sequence such that

$$x_{i-1} \leq x_i \geq x_{i+1}$$

or at the edges a peak is defined as

$$x_1 \geq x_2 \quad \text{and/or} \quad x_{n-1} \leq x_n.$$

1. Look at the “middle” number at position $[n/2]$

Based on a problem discussed in MIT 6.006 Introduction to Algorithms

Find a peak

Input A sequence of n numbers, $x_1, x_2, \ldots, x_n$.

Objective Find a peak, defined as a position $i \in (1, \ldots, n)$ in the sequence such that

$$x_{i-1} \leq x_i \geq x_{i+1}$$

or at the edges a peak is defined as

$$x_1 \geq x_2 \quad \text{and/or} \quad x_{n-1} \leq x_n.$$

1. Look at the "middle" number at position $[n/2]$
2. If $x_{n/2-1} > x_{n/2}$ then consider the sequence to the left, $x_1, \ldots, x_{n/2-1}$

Based on a problem discussed in MIT 6.006 Introduction to Algorithms

Find a peak

Input A sequence of n numbers, $x_1, x_2, \ldots, x_n$.

Objective Find a peak, defined as a position $i \in (1, \ldots, n)$ in the sequence such that

$$x_{i-1} \leq x_i \geq x_{i+1}$$

or at the edges a peak is defined as

$$x_1 \geq x_2 \quad \text{and/or} \quad x_{n-1} \leq x_n.$$

1. Look at the "middle" number at position $[n/2]$
2. If $x_{n/2-1} > x_{n/2}$ then consider the sequence to the left, $x_1, \ldots, x_{n/2-1}$
3. Else, if $x_{n/2} < x_{n/2+1}$ then consider the sequence to the right, $x_{n/2+1}, \ldots, x_n$.

Based on a problem discussed in MIT 6.006 Introduction to Algorithms

Find a peak

Input A sequence of n numbers, $x_1, x_2, \ldots, x_n$.

Objective Find a peak, defined as a position $i \in (1, \ldots, n)$ in the sequence such that

$$x_{i-1} \leq x_i \geq x_{i+1}$$

or at the edges a peak is defined as

$$x_1 \geq x_2 \quad \text{and/or} \quad x_{n-1} \leq x_n.$$

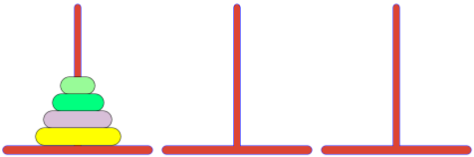
1. Look at the "middle" number at position $[n/2]$
2. If $x_{n/2-1} > x_{n/2}$ then consider the sequence to the left, $x_1, \ldots, x_{n/2-1}$
3. Else, if $x_{n/2} < x_{n/2+1}$ then consider the sequence to the right, $x_{n/2+1}, \ldots, x_n$.
4. Else, $x_{n/2}$ is a peak.

Based on a problem discussed in MIT 6.006 Introduction to Algorithms

Recursion: Tower of Hanoi

```
def thanoi(pieces, movefrom=1, moveto=2, other=3):
    if pieces == 1:
        print(f'Move ring from {movefrom} to {moveto}')
    else:
        thanoi(pieces-1, movefrom, other, moveto)
        thanoi(1, movefrom, moveto, other)
        thanoi(pieces-1, other, moveto, movefrom)

thanoi(4)
```



Output

Move ring from 1 to 3
Move ring from 1 to 2
Move ring from 3 to 2
Move ring from 1 to 3
Move ring from 2 to 1
Move ring from 2 to 3
Move ring from 1 to 3
Move ring from 1 to 2
Move ring from 3 to 2
Move ring from 3 to 1
Move ring from 2 to 1
Move ring from 3 to 2
Move ring from 1 to 3
Move ring from 1 to 2
Move ring from 3 to 2

Code from: cs4fn Puzzle Book Issue 1. Image from: https://www.mathsisfun.com/games/towerofhanoi.html

Tower of Hanoi

Problem

- What is the time complexity $T(n)$ for the solution to the Tower of Hanoi problem? (The number of moves as a function of the number of rings n)

Hint: With one ring it takes one move, $T(1) = 1$. Two rings requires three moves, $T(2) = 3$. To solve an $n+1$-rings problem we solve an n-rings problem, move 1 ring, and solve an n-rings problem again, so we have $T(n+1) = 2 \cdot T(n) + 1$. You can use this to find $T(n)$ for $n = 1, 2, 3, 4 \dots$ and see if you can spot the pattern.

Tower of Hanoi

Problem

- What is the time complexity $T(n)$ for the solution to the Tower of Hanoi problem? (The number of moves as a function of the number of rings n)

Hint: With one ring it takes one move, $T(1) = 1$. Two rings requires three moves, $T(2) = 3$. To solve an $n+1$-rings problem we solve an n-rings problem, move 1 ring, and solve an n-rings problem again, so we have $T(n+1) = 2 \cdot T(n) + 1$. You can use this to find $T(n)$ for $n = 1, 2, 3, 4 \ldots$ and see if you can spot the pattern.

Solution

Using the recursion we get

n	1	2	3	4	5	6	7	8
$T(n)$	1	3	7	15	31	63	127	255

From this we can spot the pattern $\underline{T(n) = 2^n - 1}$

Software tools

LaTeX

LaTeXis a markup language for writing documents. Once you get used to it, it is way better than everything else.

- Install on your own computer or *or use online at `overleaf.com`*
- Get started with the tutrial at `latex-tutorial.com/tutorials`

A bit of a learning curve, so you might as well get started.

Git

Git is a distributed system for version control, especially useful for software develpment in a team.

- Use a commercial system like github or *DTU Compute's free* *`lab.compute.dtu.dk`*
- Worksheet "git_tutorial.pdf" on DTU Learn.
- A good free book/reference at `git-scm.com/book/en/v2`

A bit of a learning curve, so you might as well get started.

Tasks

Tasks for today

Tasks today

- Continue work on lab report
- Start learning about on `latex-tutorial.com/tutorials/` and `overleaf.com`
- Start learning about Git with the worksheet "git_tutorial.pdf" on DTU Learn.

Today's feedback group

- Viet Hoang Nguyen
- Lukas Peter Dyhr
- Philip Kierkegaard
- Ali Mohammed Fathi Afif

Lab report hand in

- Lab 1: Image recognition (Deadline: Thursday 14 September 20:00)

Next time

- Read the notes "Symbolic AI" + Solve all problems