Session 40: Introduction to Complexity

- Computational Complexity
- Time and Space Complexity
- Worst and Average Case Complexity

The Complexity of Algorithms

Given an algorithm, how efficient is this algorithm for solving a problem given an input of a particular size (**computational complexity**)?

- How much time does this algorithm use to solve the problem for an input of a given size (**time complexity**)?
- How much computer memory does this algorithm use to solve the problem for an input of a given size (**space complexity**)?

The Complexity of Algorithms

Given an algorithm, how efficient is this algorithm for solving a problem given an input of a particular size (**computational complexity**)?

- How much time does this algorithm use to solve the problem for an input of a given size (**time complexity**)?
- How much computer memory does this algorithm use to solve the problem for an input of a given size (**space complexity**)?

Understanding complexity is important

- To understand whether it is practical to use an algorithm for inputs of a particular size
- To compare the efficiency of different algorithms for solving the same problem.

Time Complexity

We will focus on **time complexity**

- If the algorithm is sequential, all operations are executed in sequential order.
- Then time complexity corresponds to the **number of operations** performed.
- We will use big-*O* and big-Theta notation to describe the time complexity.

We **ignore implementation details** (including the data structures used and both the hardware and software platforms) because it is extremely complicated to consider them.

Determining Time Complexity

We determine the number of basic operations

- E.g., comparisons and arithmetic operations (addition, multiplication, etc.).
- We assume all operations use a constant time.
- The time for the basic operations can be different from one computer to the next.
- We ignore minor details, such as the “house keeping” aspects of the algorithm

Worst-Case Time Complexity

We will focus on the **worst-case time** complexity of an algorithm

- An upper bound on the number of operations an algorithm uses to solve a problem with input of a particular size.
- It is usually much more difficult to determine the **average case time complexity** of an algorithm, the average number of operations an algorithm uses to solve a problem over all inputs of a particular size.

Complexity Analysis of Algorithms

Example: Worst case time complexity of the algorithm for finding the maximum element in a finite sequence.

procedure *max*(a_1, a_2, …., a_n: integers)

max := a_1

for *i* := 2 to *n*

    if *max* < a_i then *max* := a_i

return *max*

loop is executed n-1 times

In each loop (while $i \leq n$)
- a test $i \leq n$ is performed to see whether the for loop ends
- a test $max < a_i$ is performed

In the last step (when $i = n+1$)
- a test $i \leq n$ is performed and the loop is ended

So in total $2(n-1)+1 = 2n-1$ comparisons are done.

The complexity is $\Theta(n)$

Summary

- Computational Complexity
  - Abstracts from implementation details
- Time and Space Complexity
  - Time complexity corresponds to number of operations
- Worst and Average Case Complexity
  - Worst case complexity in general easier to analyse