# Lecture 1: The Role of Algorithms in Computer Science

- Algorithms
  - Informally: any well defined computational procedure
    - Input values
    - Output values
  - A computational sequence of steps that transform an input into an output
    - Algorithm describes such a transformation
  - Tool to solve a well specified computational problem
- Sorting problem
- Input: sequence of numbers $(a_1, a_2, \ldots, a_n)$
- Output: permutation (reordering), $(a'_1, a'_2, \ldots, a'_n)$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.
- For sequence (31, 41, 59, 26, 41, 58), the output is: (26, 31, 41, 41, 58, 59)
- The input sequence is known as instance
- An instance
  - Satisfies the problem restrictions
  - Necessary to compute the problem solution
- Sorting is a fundamental operation in computer science
  - Used as an intermediate step
  - There are many good sorting algorithms
  - Which one is the best for a given task?
    - It depends?
    - Number of elements to sort?
    - Are some of the elements already sorted?
    - Are there restrictions in the values to sort?
    - What kind of storing are we using (main memory, hard drives, tapes)?
  - We'll work with different sorting algorithms
- Correct, an algorithm is correct if and only if for each input instance, it halts when it obtains the correct output
- We say that a correct algorithm solves the given computational problem
- An incorrect algorithm
  - Might not halt at all for some instances
  - Might halt with an incorrect answer
- Incorrect algorithms might be useful
  - If we can control their error rate
- We'll work with correct algorithms
- Algorithm specification
- Can be specified in English
- As a computer program
- As pseudocode
- There's only one restriction
  - Specification must provide a precise description of the computational procedure to follow
- Common characteristics of algorithms
  - They have many candidate solutions but most of them are useless for us, finding the one we need might be difficult
  - They have practical applications, as in the shortest path
- Data structures
  - A way to store and organize data to facilitate access and modification
  - A data structure doesn't work well for every purpose, they have strengths and limitations
- Technique
  - It might be that the algorithm we need isn't published yet
  - We need a technique to design and analyze algorithms
  - So we can develop new ones
  - Show they produce the right answer
  - Understand their efficiency
- Hard Problems
- There exist some problems for which an efficient solution isn't known
  - NP-complete algorithms
  - No efficient algorithm has been found for NP-complete problems
    - We don't know either whether there exist or not efficient algorithms for NP-complete problems
  - Property of NP-complete algorithms

        * If there exists an efficient algorithm for any of them, then there exist efficient algorithms for all of them
    - Several NP-complete problems are similar (not identical) to problems for which we know there exist efficient algorithms
        * A subtle change in the problem definition causes a big change in the efficiency of the best known algorithm
    - It's good to know about NP-complete problems
        * They appear frequently in real world applications
        * We may invest too much time in them
        * If we show that the algorithm is NP-complete
        * We could invest all that time in developing an algorithm that gets a good answer (not the best possible one)
- Parallelism
    - For many years single processor clock speeds increased at steady rate
    - But there are physical limitations on ever increasing clock speeds
    - Chips have been designed to to contain more than one core
    - Multicore computers are a type of parallel computer
    - Multithreaded algorithms take advantage of multiple cores
- Algorithms as a Technology
    - What would happen if computers were infinitely fast and memory were free?
    - Oh!, but since they aren't infinitely fast nor memory is free
        * Computing time and memory space are limited resources
        * We must be smart to use them
        * Need efficient algorithms in time and space
- Efficiency
    - Algorithms solving the same problem frequently have different efficiency
        * More significant than differences due to HW or SW
- Efficiency, sorting algorithms
    - InsertionSort $c_1 n^2$ to sort $n$ elements, $c_1$ is a constant independent from $n$
    - MergeSort $c_2 n \lg n$, where $\lg n$ is $\log_2 n$ and $c_2$ is another constant independent of $n$
    - $c_1 < c_2$
    - constant factors are not as important as $n$
    - $n$ much higher than $\lg n$
    - InsertionSort faster than MergeSort for small inputs
    - There is a point (in size of $n$) in which MergeSort gets faster than InsertionSort
    - Efficiency, example
        * Computer A is faster than Computer B
        * A, 100 millions of instructions per second $\rightarrow 10^8$ inst/sec
        * B, 1 million of instructions per second $\rightarrow 10^6$ inst/sec
        * We implement InsertionSort in A, $2n^2$ ($c_1 = 2$)
        * We implement MergeSort in B, $50 n \lg n$ ($c_2 = 50$)
        * Task of sorting 1,000,000 of numbers, ($n = 10^6$)
        * $A \rightarrow \frac{2(10^6)^2 \text{inst}}{10^8 \text{inst/sec}} = 20{,}000 \text{ seconds} = 5.56$ hours
        * $B \rightarrow \frac{50 \cdot 10^6 \lg 10^6 \text{inst}}{10^6 \text{inst/sec}} = 1{,}000$ seconds $= 16.67$ minutes
        * With 10,000,000 numbers
            · B (20 min) is 20 times faster than A (2.3 days) to sort the numbers
            · The advantage of the efficiency of MergeSort
        * Algorithms and other technologies
        * Predict the resources required by the algorithm Memory, computing time
        * Consider the model of the implementation technology 1 processor RAM memory One instruction after the other, no concurrent operations
        * We need mathematical tools to describe/represent this!