Chapter 1: The Role of Algorithms in Computing

**1.1 Algorithms**

An algorithm is a computational procedure that is used for solving specific computational problem in a specific runtime. As an [effective method](https://en.wikipedia.org/wiki/Effective_method), an algorithm can be expressed within a finite amount of space and time, and in a well-defined formal language for calculating a [function](https://en.wikipedia.org/wiki/Function_(mathematics)). An algorithm is said to be correct if, for every input instance, it halts with the correct output.

**Data structures**

A data structure is a way to store and organize data in order to facilitate access and modifications.

**Technique**

Necessary techniques are applied in an algorithm in such a way so that the efficiency of the algorithms is the most and the algorithm gives the appropriate result or correct result.

**Hard Problem**

NP (nondeterministic polynomial time)-complete is such problems that the efficient solution of these problem isn’t known. For example: “traveling-salesman problem”.

Parallelism:

To elicit the best performance from multicore computers, we need to design algorithms with parallelism in mind. “Multithreaded” algorithms are used for obtaining parallelism.

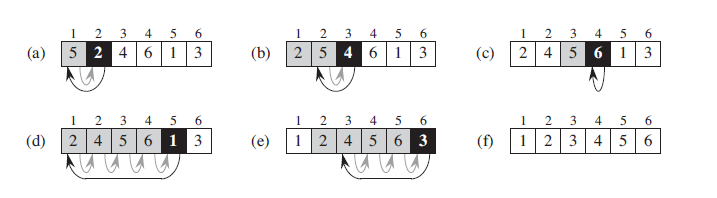
**1.2 Algorithms as a technology**

Efficiency:

The efficiency of an algorithm determines how much time it will take to be executed. For example, Insertion sort takes n2 time to sort n items, while merge sort takes nlog(n) times to sort n items. It means that merge sort is faster than insertion sort. The efficiency also depends upon the computer architecture. If we want to sort 10 million numbers in two different computers having different clock speed. Let’s say, computer A can execute 10 billion instructions per second while computer B execute 10 million of instructions per second. So, computer A is 1000 times faster than computer B. Now for sorting 10 million numbers in computer A with Insertion sort algorithm, 20000 seconds will take. But in computer B, 1163 seconds will be needed to sort 10 million numbers applying merge sort algorithm. That means computer B runs 17 times faster than computer A.

Chapter 2: Getting Started

2.1 Insertion sort



Insertion sort is a simple sorting algorithm that builds the final sorted array in n2 time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. It works the way we sort playing cards in our hands.

2.2 Analyzing algorithms

It’s used for identifying the efficient algorithm of a problem. Analyzing algorithms occasionally determines:

1. Executational time.

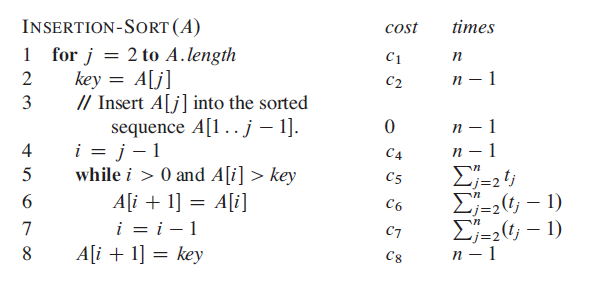
The RAM model contains instructions commonly found in real computers: *arithmetic* (such as add, subtract, multiply, divide, remainder, floor, ceiling), *data movement* (load, store, copy), and *control* (conditional and unconditional branch, subroutine call and return). Each such instruction takes a constant amount of time.

1. Memory

The different types of data types allocates different amount of memory.

1. Communication band-width
2. Computer Hardware

Analysis of insertion sort

The running time of an algorithm on a particular input is the number of primitive operations or “steps” executed. Here, supposed that a constant amount of time is required to execute each line of our pseudocode.

For the best case, the total runtime of the algorithm forms a linear function. And in worst case, it forms a quadratic function. So, here the runtime is n2.

Order of growth

We can simplify the running time equation and it is called order of growth or rate of growth. For the equation an2+bn+c, we can ignore the lower order terms and also the co-efficient of leading terms. And we write that insertion sort has a worst-case running time of, Θ(n2) (pronounced “theta of n-squared”).

Chapter 3: Growth of Functions

**Asymptotic notation:** Asymptotic analysis of an algorithm refers to defining the mathematical

boundation/framing of its run-time performance. Using asymptotic analysis, we can very well

conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to

work in a constant time. Other than the "input" all other factors are considered

constant. Asymptotic analysis refers to computing the running time of any operation in

mathematical units of computation. For example, the running time of one operation is computed

as *f* (n) and may be for another operation it is computed as *g* (n 2 ). This means the first operation

running time will increase linearly with the increase in n and the running time of the second

operation will increase exponentially when n increases. Similarly, the running time of both

operations will be nearly the same if n is significantly small.

**Θ-notation:**

The Θ-notation asymptotically bounds a function from above and below. When we have

only an asymptotic tight bound, we use Θ-notation.

**O-notation:**

The Θ-notation asymptotically bounds a function from above and below. When we have

only an asymptotic upper bound, we use O-notation. We use O-notation to give an upper

bound on a function, to within a constant factor.

**Ω-notation:**

Just as O-notation provides an asymptotic upper bound on a function, Ω-notation

provides an asymptotic lower bound.

**o-notation:**

The asymptotic upper bound provided by O-notation may or may not be asymptotically

tight. We use o-notation to denote an upper bound that is not asymptotically tight.

**ω-notation:**

By analogy, ω-notation is to Ω-notation as o-notation is to O-notation. We use ω-notation

to denote a lower bound that is not asymptotically tight.

**Standard notations and common functions:**

**Monotonicity:**