**The Role of Algorithm in Computing**

**Chapter 1**

* 1. *Algorithms*

Informally, an algorithm is any well-defined computational procedure that takes some value or set of values, as input and produces some value, or set of values, as output.

An algorithm can be specified in English, as a computer program, or even as a hardware design.

*Data structures*

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

*Parallelism*

Dataparallelism is parallelization across multiple processors in parallel computing environments. It focuses on distributing the data across different nodes, which operate on the data in parallel. It can be applied on regular data structures like arrays and matrices by working on each element in parallel. It contrasts to task parallelism as another form of parallelism.

* 1. **Algorithm as a technology**

*Efficiency*

Efficiency is an important think of algorithm. Different algorithms devised to solve the same problem often differ dramatically in their efficiency. These differences can be much more significant than differences due to hardware and software.

The algorithm is much more efficient than another algorithm, computer takes small time to calculate the output.

**Chapter 2**

2.1 Insertion sort

The algorithm of insertion sort

1. for j = 2 to A.length
2. key = A[j]
3. i = j – 1
4. while i>0 and A[i]>key
5. A[i+1] = A[i]
6. i = i – 1
7. A[i+1] = key

Parts of Insertion algorithm is

1. Initialization
2. Maintenance
3. Termination

2.3.1 The divide-and-conquer approach

Many useful algorithms are recursive in structure, to solve a given problem, they call themselves recursively one or more times to deal with closely related sub-problems. These algorithms typically follow a divide and conquer approach.

**Divide** the problem into a number of sub-problems that are smaller instances of the same problem

**Conquer** the sub-problems by them recursively. If the sub-problem size are small enough, however, just solve the sub-problems in a straightforward manner.

**Combine** the solutions to the sub-problems into the solution for the original problem.

Chapter 3

**Growth of Function**

3.1 *Asymptotic*

The notation we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domains are the set of natural numbers N = {0,1,2,……}

Such functions are convenient for describing the worst-case running-time function T(n), which usually is defined only

O- notation

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

**Standard notation and common functions**

*Monotonicity*

A monotonic function (or monotone function) is a function  between ordered sets that preserves or reverses the given order. This concept first arose in calculus, and was later generalized to the more abstract setting of order theory.

*Floors and ceilings*:

The floor function is the function that takes as input a real number{\displaystyle x} x and gives as output the greatest integer less than or equal to x{\displaystyle x}, denoted floor(x)=└x┘{\displaystyle \operatorname {floor} (x)=\lfloor x\rfloor }. Similarly, the ceiling function maps x{\displaystyle x} to the least integer greater than or equal to x{\displaystyle x}, denoted ceil(x)=┌x┐{\displaystyle \operatorname {ceil} (x)=\lceil x\rceil }.

*Modular arithmetic:*

Modular arithmetic is a system of arithmetic for integers, where numbers "wrap around" when reaching a certain value, called the modulus.

*Polynomials*

A polynomial is an expression consisting of variables and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponents of variables.

*Logarithms*

The logarithm is the inverse function to exponentiation. That means the logarithm of a given number x is the exponent to which another fixed number, the base b, must be raised, to produce that number x.

*Factorials:*

The factorial of a positive integer n, denoted by n!, is the product of all positive integers less than or equal to n.

*Functional iteration:*

An iterated function is a function X → X which is obtained by composing another function f: X → X with itself a certain number of times. The process of repeatedly applying the same function is called iteration.