Algorithms and Data Structure

Algorithms Analysis

What is an Algorithms?

- An *algorithm* is a well-defined procedure that consist finite sequence of precise steps for solving a computational problem.
- All algorithms must satisfy the following criteria:
  - I. **Input:** Zero or more quantity are externally supplied.
  - II. **Output**: At least one quantity must be produced
  - III. **Definiteness**: Each instruction must be clear & unambiguous
  - IV. **Finiteness**: the Algorithm must be terminated after a finite number of steps
- Study of computer Algorithm include:
  - I. Algorithm Design
    - use many design techniques to design new & useful algorithm
  - II. Algorithm Validation
    - check correctness of answer for all possible legal inputs
  - III. Algorithm Analysis
    - determine how much computing time(CPU) & storage capacity(RAM) is required by an algorithm
    - compare algorithms mainly in terms of running time but also in terms of other factors (e.g., memory requirements, programmer's effort etc.)

Algorithm Specification

- We can describe an algorithm in many ways by using:
  - ❖ A Natural language like English
  - ❖ Some graphical representation like Flowchart
  - ❖ Pseudocode that resembles like C/C++ programming Language

Example:--

```
Algorithm Sum(a, n)
// a is array of size n
{
    s=0;
    for(i=0 to n-1)
      s = s + a[i];
    return s;
}
```

General form of Algorithm Specification:

```
Algorithm AlgoName(Parameter List)
// …write comment here …
{
     statement1,
    statement2,
…………………………
}
```

Form of an Algorithm Design

- Broadly, there are two basic forms of designing an Algorithm for a problem:
  - I. Iterative Algorithm
  - II. Recursive Algorithm

- Recursive Algorithms
  - I. An algorithm is said to be recursive if the same algorithm is invoked in the body
  - II. An algorithm that calls itself is direct recursive
  - III. An algorithm A is indirect recursive if it calls another algorithm which in turn calls A.

- Although most of problems can be solved by devising an algorithm in either of forms, recursive mechanisms are extremely powerful

  E.g Computing factorial, generating Fibonacci series, etc

Example of Algorithm Specification

Example: Iterative Algorithm for Sum

```
Algorithm Sum(a, n)
// a is array of size n
{
  s=0;
  for(i=0 to n-1)
     s = s + a[i];
  return s;
 }
```

Example: Recursive Recursive for Sum

```
Algorithm Sum(a, n)
// a is array of size n
{
      if(n<=0)
         return 0;
      else
         return Sum(a, n-1) + a[n];
}
```

Algorithm Analysis

- Algorithm analysis can be carried out to measure its performance in term of space/time complexity.
- Space Complexity of an algorithm is the amount of memory space it needs to run to completion.
  - space for the code, simple variable, fixed size, constant and so on…
  - Space for instance characteristics (require during run time)
  - Other factor that affect time complexity(programmer skill, etc)
  - Input quantity
- Time Complexity of an algorithm is the amount of computer time it needs to run to completion
  - compile time (does not depend on instance characteristic & estimated only once)
  - run time
  - Other factor that affect time complexity(programming language-compiler, programmer skill, computer system, etc)
  - Input quantity

Algorithm Analysis

- What do we mean by running time analysis?
  - Determine how running time increases as the size of the problem increases.
- Input size (number of elements in the input)
  - size of an array
  - polynomial degree
  - Number of elements in a matrix
  - Number of bits in the binary representation of the input
  - vertices and edges in a graph

Types of Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are
- Best case
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

$$\boxed{\textit{Lower Bound} \le \textit{Running Time} \le \textit{Upper Bound}}$$

- Average case
  - Provides a prediction about the running time
  - Assumes that the input is random