Readings:


Objectives:

1. Understand algorithms’ role as a technology
2. To learn Design and Analysis of algorithms
3. To understand the concept “Data Structure” through Arrays

Concepts:

1. Introduction
2. The Role of Algorithms in Computing
3. Growth of Functions and Asymptotic Notations
4. Elementary Data Structures – Arrays
5. Overview of Lab2

Outline:

1. Introduction
   a. Getting Started
   b. Foundation of Algorithms

2. The Role of Algorithms in Computing
   a. Algorithms
   b. Algorithms as a technology

3. Growth of Functions
   a. Asymptotic Notation
   b. Standard notations and common functions

4. Elementary Data Structures- Working with Arrays

5. Lab2

Reference:

1. Introduction

   a. Getting Started
      - Algorithms: It can be defined as a well defined computational procedure that solves a problem

   b. Foundation of Algorithms
      - Takes input(s) and produces output(s)
      - A sequence of steps that transform the input into the output

2. The Role of Algorithms in Computing

   a. Algorithms
      - An algorithm is essence of any computer program

   b. Algorithms as a technology
      Role of algorithms relative to other technologies in computing is more important.
      A well-written algorithm can help in design of a program that excels in:
      - Correctness
      - Amount of work done (efficiency)
      - Amount of space used
      - Simplicity, clarity
      - Optimality

3. Growth of Functions

   - Asymptotic Notation
     - Introduction:
       - Asymptotic means a line that tends to converge to a curve, which may or may not eventually touch the curve.
       - It's a line that stays within bounds.
       - Asymptotic notation is a shorthand way to write down and talk about 'fastest possible' and 'slowest possible' running times for an algorithm, using high and low bounds on speed.

   - Big O Notation: Big-O is the formal method of expressing the upper bound of an algorithm's running time.
     - Definition: More formally, for non-negative functions, \( f(n) \) and \( g(n) \), if there exists an integer \( n_0 \) and a constant \( c > 0 \) such that for all integers \( n > n_0 \), \( f(n) \leq cg(n) \), then \( f(n) \) is Big O of \( g(n) \). This is denoted as \( f(n) = O(g(n)) \).
     - Theory Examples: Say that \( f(n) = 2n + 8 \), and \( g(n) = n^2 \). Can we find a constant \( c \), so that \( 2n + 8 \leq cn^2 \)?

   - Little-O Notation: For non-negative functions, \( f(n) \) and \( g(n) \), \( f(n) \) is little o of \( g(n) \) if and only if \( f(n) = O(g(n)) \), but \( f(n) \neq \Theta(g(n)) \). This is denoted as \( f(n) = o(g(n)) \). This represents a loose bounding version of Big O.
Big Omega Notation
Definition: For non-negative functions, $f(n)$ and $g(n)$, if there exists an integer $n_0$ and a constant $c > 0$ such that for all integers $n > n_0$, $f(n) \geq cg(n)$, then $f(n)$ is omega of $g(n)$. This is denoted as "$f(n) = \Omega(g(n))$".

Little Omega Notation
Definition: For non-negative functions, $f(n)$ and $g(n)$, $f(n)$ is little omega of $g(n)$ if and only if $f(n) = \Omega(g(n))$, but $f(n) \neq \Theta(g(n))$. This is denoted as "$f(n) = \omega(g(n))$".

Theta Notation
Definition: For non-negative functions, $f(n)$ and $g(n)$, $f(n)$ is theta of $g(n)$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. This is denoted as "$f(n) = \Theta(g(n))$".

How asymptotic notation relates to analyzing complexity
A trade off between time and space is noticed in algorithms. Asymptotic notation empowers you to make that trade off. A few examples of asymptotic notation

4. Elementary Data Structures- Working with Arrays

- What is an Array: An array is a particular method of storing elements of indexed data. Elements of data are stored sequentially in blocks within the array. Each element is referenced by an index, or subscript.
- Initializing an Array: The initializer for an array is a comma-separated list of constant expressions enclosed in braces ({}). The initializer is preceded by an equal sign (=).
  - static int number[3] = { 5, 7, 2 };
- Array Index Out of Bounds
- Passing an Array to a Function: // one dimension with pointer
  - void function(int *a);
  - int array[10];
  - function(array);

5. Lab 2

- Implementation of Arrays using C++
- Passing Array values to a function
- Using Arrays with Enumeration Types