Chapter 2

Introduction to Data Structure

Data Structures: A Programming Approach with C, PHI,
Dharmender Singh Kushwaha & A.K.Misra

Abstract Data Type

- The data with its type and set of operation on it (for creation, handling, manipulation and retrieving it) is referred to as Abstract Data Type (ADT). The primitive data types are:
  - Integers : byte, sbyte, short, ushort, int, long
  - Floating Point : float, double
  - Boolean : bool
  - Characters : char
  - Strings : string

Contd…

- While writing a program, we need to implement these abstract data types.
- This implementation of particular abstract data type is called a Data Structure.
- Thus the term data structure refers to a scheme for organizing related pieces of information which also should help us in reducing the complexity and increasing the efficiency of an algorithm.

Linear and nonlinear data structures

- A data structure is said to be linear if every element is placed or attached to its previous and next item.
  - Eg. array, linked list, stack or queue
- It is non-linear if every element is attached to many other elements in various ways to reflect a particular relationships.
  - Eg. trees, graph or heap
- In linear data structure data items are arranged in a linear sequence while in non-linear data structures, data items are not in a sequence.

Algorithm

- Logical sequence of discrete steps that describes a complete solution to a given problem in a finite amount of time is an algorithm.
- The quality of the code, number of errors during compilation and the amount of resources required to compile and execute the program is dependent on the quality of the algorithm.
- Hence we should be in a position to identify a good or a bad algorithm.

Analyzing & Comparing Algorithms

- We all write somewhat different algorithms which correctly solve the same problem. How can we choose the best among them?
- An important question often asked is “How efficient is an algorithm or piece of code? Efficiency covers lots of issues like:
  - CPU time usage
  - memory usage
  - I/O or
  - network usage

Analyzing & Comparing Algorithms

- In general, when we talk of performance, we are interested in how much CPU time or memory is actually used when a program is under execution.
- This depends on the machine, compiler, etc. as well as the style of coding.
- When we talk of complexity, we are interested in how does the resource requirements of a program or algorithm scale, i.e., what happens as the size of the problem being solved gets larger.
- Thus complexity affects performance.

Analyzing & Comparing Algorithms

- The data structures used for a particular algorithm can greatly affect its performance.
- If we have ‘n’ array elements, we might have to perform ‘n’ number of checks in order to find any element.
- That is, the time it takes to search an array is linearly proportional to the number of elements in the array.
- With binary search trees, the time required is sub-linear.
- Thus when we are searching large amounts of data, the data structure chosen can make a difference in the application's performance.

Analyzing & Comparing Algorithms

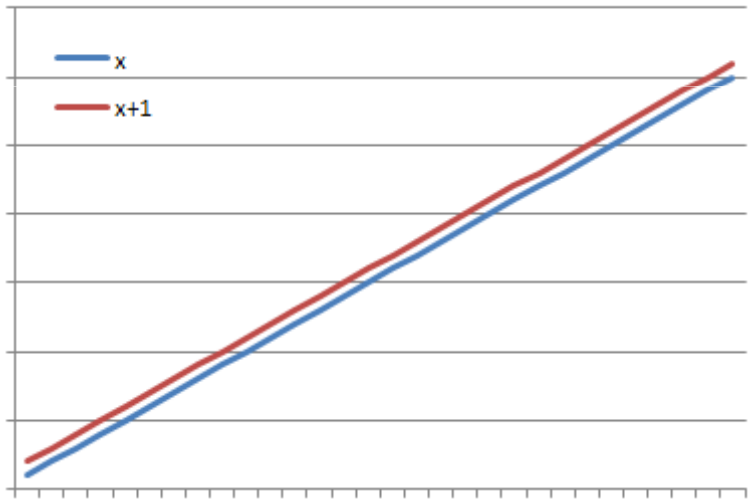
- When we consider the complexity of a program, we don't really care about the exact number of operations that have to be carried out.
- Instead we are interested in how the number of operations relates to the problem size.
- If the problem size doubles, does the number of operations stay the same, double or increase in some other manner.

How asymptotic notation relates to analyzing complexity

- Generally, there is always a tradeoff between time and space for any algorithm.
- This is important in data structures because we want a structure that behaves efficiently as we increase the amount of data it handles.
- We should understand that algorithms that are efficient with large amounts of data are not always simple and efficient for small amounts of data.

What is Asymptotic?

- Asymptotic is a line that continually approaches a given curve but does not meet it at any finite distance.
- If x is asymptotic with x + 1, we have the graph as shown here



What is Asymptotic?

- Let f(x) and g(x) be functions from the set of real numbers to the set of real numbers.
- We say that f and g are *asymptotic* and write f(x) ~ g(x) if:
  - f(x) / g(x) = *c* (constant)

Asymptotic Notation

- It provides information about the relative rates of growth of a pair of functions of a single integer or real variable. It hides other details, such as:
  - behaviour on small inputs because results are more meaningful when inputs are extremely large.
  - multiplicative constants and lower-order terms which can be implementation or platform dependent.

Asymptotic Notation

- Big O, Omega and Theta are the three asymptotic notations that are of particular use to us when deciding about an algorithm being good or bad for a given data set.

Contd…

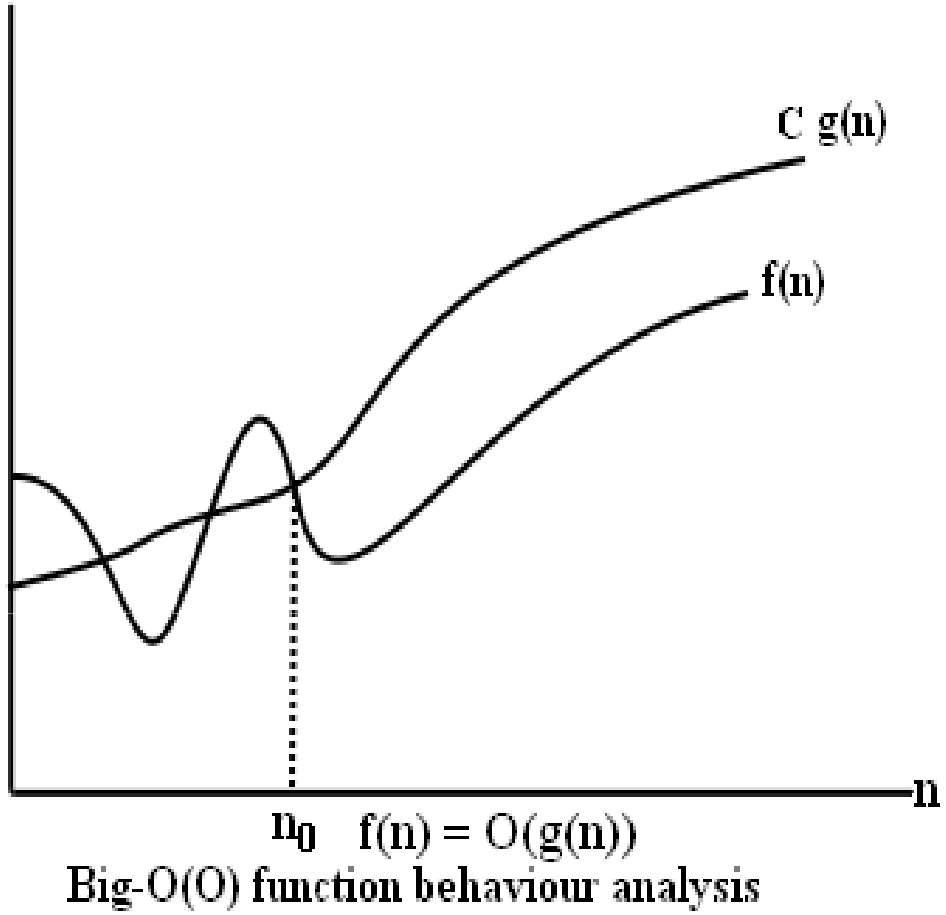
- The algorithm performance is obtained by totaling the number of occurrence of each operation when running the algorithm.
- It assumes that ‘n’ measures the size of the problem say reading n number of elements of an array.
- f(n) is a function that denotes the number of operations needed to solve the problem when the input size is ‘n’.
- The faster the f(n) grows , the slower the algorithm will we because the number of operation i.e. the amount of work performed by the algorithm is growing rapidly.

Analysing an Algorithm

- **Big O (O) Notation: Worst Case behavior**
- **Theta (Θ) Notation: Average Case behavior**
- **Omega (Ω) Notation: Best Case Behavior**

Big O Notation (O) : Worst Case behavior

- While analyzing any algorithm, big O notation is used to classify these algorithms on how they respond to changes in input size.
- Big-O is the method of expressing the upper bound of an algorithm's running time.
- It measures the longest amount of time it could possibly take for the algorithm to complete.
- It is used to analyze an algorithm and to find the number of operations it performs.



Big-O(O) function behaviour analysis

- x – axis: number of elements
- y – axis: number of operations

Data Structures: A Programming Approach with C, PHI, Dharmender Singh Kushwaha & A.K.Misra

Big O

- Since “Big –O” is an upper bound, it is often used to state the complexity of a worst case analysis.
- For a positive constant n_0 and c, we see that to the right of n_0 the value of f(n) always lies on or below cg(n).

Some common order of magnitudes

O(1)

- A computing time bounded by a constant is referred to as O(1) bounded time or constant time, meaning thereby that the time required for the operation is not dependent on the size of the input to the problem.
- Eg.- Assigning a value to the i^{th} element in an array of 'n' element is O(1) since an element in an array can be accessed directly through the index.

Some common order of magnitudes

O(n)

- All O(n) functions are said to execute in linear time.
- Eg. Printing all the elements in a list of ‘n’ elements is O(n) or searching for a particular value in a list of unordered element is also O(n) because we have to search every element in the list to find it.

Some common order of magnitudes

$O(\log_2 n)$

- This kind of function do more work than O(1) but lesser than O(n).
- Eg. Finding a value in a list of ordered elements using binary search algorithm is $O(\log_2 n)$.

When do Constants Matter?

- When two algorithms have **different** big-O time complexity, the constants and low-order terms only matter when the problem size is small.
- For example, even if there are large constants involved, a linear-time algorithm will always eventually be faster than a quadratic-time algorithm.

Best-case and Average-case Complexity

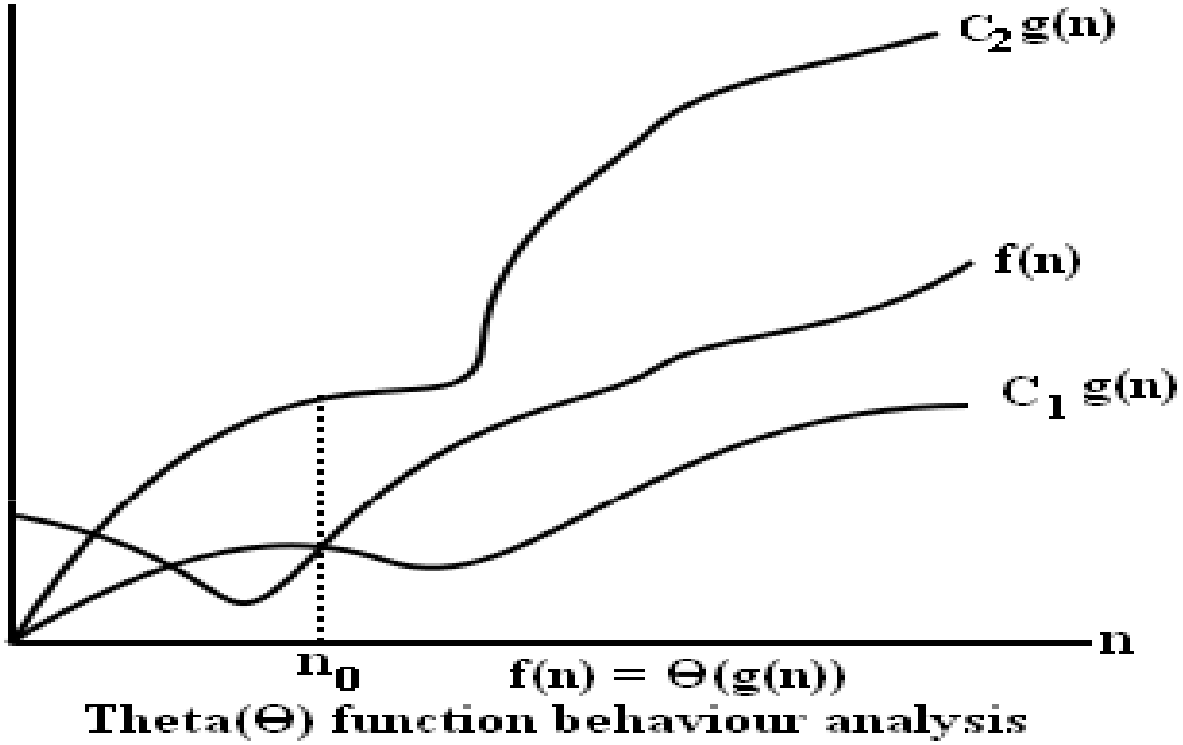
- We may want to consider the **best** and **average** time requirements of a method as well as its worst-case time requirements.
- Which is considered the most important will depend on several factors.
- For example, if an algorithm is part of a time-critical system like the one that controls an airplane, the worst-case times are probably the most important in order to avert any mishap.

Best-case and Average-case Complexity

- Here the best-case and average-case times for that computation are not relevant. The computation needs to be guaranteed to be fast enough to finish before the plane hits the mountain.

Theta (Θ) Notation: Average Case behavior

- This represents the average case scenario for any algorithm.
- $\Theta(g(n))$ is a set of functions, we write "$f(n) = \Theta(g(n))$" to mean that $f(n) \in \Theta(g(n))$.
- We observe that to the right of n_0, the value f(n) always lies between $c_1\, g(n)$ and $c_2\, g(n)$.
- Thus it basically tells us that the function f(n) is bounded both from the top and bottom by the same function, g(n).

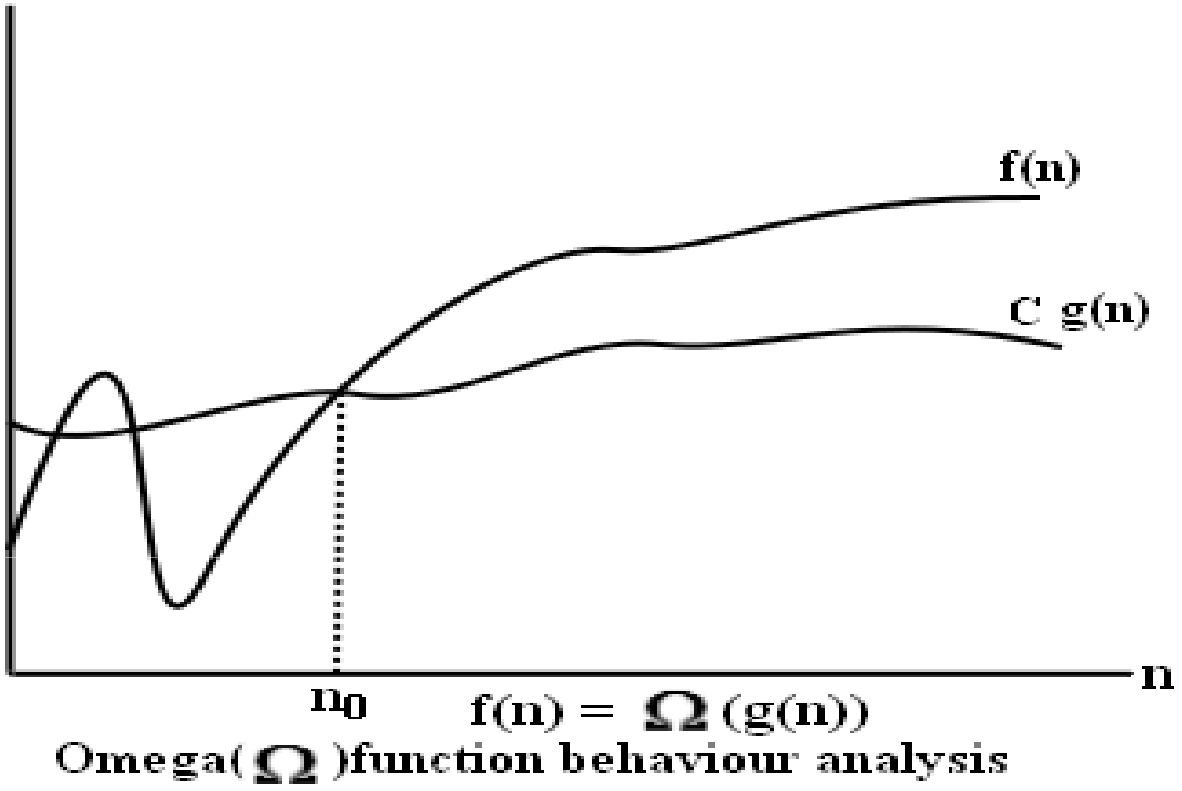


Theta(Θ) function behaviour analysis

- x – axis: number of elements
- y – axis: number of operations

Omega (Ω) Notation: Best Case Behavior

- This represents lower bound for a function. If there are positive constant n_0 & c such that to the right of n_0, the value of f**(n)** always lies on or above c g(n).
- This is almost the same definition as Big O, except that "f(n) ≥ cg(n)", that makes g(n) a lower bound function instead of an upper bound function.
- Thus it represents the best case scenario for a given data size.



Omega(Ω)function behaviour analysis

- x – axis: number of elements
- y – axis: number of operations

Example

- Compute the complexity of the following loop that adds two to each member of an array of n numbers:

```
for (int p=0; p<n; p++)
{
  q[p]=a[p]+2
}
```

- **Solution:** The above loop is executed from 0 to (n-1) i.e. n times. After removing constants (like the time it takes it to perform the addition), the upper bound on the execution time will be O(n).

Data Structures Widely Used in Computer Sciences

- Arrays
- Linked List
- Stack
- Queues
- Tree
- Graph
- Heap

Array

- Array is the collection of similar types of objects that maps a series of memory sequence with same name.
- Array can be defined as variable name with the size of array given in index.
- Due to index based data structure, any element can be directly accessed from the array by just giving the index of that element.

Linked list

- Linked list is data structure in which node is made up of two elements one with node value and the other is the pointer to the next node.
- Linked list can dynamically vary its size during the execution of program.
- Many types of linked lists are used such as linear singly linked list, circular linked list and doubly linked list.
- Linked list are commonly used to define abstract data types such as queue, stack and tree.

Stack

- Stack is the last in first out (LIFO) data structure.
- It can be used by two common commands push and pop.
- Push inserts a element into stack at the top while a pop fetches last element from the stack.
- Element that is inserted first is fetched out last.
- Stacks are used in computations like in calculator for expression evaluation and syntax parsing.

Queue

- Queue is the data structure which supports first in first out (FIFO) operation.
- Element which is inserted first would exit first.
- Queue has two ends front and rear. Front points to first element while rear points to last element of the queue.
- Queues are often implemented in operating system for ready queue or waiting queue or to use as a buffer of inter process communication usage.

Tree

- Tree is the data structure in which each node has either 0 or n number of children, but with one parent.
- Top node in the tree is called root node, while nodes with 0 children are called leaf nodes.
- Common operations in the tree include insertion, deletion, and traversing.
- Trees are commonly used when the maintenance of hierarchical data is required and for applications where search is the common feature.

Graph

- Graph is a data structure which is the collection of edges and vertices, in which a pair of vertices is connected by an edge.
- Graphs can be either directed or undirected.
- Undirected graph does not have any direction in the edges while directed graphs have directions.
- Applications in which graph can be used are sitemap for portals and problems of finding shortest paths.
- Graphs are also used in network analysis.

Heap

- Heap is a data structure based on tree in which each element is associated with a key.
- Two types of heaps are common namely max-heap and min-heap.
- Applications in which heap are broadly used are heap sort and selection algorithms like finding the max and min of the elements.