Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Department of Information Technology

Chapter-1
Introduction to Data structures and Analysis

Introduction to Data structures, Need of Data structures, Types of Data structures : Linear and non linear data structures Arrays, Stacks, Queue, Linked list and Tree, Graph, Recursion, ADT (Abstract Data type). Introduction to Analysis, Algorithms, characteristics of an algorithms, Time and Space complexities, Order of growth functions, Asymptotic notations

UNIT-1

Introduction to Data structure and Analysis

Introduction to data Structures

Data structure is a way of collecting and organising data in such a way that we can perform operation on these data in an effec-tive way. Data Structures is about rendering data elements in terms of some relationship for better organization and storage. for example, we have data player's name "Virat" and age 26. Here "Virat" is of string data type and 26 is of integer data type.

We can organize this data as a record like Player record. Now we can collect and store player's records in a file or database as a data structure. For example "Dhoni" 30 "Gambhir" 31, "Sehwag" 33

In simple language, Data Structures are structures programmed to store ordered data so that various operation can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Need for Data Structure

As applications are getting complex and data rich, there are three common problems that application face now-a-days.

Data Search :- Consider an inventory of 1 million (10^6) item of a store. If application is to search an item, it has to search an item in 1 million (10^6) item every time slowing down the search. As data grows, search will become slower.

Processor speed :- Processor speed although being very high, falls limited if data grows to billion records.

Multiple requests :- As thousand of users can search data simultaneously on a web server even the fast server fails while searching the data.

To solve the above-mentioned problems, data structures come to rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Type of Data Structures: Linear and Non-linear Data Structures.

A data structure is a method for organizing and storing data, which would allow efficient data retrieval and usage. Linear data structure is a structure that organizes its data elements one after the other. Linear data structures are organized in a way similar to how the computer's memory is organized. Nonlinear data structures are constructed by attaching a data element to several other data element in such a way that it reflects specific relationship among them. Nonlinear data structures are organized in a different way than the computer's memory.

Linear data Structures

Linear data structures organize their data element in a linear fashion, where data elements are attached one after the other. Data elements in a liner data structure are traversed one after the other and only one element can be directly reached while traversing. Linear data structure are very easy to implement, since the memory of the computer is also organized in a linear fashion. Some commonly used linear data structures are arrays, linked list, stacks and queues. An arrays is a collection of data element where each element could be identi-fied using an index. A linked list is a sequence. A stack is actually a list where data elements can only be added or removed from the top of the list. A

Queue is also a list, where data elements can be added from one end of the list and removed from the other end of the list.

Non-Linear data Structure

In non-linear data structure data elements are not organized in a sequential fashion. A data item in a nonlinear data structure could be attached to several other data element to reflect a special relationship among them and all the data items cannot be traversed in a single run. Data structure like multidimensional array is simply a collection of one-dimentional arrays. A tree is a data structure that is made up of a set of linked nodes, which can be used to represent a hierarchical relationship among data elements. A graph is a data structure that is made up of a finite set of edges and vertices. Edges represent connection or relationship among vertices that stores data elements.

Difference between Linear and Nonlinear Data Structures :-

Data Main difference between linear and nonlinear data structure lie in the way they organize data element. In linear data structures, data element are organized sequentially and therefore they are easy to implement in the computer's memory. In nonlinear data structure, a data element can be attached to several other data element to represent specific relationships that exist

among them. Due to this nonlinear structure they might be difficult to be implemented in computer's linear memory compared to implementing linear data structures. Selecting one data structure type over the other should be done carefully by considering the relation-ship among the data elements that needs to be stored.

Classification Data Structures

Data structure can be broadly classified in two categories - linear structure and hierarchical structure. Arrays, linked lists, stacks and queue are linear structures, while trees, graphs, heaps etc are hierarchical structure.

Arrays

Arrays are statically implemented data structure by some programming language like C and C++ hence the size of this data structure must be known at compile time and cannot be altered at run time. But modern programming a way to alter the size of them at run time. Arrays are the most common data structure used to store data.

Stack

Stack is a last-in-first out strategy ~~data~~ data structure; this means that the element stored in last will be removed first. Stack has specific but very useful application some of them are as follows :-

① Solving Recursion - recursive calls are placed onto a stack & removed from there once they

processed.
② Evaluating post-fix expression
③ Solving Tower of Hanoi
④ Backtracking
⑤ Depth-first search
⑥ ~~Conv~~ Converting a decimal number into a binary number

Queue

Queue is a first-in first out data structure The element that is added to the queue data structure first, will be removed from the queue first. Dequeue, priority queue and circular queue are the various of Queue data structure. Queue has following application uses:

- Access to shared resources
- Multiprogramming
- Message queue

Trees

Tree is a hierarchical data structure. They very top element of a tree is called the root of the tree. Except the root elements in the left sub-tree come before the root in sorting order and all those in the right sub-tree come after the root.

Tree is the most useful data structure when you have hierarchical information to store. For example, directory structure of a file system ~~there~~ there are many variant of tree you will come across. Some of them are Red-black tree, threaded binary tree, AVL tree etc.

Graph

Graph is networked data structure that connect a collection of nodes called vertices, by connection called edges. An edge can be seen as a path or communication link between two nodes. These edges can be either directed or undirected. If a path is directed then you can move in one direction only while in an undirected path the movement is possible in both directions.

Recursion

Recursion is the process of repeating items in a self-similar way. In programming language if a program allows you to call a function inside the same function, then it is called a recursive call of the function

```
void recursion () {
        recursion(); /* function calls itself */
}
int main () {
    recursion ();
}
```

The C programming language supports recursion ie a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, Otherwise it will into an infinite loop.

Recursive function are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating fibonacci series, etc.

An ADT is a mathematical model of data structure that specifies the type of data stored the operation supported on them and the types of parameters of the operation, An ADT specifies what each operation does but not how it does it. Typically, an ADT can be implemented using one of many different data structure. A useful first step in deciding what data structure to use in a program is to specify an ADT for the program
In general, the steps of building ADT to data structures are :-

1. Understand and clarify the nature of the target information unit.
2. Identify and determine which data objects and operation to include in the models
3. Express this property somewhat formally so that it can be understood and communicate well
4. Translate this formula specification into paper language. In C++ this becomes a.h file. In Java, this is called "User interface"
5. Upon finalizing specification, write necessary implementation. This includes storage scheme and operational detail. Operational detail is expressed as seperate function.

Introduction to Algorithm Analysis

There are many important things that should be taken care of, like user friendliness modularity, security, maintainbility etc. Why to worry about performance?
The answer to this is simple, we can have all the above things only if we have performance So performance is like currency through which we can buy all the above things.
Given two algorithms for a task, how do we find out which one is better?
One naive way of doing this is implement both the algorithms and run the two programs on your computer for different inputs and see which on takes less time. There are many problems with this approch for analysis of algorithms.

1) It might be possible that for some inputs first algorithm performs better than the second. And for some inputs second performs better.
2) It might also possible that for some inputs first algorithm perform better on one machine and the second works better on other machine for some other inputs.

What is an Algorithm?

An algorithm is a set of instruction or logic, written in order, to accomplish a certain predefined task. Algorithm is not the

complete code or program, it is just the core logic (solution) of a problem, which can be expressed either as an informal high level description as pseudocode or using a flowchart.

Every Algorithm must satisfy the following properties :-

① Input - There should be 0 or more inputs supplied externally to the algorithm.

② Output - There should be atleast 1 output obtained

③ Definiteness - Every step of the algorithm should be clear as well defined

④ Finiteness - The algorithm should have finite number of steps.

⑤ Correctness - Every step of the algorithm must generate a correct output.

An algorithm is said to be efficient and fast if it takes less time to execute and consumes less memory space. The performance of an algorithm is measured on the basis of following properties

1) Time Complexity
2) Space Complexity

he second works better on other machine for some other inputs.

Space Complexity

Its the amount of memory space required by the algorithm, during the course of its execution. Space complexity must be taken seriously for multi-user system and in

situation where limited memory is available
An algorithm generally requires space for following components :-

1. Instruction Space : Its the space required to store the executable version of the program This space is fixed, but varies depending upon the number of lines of code in the program.
2. Data Space :- Its the space required to store all the constants and variables value.
3. Environment Space :- Its the space required to store the environment information needed to resume the suspended function.

Time Complexity :

Time complexity is a way to represent the amount of time needed by the program to run till its completion. We will study this in details in later section.

Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-18

Asymptotic Analysis

Asymptotic analysis is the big idea that handles above issues in analyzing algorithm In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time) We calculate, how does the time (or space) taken by an algorithm increases with the input size. Asymptotic Analysis is not perfect. but that's the best way available for analyzing algorithm.

We can have three cases to analyze an algorithm.

1. Worst Case
2. Average Case
3. Best Case

Let's us consider the following implementation of linear search

```
#include <stdio.h>
// Linearly search x in arr[]. If x is present
   then return the indexe,
// Otherwise return -1
int search (int arr[], int n, int x)
{
      int i;
      for (i=0; i<n; i++)
      {
        if (arr[i] == x)
          return i;
      }
return -1;
```

```
}
/* Driver program to test above function */
int main()
{
    int arr[] = {1, 10, 30, 15};
    int x = 30;
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("%d is present at index %d", x,
      search(arr, n, x));
    getchar();
    return 0;
}
```

Worst Case Analysis (Usually Done)

In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that cause maximum number of operations to be executed. For linear search, the worst case happens when the element to be searched (~~-) functions e~~ (x in the above code) is not present in the array. When x is not present, the search() functions compares it with all the elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be $\Theta(n)$.

Average Case Analysis (Sometimes done)

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of

inputs. We must know (or predict) distrib - ution of cases. For the linear search problem let us assume that all cases are uniformly distributed (including the case of x not being present in array) So we sum all the cases and divide the sum by (n+1). following is the value of average case time complexity

$$\text{Average Case Time} = \frac{\sum_{i=1}^{n+1} \Theta(i)}{n+1}$$

$$= \frac{\Theta((n+1) * (n+2)/2)}{n+1}$$

$$= \Theta(n)$$

Best Case Analysis

In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operation to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operation in the best case is constant (not dependent on n) So time complexity in the best case would be $\Theta(1)$

Most of the times, we do worst case analysis to analyze algorithms. In the worst analysis, we guantee an upper bound on running time of an algorithm which is good information.

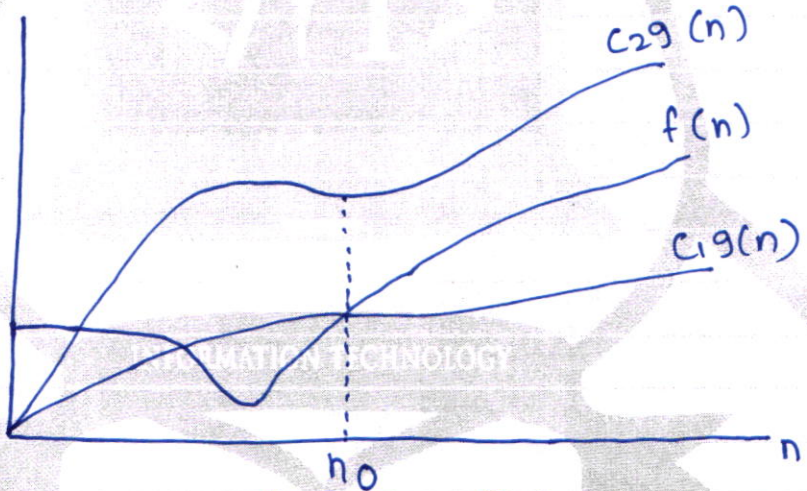
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-

Asymptotic notations (University Question)

Asymptotic notations are mathematical tools to represent time complexity of algorithm for asymptotic analysis. The following 3 asymptotic notations are mostly used to represent time complexity of algorithm.

1) Θ Notation: The theta notation bounds a functions from above and below, so it defines exact asymptotic behavior.



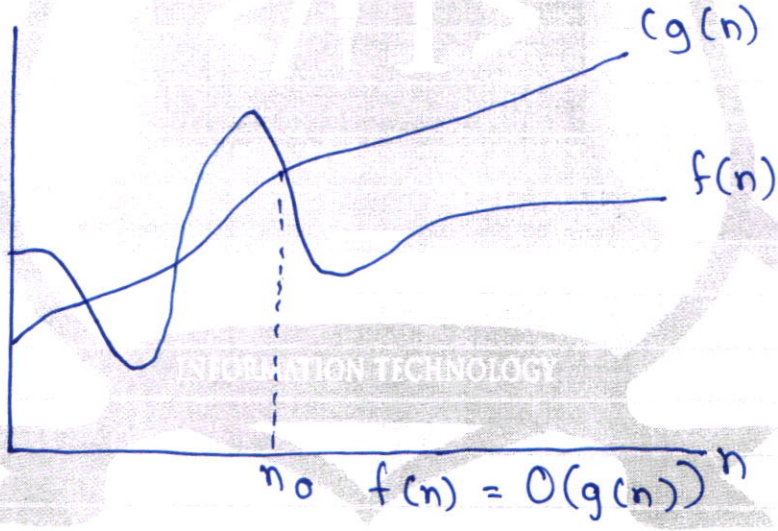
$f(n) = \Theta(g(n))$

For a given function g(n), we denote Θ(g(n)) is following set of functions.

$\Theta(g(n)) = \{f(n)$: there exist positive constant C_1, C_2 and $n\Theta$ such that $\Theta <= C_1 * g(n) <= f(n) <= C_2 * g(n)$ for all $n >= n\Theta\}$

The above defination means if f(n) is theta of g(n), then the value f(n) is alway between C1 *g(n) and C2*g(n) for large value of n(n>=n0) The definition of theta also requires that f(n) must be non-negative for value of n greater than n0.

2) Big O Notation: The Big O notation defines an upper bound of an algorithm. it bounds a function only from above



The Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm

$O(g(n)) = \{ f(n)$: there exist positive constant C and n0 such that $0 <= f(n) <= cg(n)$ for all $n >= n0\}$

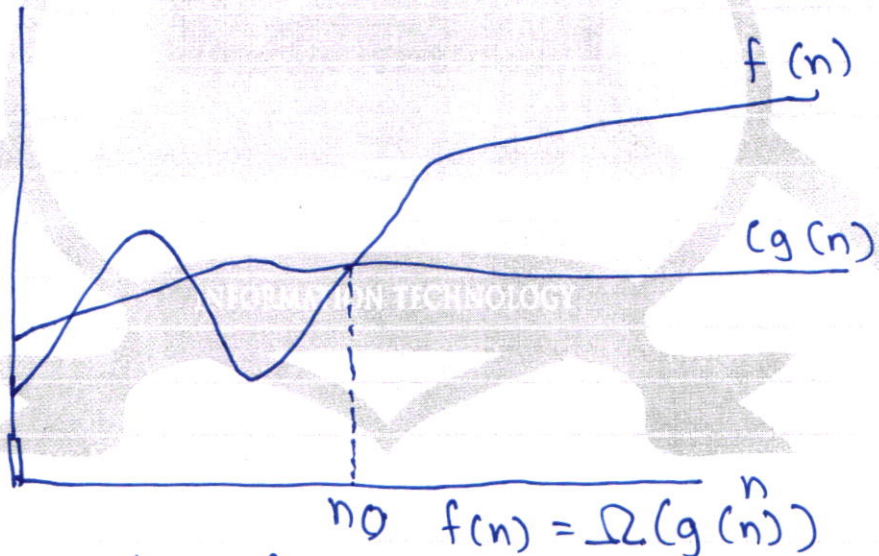
Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Semester: III Subject: DSA Academic Year: 2017-1

3) Ω Notation

Just as Big O notation provides an asymp-totic upper bound on a function, Ω notation provides an asymptotic lower bound.

Ω Notation < can be useful when we have lower bound on time complexity of an algorithm. As discussed in the previous post the best case performance of an algorithm is generally not useful. the Omega notation is the least used notation among all three.



For a given function g(n), we denote by Ω(g(n)) the set of function

Ω (g(n)) = {f(n): there exist positive constant C and n0 such that 0 <= cg(n) <= f(n) for all n >= n0 }.