



Dr. Vishwanath Karad

**MIT WORLD PEACE
UNIVERSITY** | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Fundamentals of Data Structures (CET1043B)

S. Y. B. Tech CSE

Semester – III

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY



Fundamentals of Data Structures (CET1043B)

- Pre-requisites
 - Programming Foundations
- Course Objectives
 - Knowledge
 - Learn the linear data structure and its fundamental concept.
 - Skills
 - Understand the different linear data structures such as Array, Stack, Queue and Linked list.
 - Study the different searching and sorting technique.
 - Attitude
 - Learn to apply advanced concepts of linear data structure to solve real world problems.



Fundamentals of Data Structures (CET1043B)

□ Course Outcomes

- After completion of the course the students will be able to
 - To develop skills for writing and analyzing algorithms to solve domain problems.
 - To compare and contrast different linear data structures and identify appropriate usage.
 - To demonstrate the use of sequential data structures.
 - To analyze different searching and sorting algorithms so as to understand their applications.



Fundamentals of Data Structures (CET1043B) Theory Syllabus

- **Unit I – Introduction to Data Structures:** Data, Data Objects, Abstract Data types (ADT) and Data Structures, Types of data structures (Linear and Non-linear, Static and dynamic) Introduction to algorithms Analysis of Algorithms- Space complexity, Time complexity, Asymptotic notations-Big-O, Theta and Omega, finding complexity using step count method, Analysis of programming constructs-Linear, Quadratic, Cubic, Logarithmic.
- **Unit II – Linear Data Structures:** Array as an Abstract Data Type, Sequential Organization, Storage Representation and their Address Calculation: Row major and Column Major, Multidimensional Arrays: Concept of Ordered List, Single Variable Polynomial: Representation using arrays, Polynomial as array of structure, Polynomial addition, Polynomial evaluation and Polynomial multiplication. Sparse Matrix: Sparse matrix representation using array, Sparse matrix addition, Transpose of sparse matrix- Simple and Fast Transpose, Time and Space trade-off.



Fundamentals of Data Structures (CET1043B)

- **Unit III – Searching and Sorting:** Searching: Search Techniques-Sequential Search/Linear Search, Variant of Sequential Search- Sentinel Search, Binary Search, and Fibonacci Search. Sorting: Types of Sorting-Internal and External Sorting, General Sort Concepts-Sort Order, Stability, Efficiency, and Number of Passes, Comparison Based Sorting Methods-Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge sort and Shell Sort, Non-comparison Based Sorting Methods-Radix Sort, and Bucket Sort, Comparison of All Sorting Methods and their complexities.
- **Unit IV – Linked List:** Linked List as an Abstract Data Type, Representation of Linked List Using Sequential Organization, Representation of Linked List Using Dynamic Organization, Types of Linked List: singly linked, Circular Linked Lists, Doubly Linked List, Doubly Circular Linked List, Primitive Operations on Linked List, Polynomial Manipulations-Polynomial addition. Generalized Linked List (GLL) concept, Representation of Polynomial using GLL. Case Study : Garbage Collection



Fundamentals of Data Structures (CET1043B)



Unit V – Stacks: Stack as an Abstract Data Type, Representation of Stack Using Sequential Organization, stack operations, Multiple Stacks Applications of Stack-Expression Conversion and Evaluation, Linked Stack and Operations, Recursion.
Queues: Queue as Abstract Data Type, Representation of Queue Using Sequential Organization, Queue Operations Circular Queue, Advantages of Circular queues, Linked Queue and Operations Deque-Basic concept, types (Input restricted and Output restricted), Application of Queue: Job scheduling.



Fundamentals of Data Structures (CET1043B) List of Lab Assignments

1. Write a C program to compute following computation on matrix: a) Addition of two matrices b) Subtraction of two matrices c) Multiplication of two matrices d) Transpose of a matrix
2. Write a C program for sparse matrix realization and operations on it- Transpose, Fast Transpose.
3. Write a C program to create student database using array of structures. Apply searching (Linear and Binary Search) and sorting techniques (Insertion Sort, Selection Sort, Shell sort).
4. Write a C program to store first year percentage of students in array. Write function for sorting array of numbers in ascending order using bucket sort and display top five scores.
5. Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of second, third and final year of department can be granted membership on request. Similarly, one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C program to maintain club member 's information using singly linked list. Store student PRN and Name. Write functions to: a) Add and delete the members as well as president or even secretary. b) Compute total number of members of club c) Display members d) sorting of two linked list e)merging of two linked list f)Reversing using three pointers



Fundamentals of Data Structures (CET1043B) List of Lab Assignments

6. Implement following polynomial operations using Circular Linked List: i) Create ii) Display iii) Addition.
7. Implement stack as an ADT and apply it for different expression conversions (infix to postfix or infix to prefix (Any one), prefix to postfix or prefix to infix, postfix to infix or postfix to prefix (Any one)).
8. Write a C program to evaluate postfix expression using stack.
9. Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write a program for simulating job queue. Write functions to add job and delete job from queue

Fundamentals of Data Structures (CET1043B)

□ Learning Resources:

□ Text Books:

1. Fundamentals of Data Structures, E. Horowitz, S. Sahni, S. A-Freed, Universities Press.
2. Data Structures and Algorithms, A. V. Aho, J. E. Hopperoft, J. D. Ullman, Pearson.

□ Reference Books:

1. The Art of Computer Programming: Volume 1: Fundamental Algorithms, Donald E. Knuth.
2. Introduction to Algorithms, Thomas, H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press.
3. Open Data Structures: An Introduction (Open Paths to Enriched Learning), (Thirty First Edition), Pat Morin, UBC Press.

Fundamentals of Data Structures (CET1043B)

□ Supplementary Readings:

1. Aaron Tanenbaum, "Data Structures using C", Pearson Education.
2. R. Gilberg, B. Forouzan, "Data Structures: A pseudo code approach with C", Cenage Learning, ISBN 9788131503140.

□ Web Resources:

□ Web Links:

- https://www.tutorialspoint.com/data_structures_algorithms/

□ MOOCs:

- <http://nptel.ac.in/courses/106102064/1>
- <https://nptel.ac.in/courses/106103069/>



Introduction to Data Structures

- ❑ Data, Data objects, Data Types
- ❑ Abstract Data types (ADT) and Data Structure
- ❑ Types of data structure
- ❑ Introduction to Algorithms
- ❑ Algorithm Design Tools: Pseudo code and flowchart
- ❑ Analysis of Algorithms- Space complexity, Time complexity, Asymptotic notations



Data, Data Objects and Data Types

- Computer Science is **study of data**
 - Machines that **hold data**
 - Languages for describing **data manipulations**
 - Foundations which describe **what kinds of refined data** can be produced from raw data
 - **Structures of refining data**

Data, Data Objects and Data Types

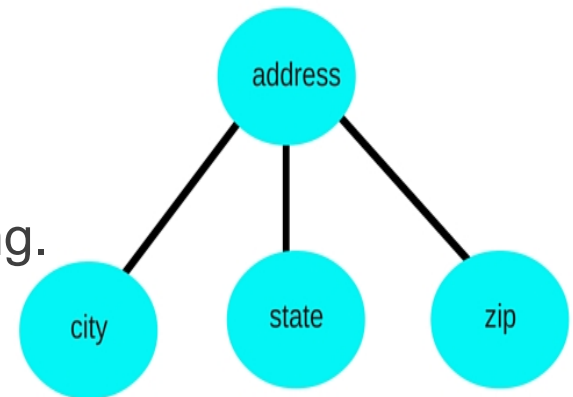
Data is of two types

Atomic Data

It consist of single piece of information. It cannot be divided into other meaningful pieces of data. e.g Name of Person, Name of Book

Composite Data

It can be divided into subfields that have meaning.
e.g. Address, Telephone number



Data object is referring to set of elements say D.

Roll_Number

Name

Percentage

©QJun * illustrationsOf.com/71457

For Example:

If student is one object then it will consist of different data like roll no, name, percentage, address etc.

Data, Data Objects and Data Types

□ Data Types

- A Data type is a term which **refers to the kinds of data** that variables may hold in a programming languages.
- For Example: **In C programming languages, the data types are integer, float, character etc.**
- **Data type is a way to classify various types of data** such as integer, string, etc. which determines the values that can be used with the corresponding type of data, the type of operations that can be performed on the corresponding type of data.
- There are two data types –
 - Built-in Data Type
 - Derived Data Type

Data, Data Objects and Data Types

Built-in Data Type

Those data types for which a language has built-in support are known as **Built-in Data types**.

For example, most of the languages provide the following built-in data types.

- Integers
- Boolean (true, false)
- Floating (Decimal numbers)
- Character and Strings

Derived Data Type

These data types are normally **built by the combination of primary or built-in data types** and associated operations on them.

For example –

- List
- Array
- Stack
- Queue



Abstract Data Type(ADT) and Data Structure

☐ Abstract Data Type

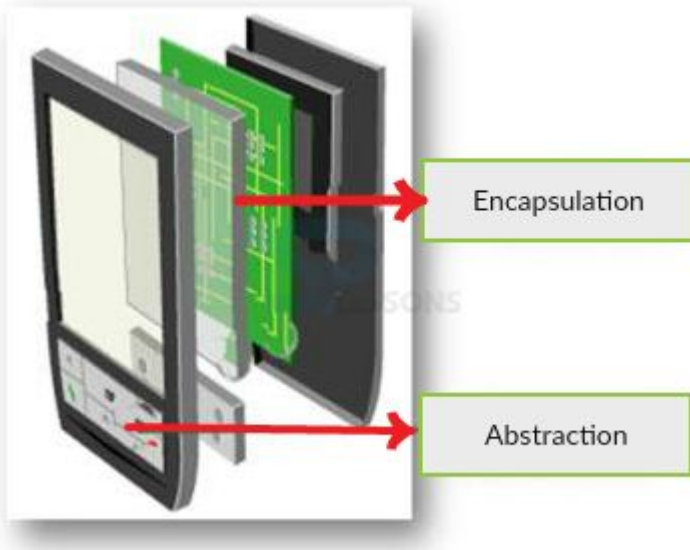
- ☐ Concern about what can be done not how it can be done

Abstract Data Type consist of

- ☐ Declaration of Data
- ☐ Declaration of Operations
- ☐ Encapsulation of data and operations

Abstract data types

An abstract data type is a type with associated operations, but whose representation is hidden.



- The calculator explains it very well.
- One can use it different ways by giving various values and perform operations.
- But, mechanism **how the operation is done is not shown.**
- This process of hiding the information is called as *Abstraction*.

Abstract Data Types (ADT)

- An **ADT** is composed of
 - A collection of data
 - A set of operations on that data
- Specifications of an **ADT** indicate
 - What the ADT operations do, not how to implement them
- Implementation of an **ADT**
 - Includes choosing a particular data structure

Abstract Data Type(ADT) and Data Structure

Abstract Data Type Examples

- ☐ Array
- ☐ Tree
- ☐ Graph
- ☐ Linked List
- ☐ Matrix

```
structure ARRAY(value, index)
  declare CREATE()→array
           RETRIVE(array,index)→value
           STORE(array,index,value)→array;
  for all A ∈ array, i,j ∈ index ,x ∈ value let
    RETRIVE (CREATE,i) :: = error
    RETRIVE (STORE(A,i,x),j) :: =
      if EQUAL(i,j) then x else
    RETRIVE(A,j)
  end
end ARRAY
```

Data Structure

- A data structure is a set of domains D , a structured domain $d \in D$, a set of functions F and a set of axioms A .
- The triple (D, F, A) denotes the data structure $d \in D$ and it will usually be abbreviated by writing d .
- The triple (D, F, A) is referred to as an abstract data type (ADT).
- It is called **abstract** precisely because the axioms do not imply a form of representations/implementation.

Data Structure

Example

Structure NATNO

Declare ZERO() \rightarrow natno

ISZERO(NATNO) \rightarrow Boolean

SUCC(natno) \rightarrow natno

ADD(natno,natno) \rightarrow natno

for all $x,y \in \text{natno}$ let

ISZERO(ZERO) $::=$ true;

ADD(ZERO,y) $::=$ y;

ISZERO(SUCC(x)) = false

D=NATNO

D \in D= {Boolean, natno}

F={ZERO,ISZERO,ADD,SUCC}

AXIOMS

Types of Data Structures

□ Linear data structure:

- The data structure where **data items are organized sequentially or linearly** where data elements attached one after another is called linear data structure. **It has unique predecessor and Successor.**
- **Ex: Arrays, Linked Lists**

□ Non-Linear data structure:

- The data structure where **data items are not organized sequentially** is called non linear data structure. **It don't have unique predecessor or Successor.**
- In other words, A data elements of the non linear data structure could be connected to more than one elements to reflect a special relationship among them.
- **Ex: Trees, Graphs**

Types of Data Structures

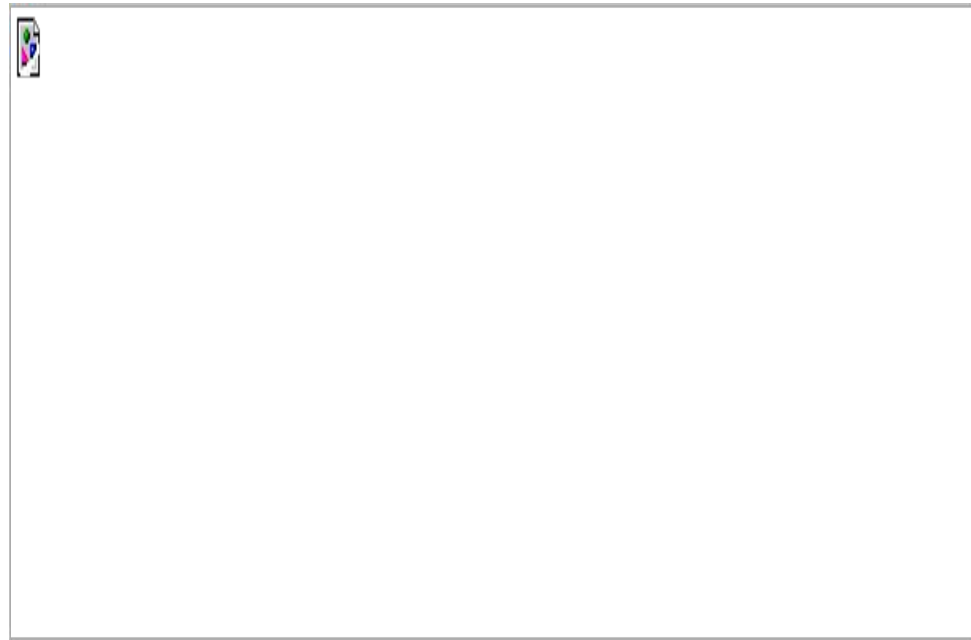
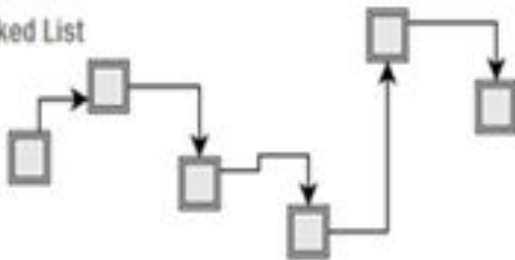
Linear Data Structure

Non Linear Data Structure

Array



Linked List



Types of Data Structures

□ Static data structure:

- Static Data structure has fixed memory size. It is the memory size allocated to data, which is static.

□ **Ex: Arrays**

□ Dynamic data structure:

- In Dynamic Data Structure, the size can be randomly updated during run time which may be considered efficient with respect to memory complexity of the code.

□ **Ex: Linked List**

- In comparison to dynamic data structures, static data structures provide easier access to elements. Dynamic data structures, as opposed to static data structures, are flexible.

Introduction to Algorithms

Algorithm

- **Solution to a problem** that is independent of any programming language.
- **Sequence of steps** required to solve the problem
- Algorithm is a finite set of instructions that if followed, accomplishes a particular task
- All algorithms must satisfy the following criteria:
 - **Input:** Zero or more Quantities are externally supplied
 - **Output:** At least one quantity is produced
 - **Definiteness:** Each instruction is clear and unambiguous
 - **Finiteness:** if we trace out the instructions of an algorithm then for all cases the algorithm terminates after a finite number of steps.
 - **Effectiveness:** Every instruction must be very basic so that it can be carried out in principle by a person using pencil and paper.

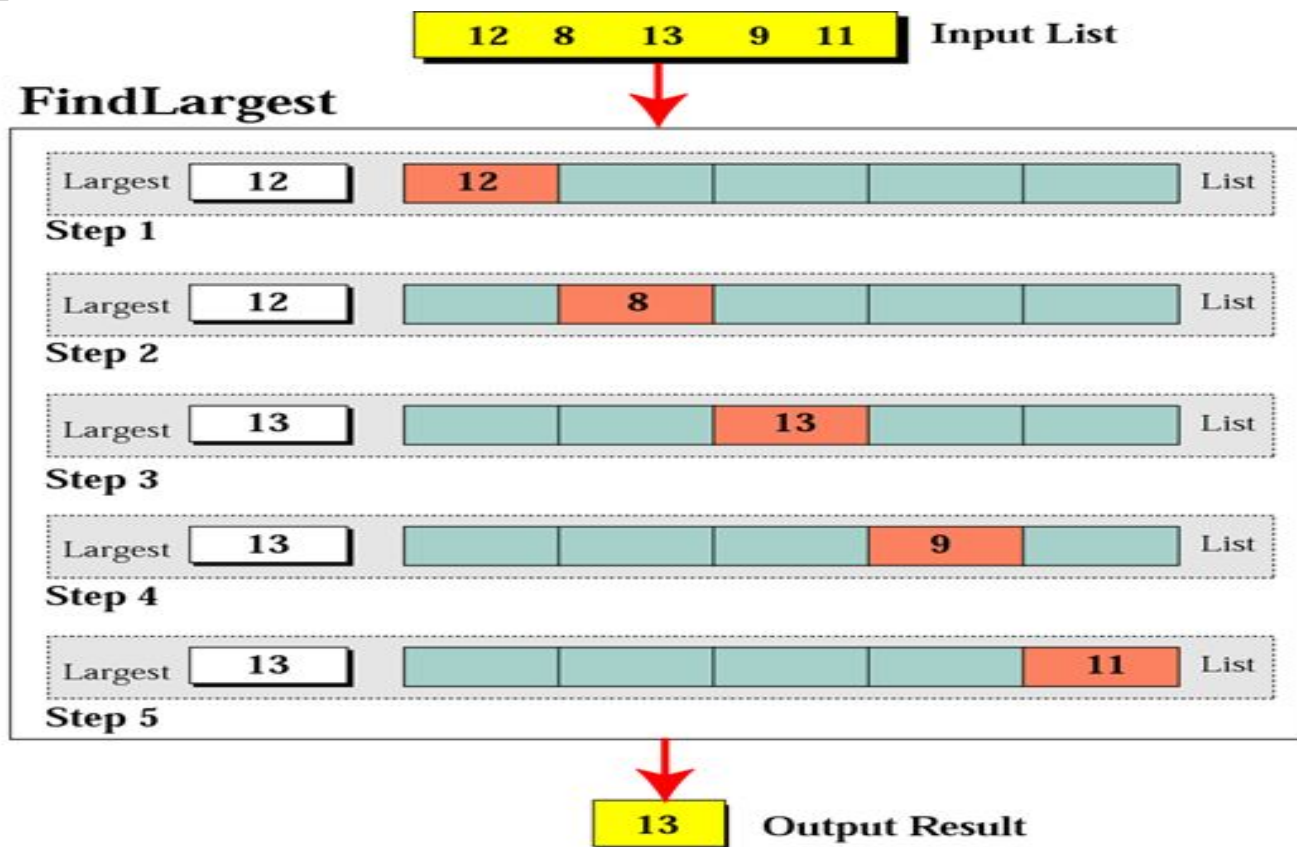
Introduction to Algorithms

Program vs Algorithm

- A program is a **written out set of statements** in a language that can be executed by the machine.
- An algorithm is **simply an idea or a solution** to a problem that is often procedurally written.

Introduction to Algorithms

Example : Finding the largest integer among five integers



Introduction to Algorithms

Defining actions in Find Largest algorithm

12 8 13 9 11 Input List

FindLargest

Set Largest to the first number.

Step 1

If the second number is greater than Largest, set Largest to the second number.

Step 2

If the third number is greater than Largest, set Largest to the third number.

Step 3

If the fourth number is greater than Largest, set Largest to the fourth number.

Step 4

If the fifth number is greater than Largest, set Largest to the fifth number.

Step 5

13

Output Result

Introduction to Algorithms

Find Largest refined

12 8 13 9 11 Input List

FindLargest

Set Largest to 0.

Step 0

If the current number is greater than Largest, set Largest to the current number.

Step 1

⋮

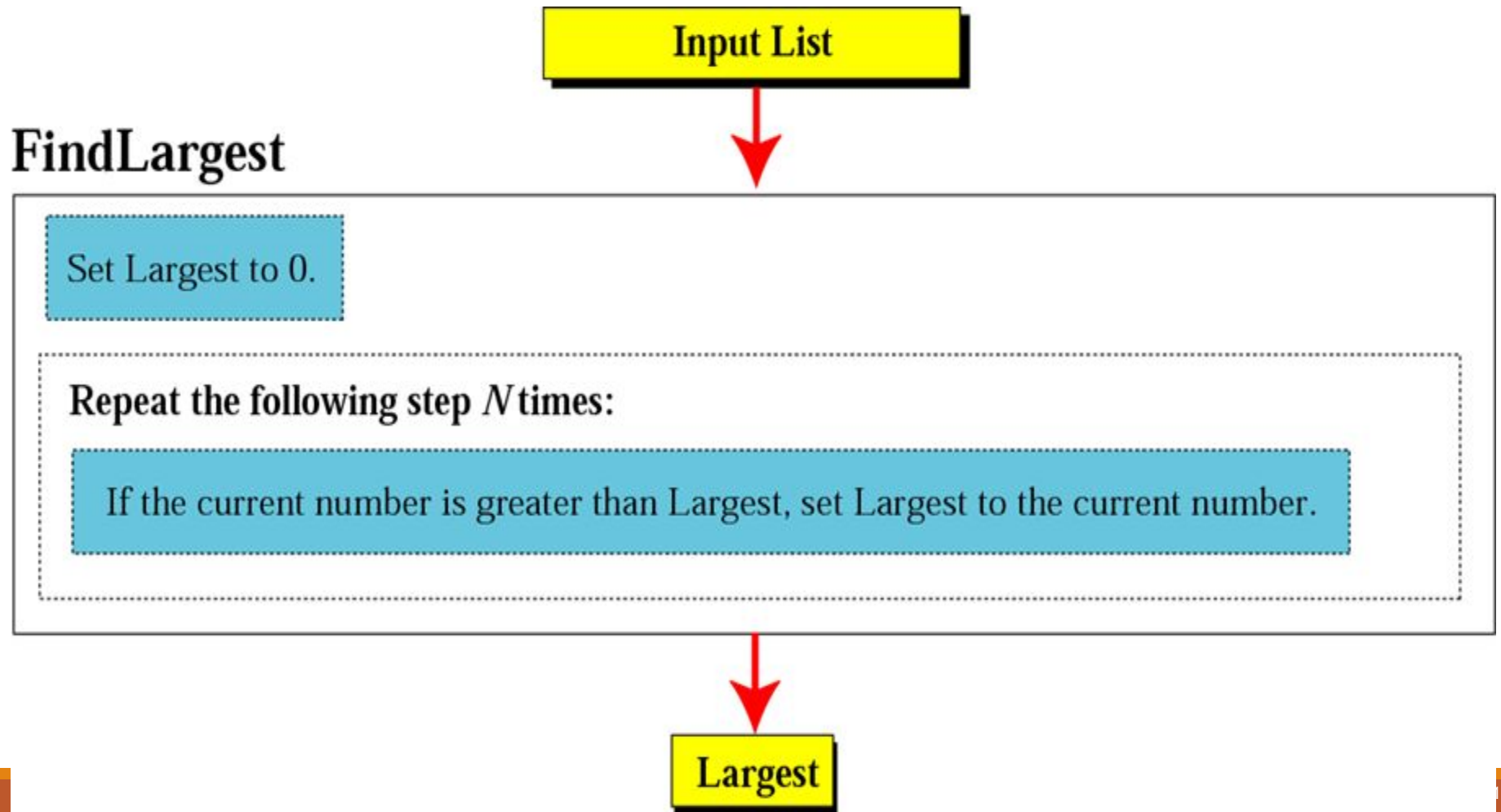
If the current number is greater than Largest, set Largest to the current number.

Step 5

13

Output Result

Introduction to Algorithms



Introduction to Algorithms

Three constructs

```
do action 1  
do action 2  
...  
...  
do action  $n$ 
```

a. Sequence

```
if a condition is true,  
then  
    do a series of actions  
else  
    do another series of actions
```

b. Decision

```
while a condition is true,  
    do action 1  
    do action 2  
    ...  
    ...  
    do action  $n$ 
```

c. Repetition



Algorithm Design Tools

➤ Pseudo Code

- ❑ is an artificial and informal language that helps programmers develop algorithms.
- ❑ Uses English-like phrases with some Visual Basic terms to outline the program

➤ Flowchart

- ❑ Graphical representation of an algorithm.
- ❑ Graphically depicts the logical steps to carry out a task and shows how the steps relate to each other.

Algorithm Design Tools

Flowchart

Example 1: Print 1 to 20:

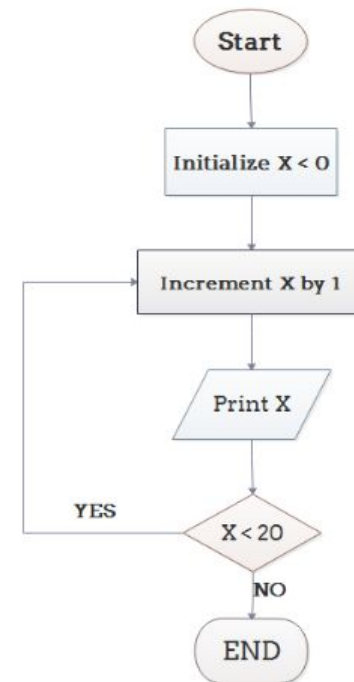
Algorithm

Step 1: Initialize X as 0,

Step 2: Increment X by 1,

Step 3: Print X,

Step 4: If X is less than 20 then go back to step 2.



Pseudo Code

- ❑ **Pseudocode** is an **informal high-level description** of the operating principle of a computer program or other algorithm.
- ❑ It uses the **structural conventions of a normal programming language**, but is intended for human reading rather than machine reading.

```
Algorithm SORT(A, n)
{
    for (i = 1; i < n; i++)
    {
        j = i ;
        for (k = j + 1; k < n; k++)
        {
            if A[k] < A[j]
                j = k;
        }
        t = A[i];
        A[i] = A[j];
        A[j] = t
    }
}
```

Pseudo Code

Examples

Algorithm grade_assignment()

```
{  
    if (student_grade >= 60)  
        print "passed"  
    else  
        print "failed"  
}
```

Examples

Algorithm grade_count()

```
{  
    total=0  
    grade_counter =1  
  
    while (grade_counter<10)  
    {  
        read next grade  
        total=total + grade  
        grade_counter=grade_counter + 1  
    }  
    class_average=total/10  
    print class_average.  
}
```

Pseudo Code

Some Keywords That Should be Used And Additional Points:

- ☐ Algorithm Keyword is used
- ☐ Curly brackets are used instead of begin-end
- ☐ Directly programming syntaxes are used
- ☐ Easy to convert into program
- ☐ Semicolons used

Pseudo Code

Some Keywords That Should be Used And Additional Points:

- ☐ Words such as set, reset, increment, compute, calculate, add, sum, multiply, ... print, display, input, output, edit, test , etc. with careful indentation tend to foster desirable pseudocode.
- ☐ Also, using words such as Set and Initialize, when assigning values to variables is also desirable.



Pseudo Code

Formatting and Conventions in Pseudo code

- ☐ INDENTATION in pseudocode should be identical to its implementation in a programming language.
- ☐ Use curly brackets for indentation
- ☐ No flower boxes (discussed ahead) in your pseudocode.
- ☐ Do not include data declarations in your pseudocode.
- ☐ But do cite variables that are initialized as part of their declarations. E.g. "initialize count to zero" is a good entry.



Pseudo Code

Calls to Functions should appear as:

Call FunctionName (arguments: field1, field2, etc.)

Returns in functions should appear as:

Return (field1)

Function headers should appear as:

FunctionName (parameters: field1, field2, etc.)

Functions called with addresses should be written as:

Call FunctionName (arguments: pointer to fieldn, pointer to field1, etc.)

Function headers containing pointers should be indicated as:

FunctionName (parameters: pointer to field1, pointer to field2, ...)

Returns in functions where a pointer is returned:

Return (pointer to fieldn)

Pseudo Code

1. Function Call

EVERY function should have a flowerbox PRECEDING IT.

This flower box is to include the functions name, the main purpose of the function, parameters it is expecting (number and type), and the type of the data it returns.

All of these listed items are to be on separate lines with spaces in between each explanatory item.

FORMAT of flowerbox should be

Function: (cryptic text describing single function
..... (indented like this)

.....
Calls: Start listing functions "this" function calls
Show these functions: one per line, indented

Called by: List of functions that calls "this" function
Show these functions: one per line, indented.

Input Parameters: list, if appropriate; else None

Returns: List, if appropriate.

Pseudo Code

➤ Advantages and Disadvantages

Pseudocode Disadvantages

- ☐ It's not visual
- ☐ There is no accepted standard, so it varies widely from company to company

Pseudocode Advantages

- ☐ Can be done easily on a word processor
- ☐ Easily modified
- ☐ Implements structured concepts well



Analysis of Algorithms

- Finding Efficiency of an algorithm in terms of
 - ☐ Time Complexity
 - ☐ Space Complexity

Analysis of Algorithms

□ What is time complexity

- Finding amount of time required for executing set of instructions or functions
- It is represented in terms of frequency count
- Frequency count is number of time every instruction of a code is to be executed.

□ What is space complexity

- Finding amount of memory space the program is going to consume.
- It is calculated in terms of variables used in program.

Common Rates of Growth

Let n be the size of input to an algorithm, and k some constant. The following are common rates of growth.

- Constant: $O(k)$, for example $O(1)$
- Linear: $O(n)$
- Logarithmic: $O(\log_k n)$
- Linear : n $O(n)$ or $n \log n$: $O(n \log_k n)$
- Quadratic: $O(n^2)$
- Polynomial: $O(n^k)$
- Exponential: $O(k^n)$

Example

```
void fun()
{
    int a;
    a=5;
    printf("%d",a);
}
```

```
void fun()
{
    int a;
    a=0;
    for(i=0;i<n;i++)
    {
        a=a + i;
    }
    printf("%d",a);
}
```

Solving Problems

Find Frequency Count and Time Complexity

```
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        C[j][j]=0;
        for(k=1;k<=n;k++)
            C[i][j]=c[i][j]+a[i][k]*b[k][j];
    }
}
```

```
i=1;
do
{
    a++;
    if(i==5)
        break;
    i++;
}
while(i<=n);
```

```
i=1;
while(i<=n)
{
    a++;
    if(i==5)
        break;
    i++;
}
```

Find Frequency Count and Time Complexity

```
i=n;  
while(i>=1){  
    i--;  
}
```

```
i=10;  
for(i=10;i<=n;i++)  
    for(j=1;j<i;j++)  
        x=x+1;
```

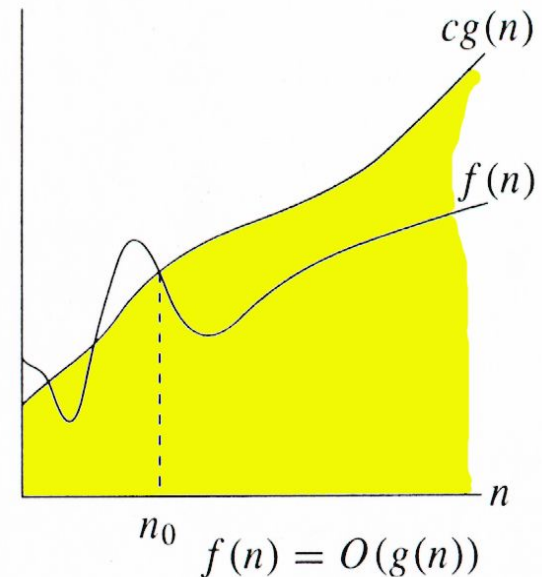
```
i=0;  
for(i=0;i<=n;i++)  
    for(j=1;j<i;j++)  
        x=x+1;
```


O-notation Example

For a given function $g(n)$, we denote $O(g(n))$ as the set of functions:

$O(g(n)) = \{ f(n) \mid \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c g(n) \text{ for all } n \geq n_0 \}$

It is used to represent the worst case growth of an algorithm in time or a data structure in space when they are dependent on n , where n is big enough.



Big Oh - Example

$$f(n) = n^2 + 5n = O(n^2)$$

$$g(n) = n^2 \dots\dots\dots c = 2$$

$n \qquad n^2 + 5n \qquad 2n^2$

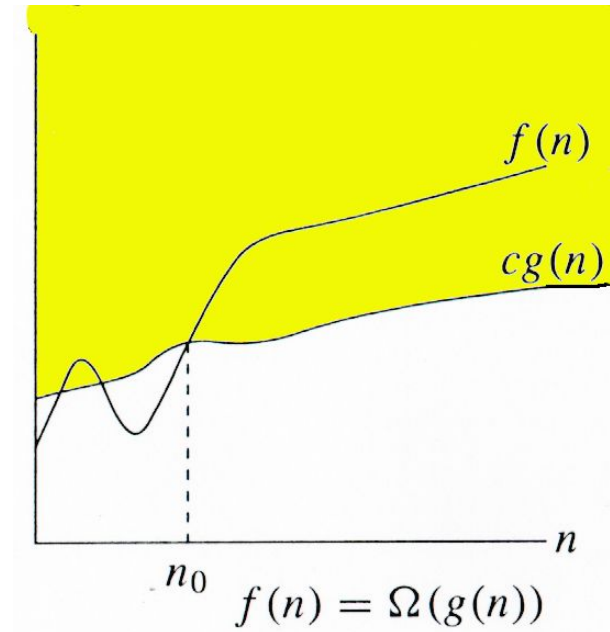
| | | |
|---|----|----|
| 1 | 5 | 2 |
| 2 | 14 | 8 |
| 5 | 50 | 50 |

$$f(n) \leq c g(n) \text{ for all } n \geq n_0 \text{ where } c=2 \text{ \& } n_0=5$$

Ω -notation

$\Omega(g(n))$ represents a set of functions such that:

$\Omega(g(n)) = \{f(n): \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq c g(n) \leq f(n) \text{ for all } n \geq n_0\}$



Big Omega - Example

Example 1 :

$$f(n) = n^2 + 5n$$

$$g(n) = n^2 \dots\dots\dots c = 1$$

| n | $n^2 + 5n$ | $c \cdot n^2$ |
|---|------------|---------------|
| 1 | 5 | 1 |
| 2 | 14 | 4 |
| 5 | 50 | 25 |

$f(n) \geq c g(n)$ for all $n \geq n_0$
where $c=1$ & $n_0=1$

Example 1 :

Prove that if $T(n) = 15n^3 + n^2 + 4$, $T(n) = \Omega(n^3)$.

Proof.

Let $c = 15$ and $n_0 = 1$.

Must show that $0 \leq cg(n)$ and $cg(n) \leq f(n)$.

$0 \leq 15n^3$ for all $n \geq n_0 = 1$.

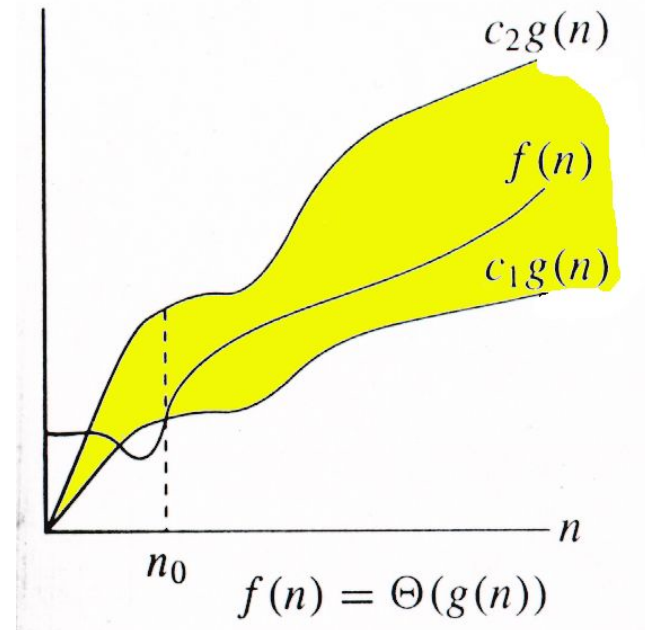
$$cg(n) = 15n^3 \leq 15n^3 + n^2 + 4 = f(n)$$

Θ -notation

Asymptotic tight bound

$\Theta(g(n))$ represents a set of functions such that:

$\Theta(g(n)) = \{f(n): \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}$



Theta Example

$f(N) = \Theta(g(N))$ iff $f(N) = O(g(N))$ and $f(N) = \Omega(g(N))$

It can be read as “ $f(N)$ has order exactly $g(N)$ ”.

The growth rate of $f(N)$ equals the growth rate of $g(N)$. The growth rate of $f(N)$ is the same as the growth rate of $g(N)$ for large N .

Theta means the bound is the tightest possible.

If $T(N)$ is a polynomial of degree k , $T(N) = \Theta(N^k)$.

For logarithmic functions, $T(\log_m N) = \Theta(\log N)$.



Analysis of Algorithms

- Algorithm analysis is done in following three cases
 - Best Case

The amount of time a program might be expected to take on best possible input data
 - Worst Case

The amount of time a program would take on worst possible input configuration.
 - Average case

The amount of time a program might be expected to take on typical(or average) input data

Example: Sorting Algorithms

Practice Assignments

1. Write a pseudo code and draw flowchart to input any alphabet and check whether it is vowel or consonant.
2. Write a pseudo code to check whether a number is even or odd
3. Write a pseudo code to check whether a year is leap year or not.
4. Write a pseudo code to check whether a number is negative, positive or zero
5. Write a pseudo code to input basic salary of an employee and calculate its Gross salary according to following:
Basic Salary \leq 10000 : HRA = 20%, DA = 80%
Basic Salary \leq 20000 : HRA = 25%, DA = 90%
Basic Salary $>$ 20000 : HRA = 30%, DA = 95%

Practice Problems

Q.1 Determine frequency count of following statements? Analyze time complexity of the following code:

```
i) for (i=1;i<=n;i++)  
    for (j=1;j<=m;j++)  
        sum=sum+i;  
  
ii) i=n;  
while(i>=1)  
{  
    i--;  
}
```

Practice Problems

Problems on frequency count & time complexity

```
for(i=1;i<=n;i++)
{
  For(j=1;j<=n;j++)
  {
    C[j][j]=0;
    For(k=1;k<=n;k++)
      C[i][j]=c[i][j]+a[i][k]*b[k][j];
  }
}
```

```
double IterPow(double X,int N)
{
  double Result=1;
  while(N>0)
  {
    Result=Result* X
    N--;
  }
  return result;
```

Practice Problems

Q.3 What is the frequency count of a statement? Analyze time complexity of following code?

```
for(i=1;i<=n;i++)  
    for(j=1;j<=m;j++)  
        for(k=1;k<=p;k++)  
        {  
            Sum=sum+i  
        }
```

Takeaway

- ☐ Data Structures plays major role in problem solving.
- ☐ Pseudo code and flowcharts are the tools used to represent the solution of a problem in effective way.
- ☐ Analysis of algorithms is done in terms of time complexity and space complexity.