Introduction to Data Structures

Unit #1

Marwadi University

Department of Computer Engineering

Data Structures
Sem 3
3130702

Ravikumar Natarajan

Highlights

- Data Management concepts
- Data types – primitive and non-primitive
- Performance Analysis and Measurement (Time and space analysis of algorithms)
- Types of Data Structures
- Linear & non-linear Data Structures

Course Outcome

- Define and classify various data structures, storage structures and common operations on them

Data Management Concepts

- A program should give correct results but along with that it should run efficiently.
- Efficient program executes:
  - Minimum time
  - Minimum memory space
- To write efficient program we need to apply Data management concepts.

Data Management Concepts

- Data Management concept includes,
  - Data collection
  - Organization of data in proper structure
  - Developing and maintaining routines for quality assurance

"A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently."

Data Structure

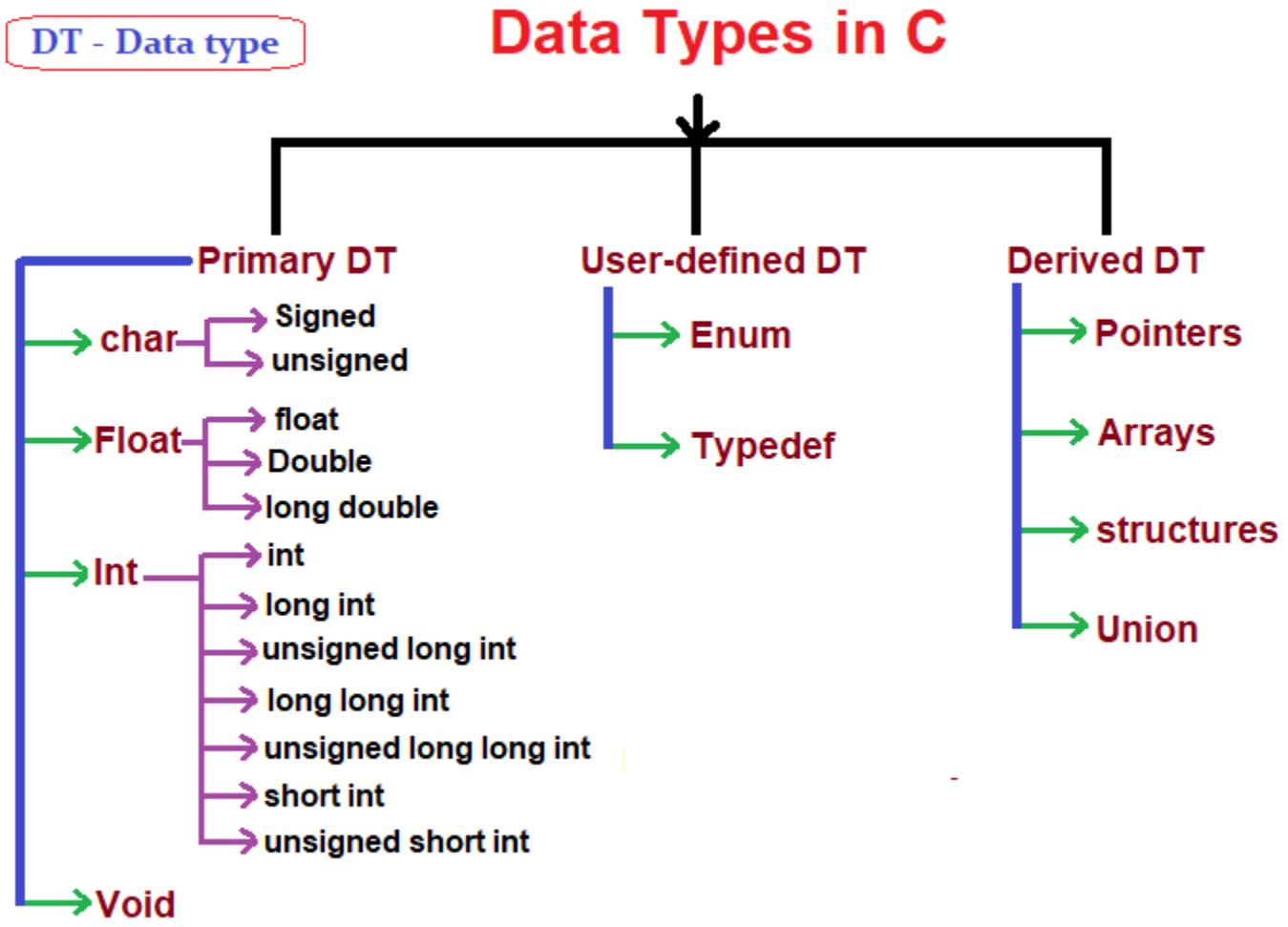
- When selecting DS, must perform below steps:
  1. Analysis of the problem to determine basic operations like insert/delete/search a data in DS
  2. Quantify the resource constraints for each operation
  3. Select DS that best meets these requirements
- When developing an application:
  - First – data and operations, that are to be performed
  - Second – representation of data
  - Third – implementation of that representation

Data Types

- A data type is a classification of data, which can store a specific type of information.
  - Primitive data types
  - Non-Primitive data types

- Primitive data types are predefined, supported by C language. e.g. int, char, float, double
- Non-Primitive data types are not defined by C language, but are created by the programmer. e.g. Stack, Queue, Linked List, Tree, Graph
- They are created using the basic data types.

Data Types


DT - Data type
Data Types in C
Primary DT
User-defined DT
Derived DT
char
Signed
unsigned
Float
float
Double
long double
Int
int
long int
unsigned long int
long long int
unsigned long long int
short int
unsigned short int
Void
Enum
Typedef
Pointers
Arrays
structures
Union

Types of Data Structure

1. **Arrays**

- An array is a data structure consisting of a collection of similar data elements, each identified by one array index.
- The elements in array are stored in consecutive memory locations.
- Array Syntax: int marks[5];

Elements	marks[0]	marks[1]	marks[2]	marks[3]	marks[4]
Array	80	60	78	56	87
Index	0	1	2	3	4

Types of Data Structure

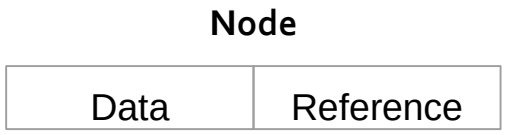
1. **Arrays**

- Limitations of Arrays:
  - Fixed size
  - Data elements are stored in contiguous memory locations which may not be available always!
  - Adding and removing of elements is tough because of shifting the elements from their positions.

Types of Data Structure

2. **Linked List**

- Consisting of a group of nodes which together represent a sequence.
- Under the simplest form, each node is composed of a data and a reference (in other words, a link) to the next node in the sequence.

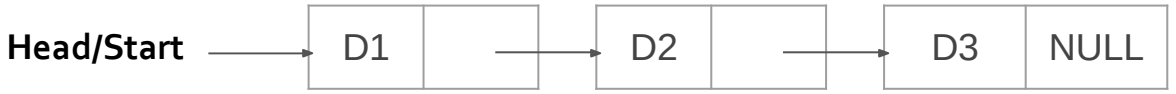


- This structure allows for efficient insertion or removal of elements from any position in the sequence.

Types of Data Structure

2. **Linked List**

- A linked list whose nodes contain two fields:
  - an integer value and a link to the next node.

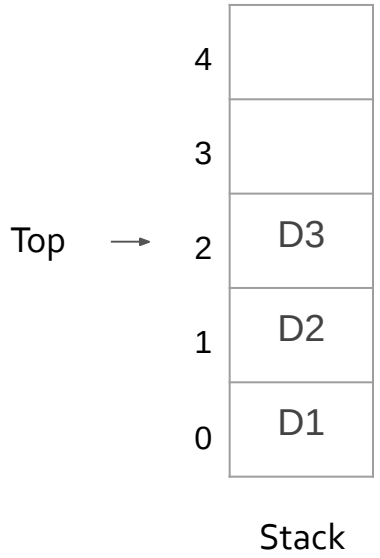


- The last node is linked to a terminator (NULL pointer) used to signify the end of the list.
- Advantage:
  - Provides quick insert and delete operations
- Disadvantage:
  - Slow search operations and requires more memory space.

Types of Data Structure

3. **Stacks**

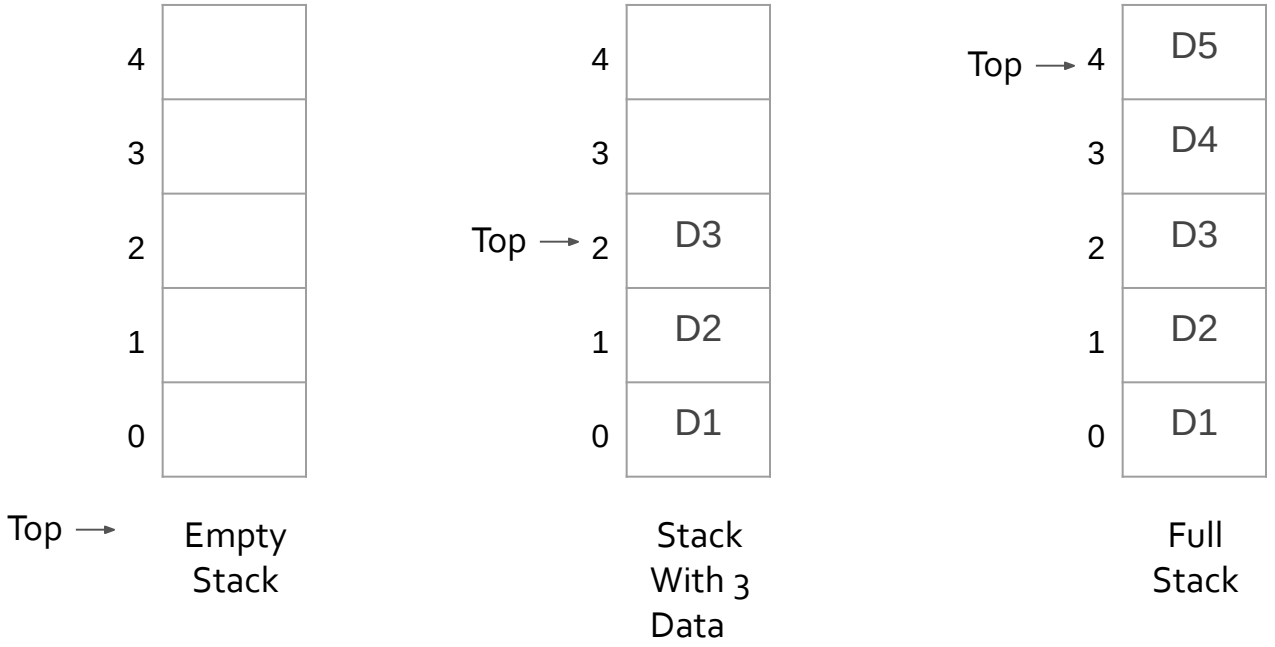
- Stack can be represented as a linear array.
- Every stack has a variable TOP associated with it, to store the address of the topmost element of the stack.



Types of Data Structure

3. **Stacks**

- A stack is a last in, first out (LIFO) data structure.
- IfTOP = NULL, then it indicates stack is empty
- IfTOP = MAX, then it indicates stack is full.



Types of Data Structure

3. **Stacks**

- Elements are removed from the stack in the reverse order to the order of their addition:
  - therefore, the lower elements are those that have been on the stack for longest time.

- Advantage: last-in first-out (LIFO) access.
- Disadvantage: Slow access to other elements.

Types of Data Structure

4. **Queue**

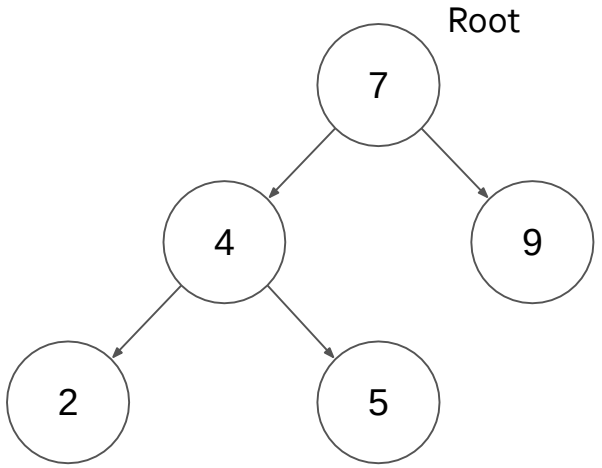
- Queue is a data structure in which data can be added to one end and retrieved from the other.
- You can think of it as a line in a grocery store.
- The first one in the line is the first one to be served. Just like a queue.
- A queue is also called a FIFO (First In First Out) to demonstrate the way it accesses data.

- Advantage: Provides first-in, first-out data access.
- Disadvantage: Slow access to other items.

Types of Data Structure

5. **Trees**

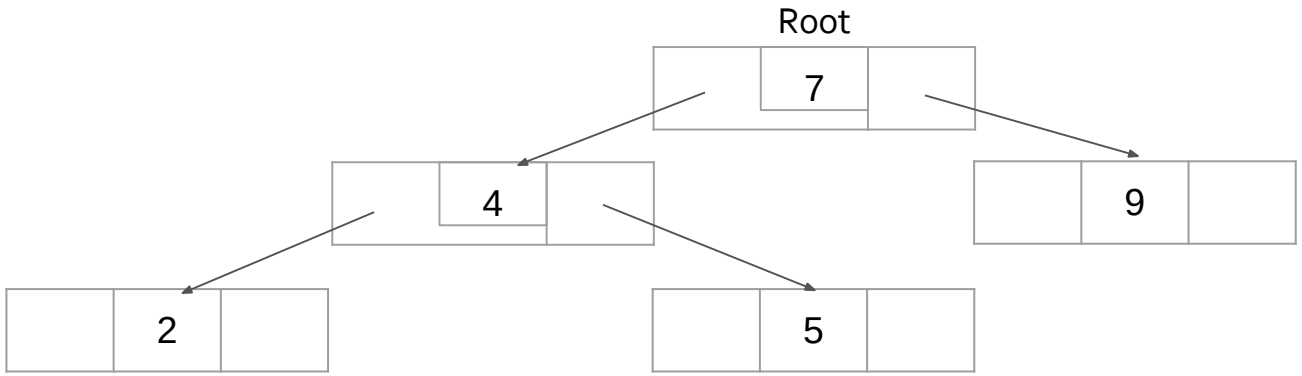
- A tree is a widely used data structure that simulates a hierarchical tree structure with a set of linked nodes.
- Every tree has a root element pointed by a root pointer.
- If root = NULL, tree is empty.
- A simple unordered tree:



Types of Data Structure

5. **Trees**

- In binary tree every node contains a left pointer, a right pointer and a data element.

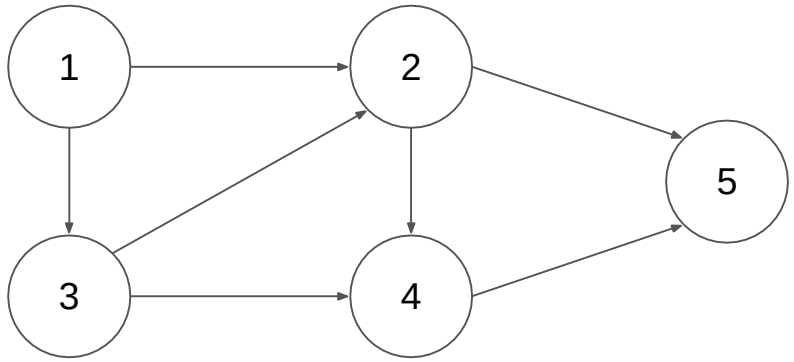


- Advantage: Provides quick search, insert, delete operations.
- Disadvantage: Complicated deletion algorithm.
- Example: Family Tree

Types of Data Structure

6. **Graph**

- A graph is a data structure that is meant to implement the graph concepts from mathematics.
- It is basically a collection of vertices(nodes) and edges that connect these vertices.



Types of Data Structure

6. **Graph**

- A graph is often viewed as a generalization of the tree structure, where complex relationship can be represented.

- Advantage: Best models real-world situations.
- Disadvantages: Some algorithms are slow and very complex.

Types of Data Structure

1. **Linear Data Structure**
   - Elements are stored sequentially
   - We can traverse either forward or backward
   - Examples: Arrays, Stacks, Queues, Linked list

2. **Non-linear Data Structure**
   - Not stored in sequential order
   - Branches to more than one node
   - Can't be traversed in a single run
   - Examples: Tree, Graphs

Abstract Data Type

- ADT is the way we look at a Data Structure, focusing on what it does and ignoring how it does its job.
  - In C for eg. FILE
  - Examples: Stack, Queue
- Implementation is hidden
- Whenever end user uses Stack, he is concerned about only the type of data and operations that can be performed on it.
- Fundamentals of how the data is stored, is invisible to users.
- They will have push(), pop(), etc. functions only.

Algorithm

- "A formally defined procedure for performing some calculation"
- An algorithm provides a blueprint to write a program to solve a particular program.
- Algorithm is a set of instructions that solve a problem.
- It is possible to have multiple algorithms to tackle the same problem, but choice depends on time and space complexity.

Key features of an Algorithm

- Any algorithm will be having finite steps.
- Algorithm exhibits three key features:
  1. Sequence
  2. Decision
  3. Repetition

Algorithm

- **Exercise:**

  AIM: Write an algorithm to find sum of first N numbers.

  Step1: Input N

  Step2: SET ITR = 1, SUM = 0

  Step3: Repeat Step 4 and 5 While ITR<=N

  Step4: SET SUM = SUM + ITR

  Step5: SET ITR = ITR + 1

  Step6: PRINT SUM

  Step7: END

Time and Space Complexity

- The analysis of algorithms is the determination of the number of resources (such as time and storage) necessary to execute them.

- **Time Complexity** of an algorithm is basically the running time of a program, as a function of the input size.

Time and Space Complexity

- **Space Complexity** of an algorithm is the amount of computer memory that is required during the program execution, as a function of input size.
- The space needed by a program depends on:
  1. Fixed Part: that varies from problem to problem. It includes instruction, constants, variables etc.
  2. Variable Part: that varies from problem to problem. It includes space needed for recursion, dynamic value allocation to variables.

Best, Worst and Average case

- Best, Worst and Average cases of a given algorithm express what the resource usage is at least, at most and on average, respectively.

- In real-time computing, the worst-case execution time is often of particular concern since it is important to know how much time might be needed in the worst case to guarantee that the algorithm will always finish on time.

Best-case performance for algorithm

- The term best-case performance is used in computer science to describe the way of an algorithm behaves under optimal conditions.
- For example, the best case for a simple linear search on a list occurs when the desired element is the first element of the list.

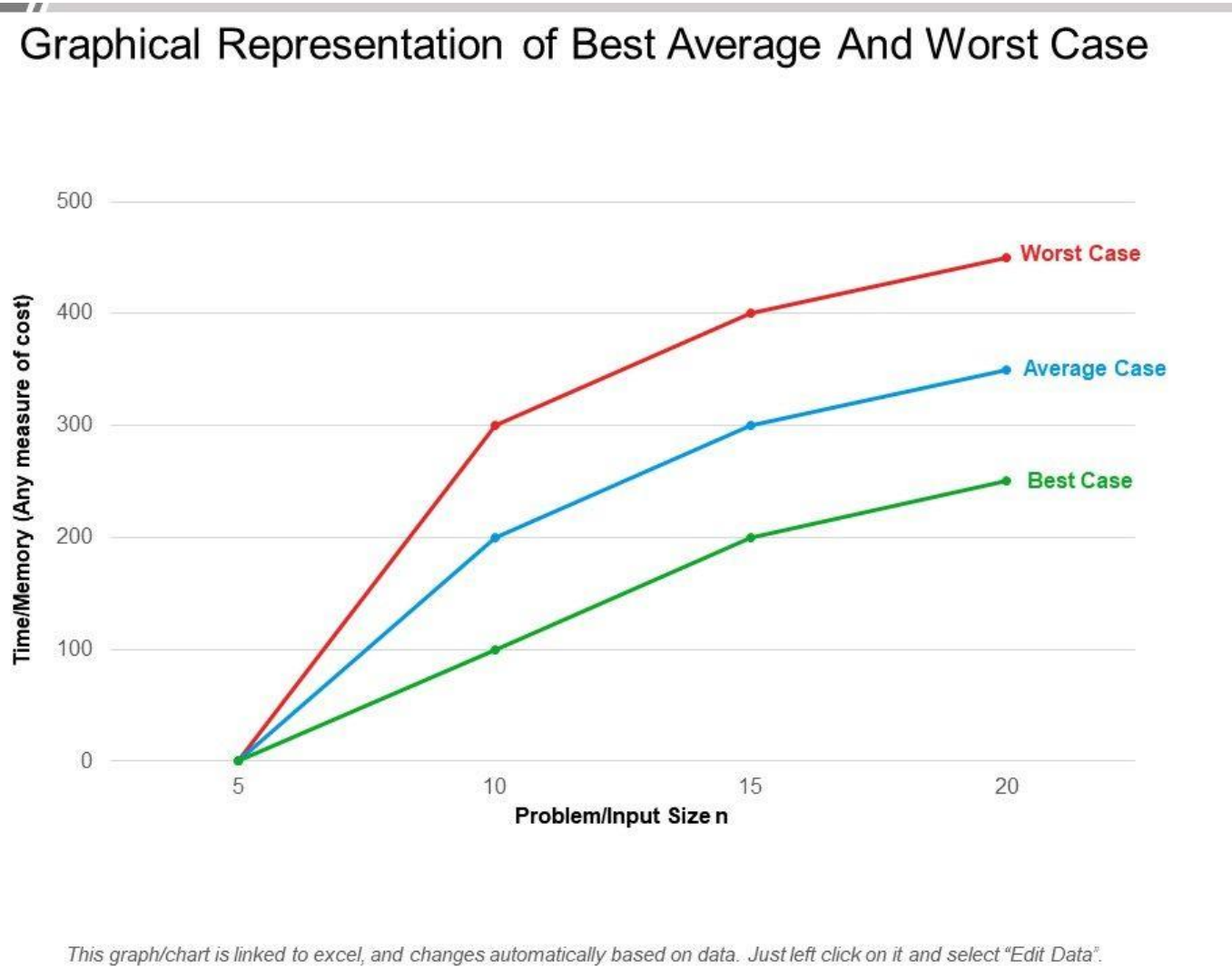
Worst-case performance for algorithm

- This denotes the behavior of the algorithm with respect to the worst-possible case of the input instances.
- Worst-case running time of an algorithm is an upper bound on the running time for any input.
- This provides an assurance that this algorithm will never go beyond this time limit.

Average-case performance for algorithm

- Running time of an algorithm is an estimate of the running time for an 'average' input.
- It specifies the expected behavior of the algorithm when the input is randomly drawn from a given distribution.

Graphical Representation of Best, Average and Worst Case



Time-Space Trade-off

- There can be more than one algorithm to solve a particular problem.
- One may require less memory space and one may require less CPU time to execute.
- Hence, there exists a time-space trade-off among algorithms.
- So, if space is a big constraint, then one might choose a program that takes less space at the cost of more CPU time.
- On the contrary, if time is a major constraint then one might choose a program that minimum time to execute at the cost of more space.

Expressing Time & Space Complexity

- Time & Space Complexity can be expressed using function f(n), where n is the input size
- Required when:
  1. We want to predict the rate of growth of complexity as size of the problem increases
  2. Multiple algorithms available, but we need to find most efficient

Expressing Time & Space Complexity

- Most widely used notation to express this function f(n) is Big-Oh notation (O).
- It provides upper bound for the complexity.

Algorithm Efficiency

- If a function is linear (without any loop or recursion), the running time of that algorithm can be given as the number of instructions it contains.
- Running time may vary because of loops or recursive functions.

Linear Loops

- To calculate the efficiency with single loop, we first need to determine number of times the loop will be executed.

```
for(i=0; i<100;i++)
        statement block;
```

- Here loop factor is 100. Efficiency is directly proportional to the number of iterations.
- The general formula: f(n)=n

```
for(i=0; i<100;i+=2)
        statement block;
```

- Here, the number of iterations are just half the number of loop factor.
- f(n) = n/2

Nested Loop

- Loops that contain inner loop.
- In order to analyze nested loops, we need to determine the number of iterations each loop completes.
- No. of iterations in inner loop X No. of iterations in outer loop = Total no. of iterations

Quadratic Loop

- Number of iterations in inner loop is equal to number of iteration in outer loop.

```
for(i=0; i<10;i++)
        for(j=0; j<10;j++)
                statement block;
```

- Generalized formula: $f(n) = n^2$

Dependent Quadratic Loop

- Number of iterations in inner loop is dependent on outer loop.

```
for(i=0; i<10;i++)
        for(j=0; j<=i;j++)
                statement block;
```

- The inner loop will execute as: 1 + 2 + . . . + 10 = 55 (Average is 5.5)
- In general terms: (n+1)/2
- Inner loop will iterate number of iterations in outer loop plus 1 divided by 2.
- So, for overall of this: f(n) = n (n+1) / 2

Big O Notation

- Limitations of Big O Notation
  - Many algorithms are hard to analyze mathematically
  - Big O analysis only tells us how the algorithm grows with the size of the problem, not how efficient it is
  - For Big O Notation, $O(n^2)$ and $O(1000000n^2)$ are equal, but in real time this may be a serious concern

Next Step

- Unit#2:
- Linear Data Structures & their representation
  - Array
  - Stack
  - Queue
  - Linked List

Thank You.