

DS

MODULE-: DATA STRUCTURES



Rohini Desai

VSIT

Vidyalankar School of
Information Technology

NAAC ACCREDITED COLLEGE

Vidyalankar School of
Information Technology
Wadala (E), Mumbai
www.vsit.edu.in


Certificate

This is to certify that the e-book titled "Data Structures" comprises all elementary learning tools for a better understating of the relevant concepts. This e-book is comprehensively compiled as per the predefined eight parameters and guidelines.



Signature
Prof. Rohini Desai
Assistant Professor
Department of IT

Date: 27-07-2017

 **DISCLAIMER:** *The information contained in this e-book is compiled and distributed for educational purposes only. This e-book has been designed to help learners understand relevant concepts with a more dynamic interface. The compiler of this e-book and Vidyalkar Institute of Technology give full and due credit to the authors of the contents, developers and all websites from wherever information has been sourced. We acknowledge our gratitude towards the websites YouTube, Wikipedia, and Google search engine. No commercial benefits are being drawn from this project.*

Contents

Introduction: Data and Information, Data Structure, Classification of Data Structures, Primitive Data Types, Abstract Data Types, Data structure vs. File Organization, Operations on Data Structure, Algorithm, Importance of Algorithm Analysis, Complexity of an Algorithm, Asymptotic Analysis and Notations, Big O Notation, Big Omega Notation, Big Theta Notation, Rate of Growth and Big O Notation.

Array: Introduction, One Dimensional Array, Memory Representation of One Dimensional Array, Traversing, Insertion, Deletion, Searching, Sorting, Merging of Arrays, Multidimensional Arrays, Memory Representation of Two Dimensional Arrays, General Multi-Dimensional Arrays

Sparse Arrays, Sparse Matrix, Memory Representation of Special kind of Matrices, Advantages and Limitations of Arrays.

Recommended Books

References:

- A Simplified Approach to Data Structures
- An Introduction to Data Structure with Applications
- Data Structure and Algorithm
- Schaum's Outlines Data structure

Prerequisites and Linking

Unit I	Pre-Requisites	Sem. II	Sem. III	Sem. IV	Sem. V	Sem. VI
Introduction	-	OOPS	-	-	-	-

UNIT 1- Introduction

Introduction

☐ Data and Information

The term data is derived from the word datum. Data is plural of datum. The term data refers to the value or simply set of values that are raw and unorganized. Data may be simple, random, and useless until it is organized. Data is valuable raw material which can be different forms such as numbers, words, alphabets etc. When data is processed i.e. organized, structured, and presented in a meaningful context so that it becomes useful for decision making and understanding, it becomes information. Data and information are interchangeable terms, but technically, no conclusion can be drawn from data. Information play a vital role in decision making which is obtained from data. Indirectly data is an important entity for decision making. Example Students marks in a subject are treated as a data item. Whereas class's average marks in the same subject is information which can be calculated from the data.

Data Structure and Classification

☐ Data Structure

- A data structure is a way of storing the data in computer's memory so that it can be used efficiently. A data structure is a logical/ mathematical model of organization of data. The choice of data structure begins with the choice of an Abstract Data Type(ADT) such as array.

☐ Classification of Data Structure

- Linear and Non- linear Data Structures
 - Static and Dynamic Data Structures
 - Homogeneous and Non-Homogeneous Data Structures
- ☐ Linear Data Structure: The elements in a linear data structure form a linear sequence. Example of linear data structures are : Array, Linked list, Queue, Stack etc.
- ☐ Non-Linear Data Structure: The elements in a non-linear data structure do not form any linear sequence. Example Tree and Graph.
- ☐ Static Data structure: Static data structures are those whose memory occupation is fixed. The memory taken by these data structures cannot be increased or decreased at run time. Example of the static data structure is an array. The size of an array is declared at the compile time and this size cannot be changes during the run time.
- ☐ Dynamic Data structure: Dynamic data structure are those whose memory occupation is not fixed. The memory taken by these data structures can be increased or decreased at run time. Example of the dynamic data structure is linked List. The size of the linked list can be changed during the run time.
- ☐ Other data structures like stack, queue, tree, and graph can be static or dynamic depending on, whether these are implemented using an array or a linked list.
- ☐ Homogeneous Data structure: Homogenous data structures are those in which data of same type can be stored. Example is array.
- ☐ Non-Homogeneous Data structure: Non-Homogenous data structures are those in which data of different type can be stored. Example is linked list.

Data Types

❑ Primitive Data types

- Primitive data types are also known as predefined or basic data types. These data types are different for different languages. For example in C language, the primitive data types for storing the integer values are int, long, short and for storing the real values are float, double and long double.

❑ Abstract Data Types(ADT)

- Abstract stands for considering apart from the details specification or implementation. Abstraction: refers to the act of representing the essential features without including the details (hiding the details same as OOPS). Encapsulation: providing data and operations on the data in a single unit (same as OOPS).

Data Structure Vs File Organization

- File organization is a study of storing the data records into files. Various file organization techniques are Sequential File Organization, Random File Organization and Indexed Sequential File Organization. These techniques differ in record sequencing and retrieval method. A data structure and file organization differ in following aspects:
 - In Implementation
 - In Access methods
 - Data structure is thought of as main memory (RAM) and file organization are auxiliary storage (Disk).

Operation on Data Structures

- Insertion- Adding a new element to a data structure
- Deletion- removing an element from a data structure
- Traversing- Accessing each element exactly once in order to process it.
- Searching- Finding the position of the desired element.
- Merging- Combining two or more list into a single list
- Splitting- Dividing a single list into two or more list.
- Copying- Creating a duplicate copy of a list
- Sorting- Arranging the elements in ascending and descending order.

Algorithm

The computer is a manmade machine which does not have any decision making capabilities. It only follows the instruction given by its user. Instructions given to the computer to solve a particular problem are known as algorithm.

- Input: An algorithm must take some inputs that are required for the solution of a problem.
- Process: An algorithm must perform certain operations on the input data which are necessary for the solution of the problem.
- Output: An algorithm should produce certain output after processing the inputs.
- Finiteness: An algorithm must terminate after executing certain finite number of steps.
- Effectiveness: Every step of an algorithm should play a role in the solution to the problem. Also each step must be unambiguous, feasible and definite.

Importance of Algorithm Analysis

There are two basic requirements to solve any particular problem.
These are:

- Data
- Instructions to manipulate data i.e. Algorithm

To efficiently process the data, we decide the suitable data structure. Once the data structure is decided, the concentration is given to the choice of algorithm. The choice of the algorithm can be made by considering the following factors:

- **Programming requirements of an algorithm** - An algorithm must use the features supported by the programming language in which it is to be implemented. If we have designed an algorithm that is optimal but does not support the programming language features then it is of no use as compared to other one which may not be optimal but supports the programming language features. An algorithm is of no use if it cannot be programmed and executed. Therefore, an algorithm must satisfy the programming features of the language.
- **Space requirement of an algorithm** - To execute any program, space is needed for various reasons:
 - Space required by data: This includes space required to store variables and constants which is generally fixed. Sometimes, space is allocated dynamically and this space is not fixed.
 - Space required by instructions: This is the space required to store the instruction sets. This space remains unchanged as the instructions of the program do not change during run time.
- **Time requirement of an algorithm** - Each algorithm takes some amount of time to execute.
 - Sometimes it is necessary to know in advance, the time required to execute the program to see whether it is within the acceptable limits or not.
 - There can be several different solutions for a particular problem each with different time requirement, one can choose the most optimal solution.

Complexity of an algorithm

Complexity is the time and space requirement of the algorithm. If time and space requirement of the algorithm is more, then complexity of the algorithm is more and if time and space requirement of the algorithm is less, of that algorithm is less.

Out of two factors, time and space requirement of the algorithm is not a very important factor because it is available at very low cost. Only the time requirement of the algorithm is considered an important factor to find the complexity. Because of the importance of time in finding the complexity, it is sometimes termed as time complexity.

The complexity of the algorithm is dependent upon the input size, still complexity can be divided into three types:

- **Worst case complexity** – If the running time of the algorithm is longest for all the inputs then the complexity is called worst case complexity. In this type of complexity, the key operation is executed maximum number of times. Worst case is the upper bound of complexity and in certain applications like traffic control, medical surgery.

- **Best case complexity** - If the running time of the algorithm is shortest for all the inputs then the complexity is called best case complexity. In this type of complexity, the key operation is executed minimum number of times.
- **Average case complexity**-If the running time of the algorithm fall between the worst and best case and the complexity is called average case complexity.

Asymptotic Analysis and Notations

Complexity can be defined as the rate at which the storage or time requirement grows as a function of the problem size.

The type of analysis known as asymptotic analysis is used to simplify the analysis of running time by removing the details which may be affected by hardware or compilers used.

To measure the complexity of an algorithm, various asymptotic notations can be used such as,

- Big O Notation
- Big Omega(Ω) Notation
- Big Theta(θ) Notation
- Little Omega Notation
- Little Theta Notation

<https://www.youtube.com/watch?v=OpebHLAf99Y>

Big O Notation

Big O notation is upper bound asymptotic notation. This means, a function means, a function $f(x)$ is Big O of function $g(x)$ and there exists the positive constants c and n_0 such that

$c \cdot g(n) \geq f(n)$ for all $n \geq n_0$

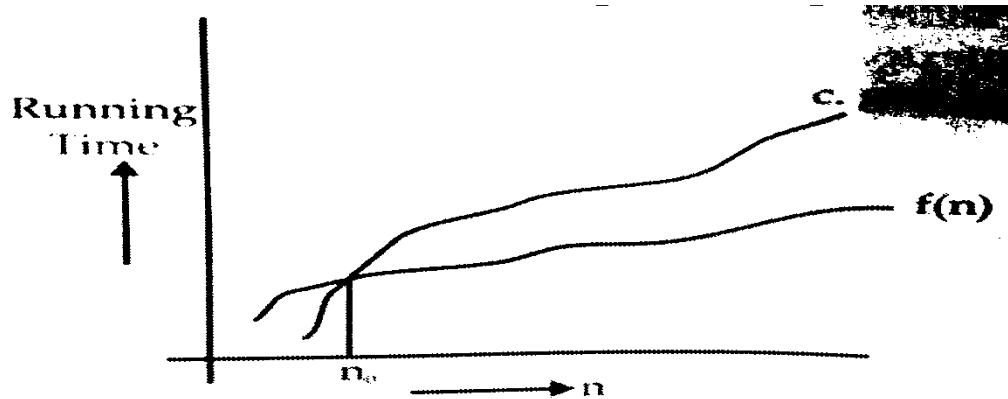
Here, $f(x)$ and $g(x)$ are the functions of non-negative integers.
For example, consider the functions,

$$f(n) = 2n + 6$$

$$g(n) = n$$

Taking constants $c = 4$ and $n_0 = 3$,

$$f(n) \leq c \cdot g(n) \text{ where } n \geq n_0$$



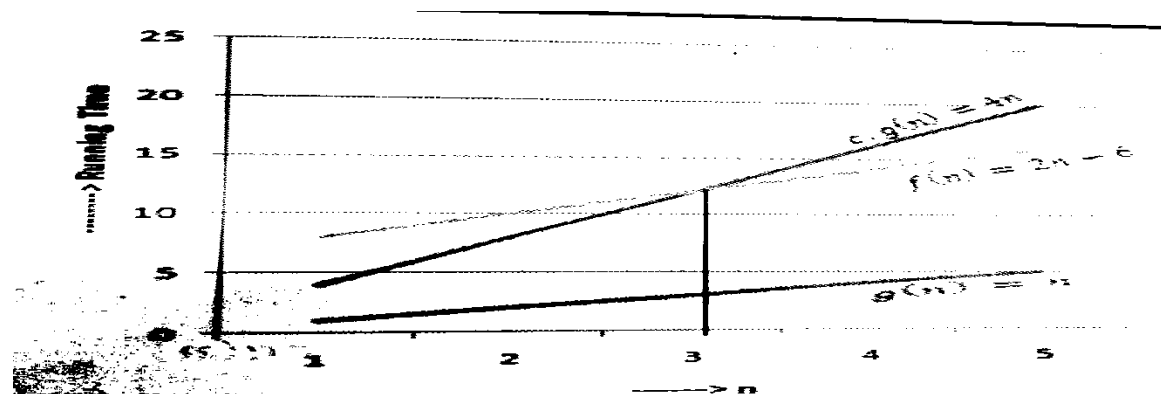
Big O Notation $c \cdot g(n) \geq f(n)$

In general, Big O notation drops all the constant and lower order terms.

For example, $5n \log_2 n$ is $O(n \log_2 n)$

$$5n - 3 \text{ is } O(n)$$

$$n^2 \log_2 n + n^2 + 8n \text{ is } O(n^2 \log_2 n)$$



Complexity = $O(g(n)) = O(n)$

Even $5n - 3$ is $O(n^3)$ but always an approximate as small as possible order of the function is taken.

The complexity of any algorithm may be,

Logarithmic expressed as $O(\log_2 n)$

Linear expressed as $O(n)$

Quadratic expressed as $O(n^2)$

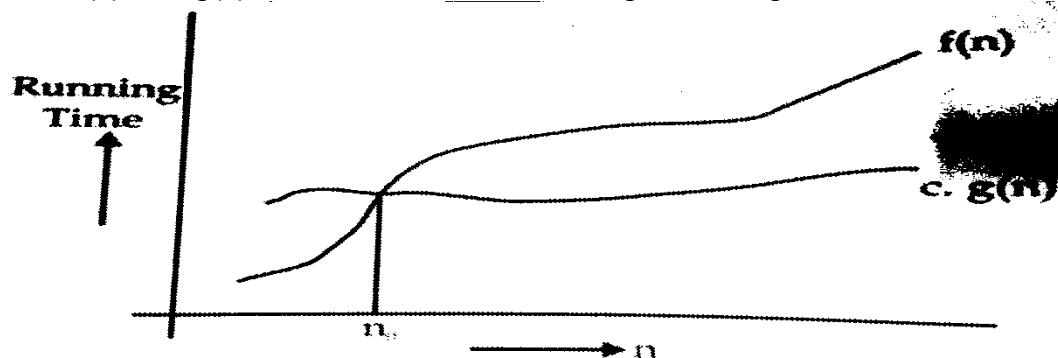
Polynomial expressed as $O(n^k)$ $k \geq 1$

Exponential expressed as $O(a^n)$ $a > 1$

Big Omega Notation

Big Omega is asymptotically lower bound notation. This means a function $f(x)$ is Big Omega of function $g(x)$ and there exists two positive constants c and n_0 such that $c \cdot g(n) \leq f(n)$ for all $n \geq n_0$.

Here, $f(x)$ and $g(x)$ are the functions of non-negative integers.



Big Omega notation $f(n) \geq c g(n)$

Big Omega is used to express the best case running time or lower bounds of the algorithmic problems. The function $g(n)$ is only a lower bound on $f(n)$.

For example,

$3n + 2$ is $\Omega(n)$ as $3n + 2 \geq 3n \quad \forall n \geq 1$

$100n + 7$ is $\Omega(n)$ as $100n + 7 \geq 100n \quad \forall n \geq 1$

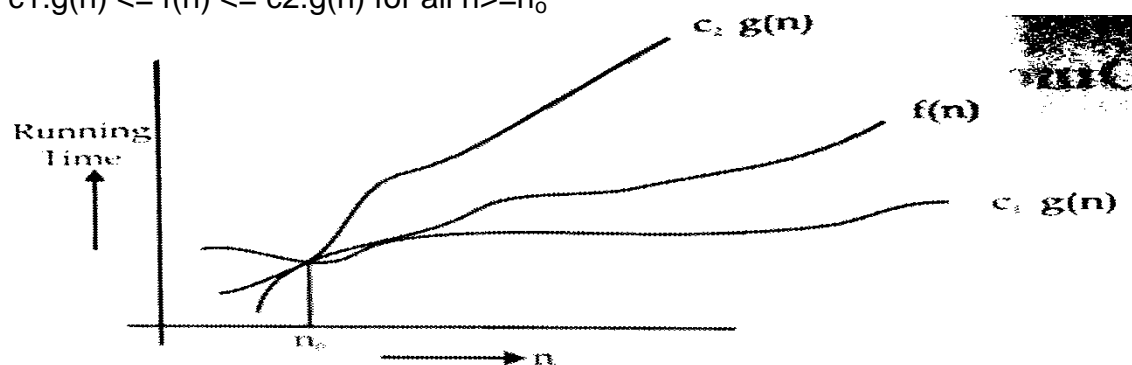
$100n^2 + 3n + 2$ is $\Omega(n^2)$ as $100n^2 + 3n + 2 \geq n^2 \quad \forall n \geq 1$

$3n + 2$ is $\Omega(1)$ as $3n + 2 \geq 1 \quad \forall n \geq 1$ but we never take $\Omega(1)$, instead we take nearest function, so $\Omega(n)$

Big Theta Notation

Big Theta is asymptotically a tight bound notation. This means a function $f(x)$ is Big Theta function $g(x)$ and there exists three positive constants c_1 , c_2 and n_0 such that

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$



Big Theta notation $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

Here, $f(x)$ and $g(x)$ are the functions of non-negative integers.

It may be noted that $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

For example,

$3n + 2$ is $\Theta(n)$ as $3n + 2 \geq 3n \quad \forall n \geq 2$ and $3n + 2 \leq 4n \quad \forall n \geq 2$

Here, $c_1 = 3$, $c_2 = 4$ and $n_0 = 2$

$100n^2 + 3n + 2$ is $\Theta(n^2)$ as $100n^2 + 3n + 2 \geq n^2 \quad \forall n \geq 5$ and

$100n^2 + 3n + 2 \leq 101n^2 \quad \forall n \geq 5$

Here $c_1=1$, $c_2= 101$ and $n_0 =5$

Theta notation is more precise as compared to Big O and Big Omega Notation.

Rate of Growth of Complexity with Input Size

The complexity function $f(n)$ increases as the size of input i.e. n increases. This rate of growth of $f(n)$ is what that we want to examine. It is accomplished by comparing $f(n)$ with some standard function. These standard functions are $\log_2 n$, n , $n \log_2 n$, n^2 , n^3 , 2^n

The function like 2^n involving n as an exponent is known as exponential function.

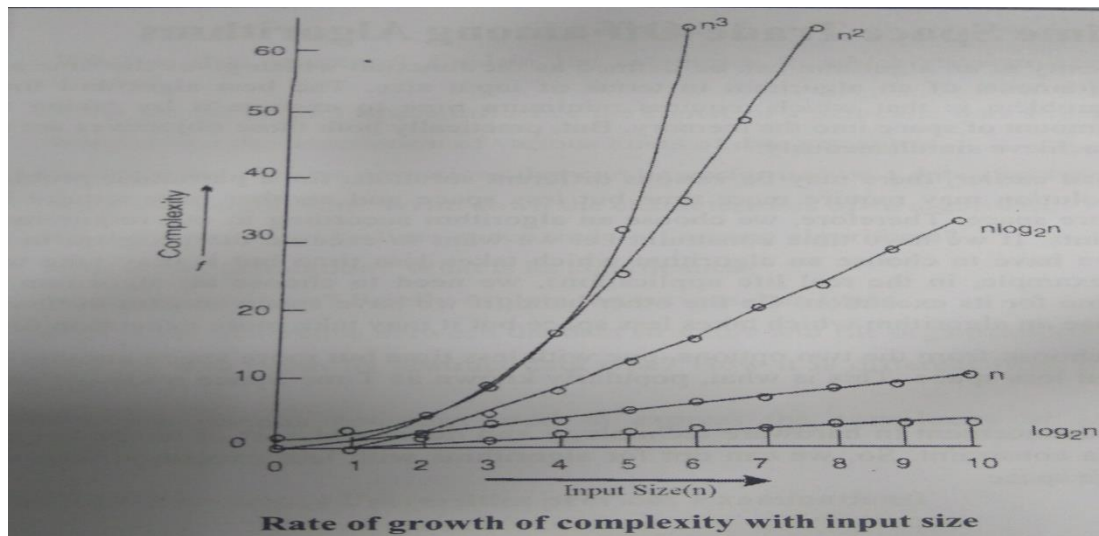
The functions like n^2 , n^3 , n are known as polynomial functions.

The functions $\log_2 n$, $n \log_2 n$ are known as logarithmic functions.

Rate of growth of some standard functions is given in the table below:

N	$O(\log_2 n)$	$O(n)$	$O(n \log_2 n)$	$O(n^2)$	$O(n^3)$
8	3	8	24	64	512
16	4	16	64	256	4096
32	5	32	160	1024	32768
64	6	64	384	4096	262144
128	7	128	896	16384	2097152
256	8	256	2048	262144	67108864

Rate of growth of some standard functions with the size of input



As it is visible in the table, logarithmic function $\log_2 n$ increases most slowly and cubic function n^3 increases very fast. The exponential function 2^n most rapidly as compared to other functions such as logarithmic, linear, quadratic or cubic functions. To compare $f(n)$ with these standard functions we use Big O notation to express the complexities of the algorithm.

Rate of growth of standard functions is plotted in the figure shown below:

The complexity of some popular algorithms is given below in the table below:

Algorithm	Complexity
Linear Search	$O(n)$
Binary search	$O(\log_2 n)$
Bubble Sort	$O(n^2)$
Quick Sort	$O(n \log_2 n)$
Insertion Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Merge Sort	$O(n \log_2 n)$
Heap Sort	$O(n \log_2 n)$

Complexity of some sorting and searching algorithms

Array

- An array is a basic data structure which can be used as building block for many other complex data structures. An array is a linear(adjacency among the elements can be identified in a sequence) collection of finite number of homogenous(all elements should be of same data type) data elements. When an array is stored in memory its elements gets stored in consecutive memory locations. The elements of the array are referenced with an index set consisting of consecutive numbers
- Consider the following array S

100	200	300	400	500	600	700	800	900	1000
1	2	3	4	5	6	7	8	9	10

- The number of elements in the array is called the size or the length of the array which can be calculated using the formula
 - Size of array = $ub - lb + 1$ where ub is upper bound and lb is lower bound of an array. In the above example
 - Size of S = $10 - 1 + 1 = 10$
- The elements of an array are referenced by using subscripts which is taken from the index set like $s[4]$ OR $s[9]$ etc. It should be noted that subscript outside the index set cannot be used as a valid subscript

Memory Representation of An Array

- The main advantage of an array is that it allows us to refer any element directly by calculating its address. We have to keep a track of the address of the first element which is also known as the base address denoted by $Base(\text{Array Name})$ that is $Base(S)$.
- Hence address of any other element will be
 - $Loc(S_k) = Base(S) + w(k - lb)$ where
 - S_k is the k^{th} element of the array
 - $Loc(S_k)$ is the address or location of S_k
 - $Base(S)$ is the address of the first element of the array
 - w is the size of the data type of the array element in terms of memory
 - $(k - lb)$ is the distance of the S_k element from the start of the array i.e lb or lower bound of the array

- Consider our example of array S having lb = 1 and ub = 10. Array S stores integers and integers takes 2 bytes of memory hence our w = 2 and assume the first element of the array is stored at 1500, thus the consecutive array elements will be 1502, 1504, 1506 etc .
- Hence address of S[8] will be
 - $Loc(S_k) = Base(S) + w(k - lb)$
 - $Loc(S_8) = 1500 + 2 * (8 - 1)$
 $= 1500 + 2 * 7 = 1514$
- So the element of the array S at index 8 will be stored at location 1514
- Consider another example
 - Base(S) = 1532
 - lb = 11
 - w = 4
 - k = 25 Answer = 1588

Various Operations Associated With Array

Traversing Algorithm

1. Set i=lb (lb=lower bound)
 2. Repeat step 3 and 4 while i<=ub (ub=upper bound)
 3. Print S[i]
 4. Set i=i+1
- [End loop]
5. Exit

Insertion Algorithm – Inserting a New element at the kth position in the array S of size n

1. Repeat steps 2 and 3 For i=n To k
 2. Set S[i+1]=S[i]
 3. Set i=i-1
- [End Loop]
4. Set s[k]= New
 5. Set n=n+1
 6. Exit

Deletion Algorithm – Deleting an element from the kth position in the array S of size n

1. Repeat Steps 2 and 3 For i =k to n-1
 2. Set S[i] =S[i+1]
 3. Set i=i+1
- [End Loop]
4. Set n=n-1
 5. Exit

Linear Search

- In the linear search technique we start comparing the desired element with the element at the first position of the array, then with the element at the second position in the array.
- This procedure continues until the desired element is found or we reach at the end of the array.

- If we don't find the desired element till the end of the array it is concluded that the array is not having the desired element.
- If we find the desired element then we display the position in the array where the element is located.

Linear Search Algorithm

1. Repeat Steps 2 and 3 For $i = 1$ to n
2. If $S[i] = \text{Data}$ then
Print "Element is found at position" : i
Exit
[End If]
3. Set $i=i+1$ [End Loop]
4. Print: "Desired element Data is not found in the array"
5. Exit

Binary Search

- It is applied on the array where the elements are sorted alphabetically or numerically.
- The first step is to find the array index of the middle element.
- Then we compare the array element present at the middle with the desired element.
- If the element at the middle index position is the desired element then stop searching. Otherwise after examining the element at the middle index position we decide which half portion of the array may contain the desired element.

<https://www.youtube.com/watch?v=j5uXyPJ0Pew>

Binary Search Algorithm

1. Set Start = lb , End =ub
2. Repeat Steps 3 to 5 while Start < End
3. Set Middle = Integer ((start+end)/2)
4. If S[middle] =Data then
 Print: "Element is found at position": Middle
 Exit
 [End If]
5. If S[Middle]< Data Then
 Set Start =Middle + 1
 Else
 Set End = Middle -1
 [End If]
 [End Loop]
6. Print: Element does not exit in the array"
7. Exit

Difference between Linear Search and Binary Search

Linear Search	Binary Search
The elements of the array need not to be in sorted order.	The elements of the array must be in any sorted order.
The complexity of linear search is $O(n)$	The complexity of binary search is $O(\log_2 n)$
Linear search is accomplished by comparing desired element with each element of the list(array) starting from 1 st element of the array	In binary search, the desired element is compared with middle element and selecting the half portion of the list in which element may be present. This procedure of halving the list is repeated till the element is found or we conclude that element is not present.
Linear search can be applied on any linear data structure even if the elements of the data structure do not occupy the contiguous memory locations.	The binary search can be applied only on array because the elements of array are in contiguous memory locations.

Complexity of Linear Search Algorithm – In a searching algorithm the key operations is comparison. Thus the complexity of the algorithm can be measured by counting the number of comparisons performed in that algorithm

1. Worst Case: In this case we find the desired element at the last position or no element.
2. Average Case: In this case probability of finding the desired element at any position in the array.

Bubble Sort

1. Repeat For p=1 to n-1
2. For i=1 to n-p
3. If S[i]>S[i+1] Then
 Exchange S[i] with S[i+1]
 [End If]
- [End Loop]
- [End Loop]
4. Exit

Merging Algorithm

1. Set i=lb1, j=lb2, k=1
2. While i<= ub1 AND j<= ub2
3. If A1[i] < A2[j] Then
 Set A3[k] =A1[i]
 Set i=i+1
 Set k=k+1
Else\
 Set A3[k] =A2[j]
 Set j=j+1
 Set k=k+1
[Else If]
[End Loop]
4. If i> ub1 Then
 While j<= ub2
 Set A3[k]=A2[j]
 Set j=j+1
 Set k=k+1
 [End Loop]
Else If j>ub2 Then
 While i<=ub1
 Set A3[k] =A1[i]
 Set i=i+1
 Set k=k+1
 [End Loop]
[End If]
5. Exit

Multidimensional Array

- These are the arrays in which elements are referenced by using 2 or more subscripts.
- The array whose elements are referenced by 2 subscripts is known as two dimensional array, with 3 subscripts is known as three dimensional array and so on
- In two dimensional array each element of the array is referenced by two subscripts like A[i][j] OR A(i,j) OR A_{i,j}
- There are two different ways to store the two dimensional array either row by row which is called row major order or column by column which is called column major order

- The choice of storing the two dimensional array depends on programming language
- The memory representation of both orders is shown below
- Row Major Order

A[1][1]	A[1][2]	A[1][3]	A[2][1]	A[2][2]	A[2][3]
---------	---------	---------	---------	---------	---------

- Column Major Order

A[1][1]	A[2][1]	A[1][2]	A[2][2]	A[1][3]	A[2][3]
---------	---------	---------	---------	---------	---------

Memory Representation of Two Dimensional Array

- Consider a two dimensional array A of size 5 x 4. Suppose base address of the array is 500 and each array element occupies two memory cells. If the programming language uses row major order for storing array elements then the address of the element A[3][3] is
- Address Of A[i][j] = $\text{Base}(A) + w[c(i-lbr) + (j-lbc)]$
Address of A[3][3] = $500 + 2 * [4 * (3 - 1) + (3 - 1)]$
 $= 500 + 2 * [8 + 2] = 520$
- If the same array is stored in column major order then the address of element A[3][3] is
- Address Of A[i][j] = $\text{Base}(A) + w[(i-lbr) + r(j-lbc)]$
Address of A[3][3] = $500 + 2 * [(3 - 1) + 5 * (3 - 1)]$
 $= 500 + 2 * [2 + 10] = 524$

Example:

- Consider a two dimensional array S[3:6,-4:2]. Here colon is used to separate the lower and upper indices for the dimension and comma is used to separate the dimensions.
In the array S,

Lower index of 1st dimension (lbr) = 3

Upper index of 1st dimension (ubr) = 6

Lower index of 2nd dimension (lbc) = -4

Upper index of 2nd dimension (ubc) = 2

Length of first dimension (number of rows) $r = ubr - lbr + 1 = 6 - 3 + 1 = 4$

Length of Second dimension (number of columns) $c = ubc - lbc + 1 = 2 - (-4) + 1 = 7$

- The above values of r and c can be used as row and column values for any given array i.e. S[4,7].

General Multi-Dimensional Array

- The concept of n-dimensional Array can be extended from one or two dimensional arrays. Total number of elements in an n-dimensional array can be calculated by multiplying the length of each dimension. To reference any element of n-dimensional array n subscript will be required. An n dimensional array B can be declared as

$B[lb_1:ub_1, lb_2:ub_2, lb_3:ub_3, \dots, lb_n:ub_n]$

- Here lb is the lower index and ub is the upper index.

- Consider a three dimensional array A having lb1,ub1 as lower index and upper index of first dimension, lb2 and ub2 as lower index and upper index of second dimension, lb3 and ub3 as lower index and upper index of three dimension. The length of each dimension(l1,l2,l3) respectively will be:
- For Row major order
Address of (A[i,j,k])=Base(A) + w[l₂l₃(i-lb₁)+l₃(j-lb₂)+(k-lb₃)]
- For Column major order
Address of (A[i,j,k])=Base(A) + w[(i-lb₁)+l₁(j-lb₂)+l₁l₂(k-lb₃)]
- Here w is the number of memory cells occupied by each array element and Base(A) is the base address of the array.
- Consider a 3-dimensional array M(-2:1,5:7,3:4) which is a collection of 24 elements. Assuming the base address of the array is 1000 and each element of the array occupies two cells calculate the address of element M(1,6,4) if the elements of the array are stored in row major order and column major order

$$\begin{array}{lll} lb_1=-2 & lb_2=5 & lb_3=3 \\ ub_1=1 & ub_2=7 & ub_3=4 \end{array}$$

$$L_1 = ub_1 - lb_1 + 1 = 1 - (-2) + 1 = 4$$

$$L_2 = ub_2 - lb_2 + 1 = 7 - 5 + 1 = 3$$

$$L_3 = ub_3 - lb_3 + 1 = 4 - 3 + 1 = 2$$

Hence total number of elements = 4 * 3 * 2 = 24
i=1, j=6, k=4, w=2, Base(M)=1000

- In Row Major Order
Address of M[i][j][k] = Base(M) + w[l₂l₃(i-lb₁) + l₃(j-lb₂) + (k-lb₃)]
Address of M[1][6][4] = 1000 + 2 * [3 * 2 * (1 + 2) + 2 * (6 - 5) + (4 - 3)] =
1000 + 2 * (18 + 2 + 1) = 1000 + 42 = 1042
- In Column Major Order
Address of M[i][j][k] = Base(M) + w[(i-lb₁) + l₁(j-lb₂) + l₁l₂ (k-lb₃)]
Address of M[1][6][4] = 1000 + 2 * [(1 + 2) + 4 * (6 - 5) + 4 * 3 * (4 - 3)] =
1000 + 2 * (3 + 4 + 12) = 1000 + 38 = 1038

Sparse Matrix

- A matrix M is said to be sparse matrix if majority of its elements are meaningless. Such kind of matrices contains very few elements which are significant or which contain majority of elements as zero. The below shown matrix is a sparse matrix because major of its elements are zero.

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 0 & 7 \end{bmatrix}$$

- A matrix M is said to be diagonal if all its non-diagonal elements as zero.

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

- A matrix is said to be upper triangular matrix if all the elements below the diagonal are zero.

$$\begin{bmatrix} 2 & 5 & 3 \\ 0 & 4 & 1 \\ 0 & 0 & 7 \end{bmatrix}$$

- A matrix is said to be strictly upper triangular if all the diagonal elements are also zero along with the elements below the diagonal.

$$\begin{bmatrix} 0 & 2 & 9 \\ 0 & 0 & 8 \\ 0 & 0 & 0 \end{bmatrix}$$

- A matrix is said to be lower triangular matrix if all the elements above the diagonal are zero and strictly lower triangular if all the diagonal elements are also zero along with the elements above the diagonal.

$$\begin{bmatrix} 2 & 0 & 0 \\ 4 & 1 & 0 \\ 5 & 8 & 3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 4 & 0 & 0 \\ 5 & 8 & 0 \end{bmatrix}$$

Memory Representation Of Special Kind Of Matrices

- Method Of Linearization – In this technique the non-zero elements of the matrix are stored in one dimensional array. For example

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 4 & 0 & 7 \end{bmatrix}$$

2	4	7
---	---	---

- Method Of Vector Representation – In this technique the non-zero elements are stored in the array along with its row id and column id.

	Row-ID	Column-ID	Elements
V[1]	1	1	2
V[2]	3	1	4
V[3]	3	3	7

Advantages of Array

- Array is the simple kind of data structure which is very easy to implement.
- Address of any element of the array can be calculated easily as elements are stored in contiguous memory locations.
- Array can be used to implement other data structures such as stack, queue, trees and graph.
- If elements of an array are stored in some logical order then binary search can be applied to search an element in the array efficiently.

Limitations of an Array

- Array is the static kind of data structure. Memory used by the array cannot be increased or decreased whether it is allocated at runtime or compile time.
- Only homogenous elements can be stored in the array. Therefore in case we want to store the data of mixed type the array cannot be used.
- Insertion and deletion of elements are very time consuming in array. When new element is to be inserted in the middle of the array then elements are required to move to create the space for new element. If an element is to be deleted from the array, all its preceding elements need to be moved to fill the vacated position of the array.

Questions:

1. What is data structure? Explain the categories in which data structures can be divided.
2. Explain data types in data structures.
3. Explain the memory representation of one dimensional array with example.
4. What are the advantages and disadvantages of binary search over linear search?
5. Explain and write the algorithm for bubble sort technique for sorting an array.
6. Suppose an array $X[7:19]$ having base address as 1500 and each element takes 4 memory cells then find the location of 5th element of array.
7. Consider a two dimensional array $D[5:7, -3:6]$. If the base address of D is 1536 and each element takes 2 memory cells then find the address of $D[6][0]$ element assuming:
 - a. Array D stored in Row major order
 - b. Array D stored in Column major order.
8. Consider a three dimensional array B whose subscript limits are $-2 \leq i \leq 5$, $1 \leq j \leq 3$ and $3 \leq k \leq 7$. If the base address of B is 2000 and each element takes 4 memory cells then find the address of $B[-2, 2, 4]$ and $B[5, 3, 6]$ element assuming:
 - a. Array B stored in Row major order
 - b. Array B stored in Column major order.
9. What is an algorithm? What are the characteristics of an algorithm?
10. Explain the memory representation of two dimensional arrays with example.
11. Explain and write the algorithm for insertion and deletion of any data item from the array.

12. Write a difference between binary search and linear search.
13. Write down an algorithm to merge two sorted arrays into a third sorted array.
14. Suppose an array Y[-2:7] having base address as 1000 and each element takes 10 memory cells then find the location of 3rd element of array.
15. Consider a two dimensional array A of order 6×8 . If the base address of A is 1000 and each element takes 4 memory cells then find the address of A[5][2] element assuming:
 - a. Array A stored in Row major order
 - b. Array A stored in Column major order.
16. Consider a three dimensional array D[-2:1,5:7,3:4]. If the base address of D is 1000 and each element takes 4 memory cells then find the address of D[1][6][4] element assuming:
 - a. Array D stored in Row major order
 - b. Array D stored in Column major order.
17. What do you mean by time-space trade-off among algorithm?
18. Explain the memory representation of three dimensional arrays with example.
19. Explain and write an algorithm for binary and linear search.
20. Explain the following matrices with example:
 - a. Sparse matrix
 - b. Diagonal matrix
 - c. Upper triangular matrix
 - d. Lower triangular matrix.
21. Explain the advantages and limitations of array.
22. Suppose an array A having base address as 2500 and each element takes 8 memory cells and the array's starting index is 9 then calculate the address of element at index 20 in the array.
23. Consider a two dimensional array B whose subscript limits are $5 \leq i \leq 9$ and $3 \leq j \leq 8$. If the base address of B is 1024 and each element takes 1 memory cells then find the address of B[8][6] element assuming:
 - a. Array B stored in Row major order
 - b. Array B stored in Column major order.
24. Consider a three dimensional array A of order $5 \times 4 \times 3$. If the base address of A is 1500 and each element takes 2 memory cells then find the address of A[2][3][1] element assuming:
 - a. Array A stored in Row major order
 - b. Array A stored in Column major order.

Multiple Choice Questions:

1. Two main measures for the efficiency of an algorithm are
 - a. Processor and Memory
 - b. Complexity and Capacity
 - c. Time and Space**
 - d. Data and Space
2. Which of the following data structure is not linear data structure?
 - a. Arrays
 - b. Linked Lists
 - c. Both of above
 - d. None of above**
3. The time factor when determining the efficiency of algorithm is measured by
 - a. Counting Microseconds
 - b. Counting the number of key operations**
 - c. Counting the number of statements
 - d. Counting the kilobytes of Algorithm
4. The space factor when determining the efficiency of algorithm is measured by
 - a. Counting the maximum memory needed by the algorithm**
 - b. Counting the minimum memory needed by the algorithm
 - c. Counting the average memory needed by the algorithm
 - d. Counting the maximum disk space needed by the algorithm
5. Which of the following case does not exists in complexity theory?
 - a. Best Case
 - b. Worst Case
 - c. Average Case
 - d. Null Case**
6. The Complexity of the average case of an algorithm is
 - a. Much more complicated to analyse than that of worst case**
 - b. Much more simpler to analyse than that of worst case
 - c. Sometime more complicated and some other times simpler than that of worst case
 - d. None of above
7. Which of the following data structure is linear data structure?
 - a. Tree
 - b. Graph
 - c. Array**
 - d. None of above
8. Which of the following data structure is linear type?
 - a. Strings
 - b. Lists
 - c. Queues
 - d. All of above**

9. Match the following

- | | |
|---------------------|--|
| a. Completeness | i. How long does it take to find a solution |
| b. Time complexity | ii. How much memory need to perform the search |
| c. Space Complexity | iii. Is the strategy guaranteed to find the solution |
- where there in one

- a. a-iii, b-ii, c-i
- b. a-i, b-ii, c-iii
- c. a-iii, b-i, c-ii**
- d. a-i, b-iii, c-ii

10. The number of comparison done by sequential search is _____

- a. $(N/2)+1$
- b. $N+1/2$**
- c. $N-1/2$
- d. $N+2/2$