Unit 1

**Syllabus:**

UNIT -1 (15 hrs) **Algorithm Analysis**: Introduction to algorithms, analyzing and designing algorithms, Growth functions, asymptotic notations, Recursive algorithm complexity, **solving recurrences**: Substitution method, recursion tree method, master method. **Searching**: Binary search, **Hashing**: Hashing, Hash tables, Hash functions, collision resolution techniques. **Sorting**: Quick sort, Counting sort, Radix sort, Merge sort, Heap sort, Insertion sort and selection sort.

## Introduction to algorithm:

1. **What is means algorithm:**
2. An **algorithm** is step by step description of any program in general language
3. An **algorithm** is a sequence of a clear instructions used to solving a problem such a way that it can implemented as a program for computer

1. An **algorithm** is like a recipe for solving a specific problem. It’s a sequence of well-defined steps or instructions that guide a computer (or a human) through a process to achieve a desired outcome.
2. **Characteristics of algorithm:**

**Characteristics of a Good Algorithm:**

1. **Correctness**:
   * An algorithm must produce the correct output for all valid inputs.
   * It should solve the problem accurately without errors.
   * Verifying correctness often involves mathematical proofs or extensive testing.
2. **Efficiency**:
   * A good algorithm should be efficient in terms of time and memory usage.
   * Efficiency matters because we want solutions that run quickly and don’t waste resources.
   * Common measures of efficiency include Big O notation and space complexity.
3. **Clarity and Simplicity**:
   * An algorithm should be easy to understand.
   * Clear, straightforward steps make it easier to implement and maintain.
   * Avoid unnecessary complexity.
4. **Language Independence**:

* An algorithm is expressed in a high-level, abstract manner that doesn’t depend on any specific programming language.
* Whether you write an algorithm in Python, Java, C++, or any other language, the underlying logic remains the same.

1. **Generality**:
   * A general-purpose algorithm can be applied to a wide range of inputs.
   * Specific algorithms (e.g., sorting, searching) should work for various data types and sizes.
2. **Deterministic**:
   * Algorithms are deterministic—they produce the same output for the same input every time.
   * There’s no randomness involved.
3. **Finiteness**:
   * An algorithm must terminate after a finite number of steps.
   * Infinite loops are not acceptable.

Remember, algorithms are everywhere! When you use your phone, search the web, or even cook a new recipe, you’re interacting with algorithms. They’re the backbone of modern computing and problem-solving.

1. Advantages and Disadvantages of algorithm:

5M

**Advantages of Algorithms:**

1. **Effective communication:**

Since it is written in natural language like English it become easy to understand the step-by-step definition of a solution to any particular problem.

1. **Easy debugging:**

A well-designed algorithm facilities easy debugging to detect the logical errors that occurred inside the program.

1. **Easy and efficient coding:**

An algorithm is nothing but a blue print of the program that helps to develop a program.

1. **Independent of programming language:**

Since it is language independent, it can be easily coded by incorporating any high level language.

**Disadvantages of Algorithms:**

* 1. **Inefficiency**

Some algorithms are inefficient in terms of time and space complexity. Leading to lower processing speed and high resources usage.

* 1. **Not universal applicable**

Not all algorithms are suitable for every problem. Using the wrong algorithm can result in suboptimal solutions.

* 1. **Optimization challenges**

Optimizing algorithm for specific use cases or dataset can be complex and time-consuming process requiring in depth knowledge and expertise

* 1. **Scalability issues:**

Some algorithm may not scale well when applied to large dataset causing performance issue as the size of database increases.

**2M**

**Advantages of Algorithms:**

* It is easy to understand.
* An algorithm is a step-wise representation of a solution to a given problem.
* In an Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

**Disadvantages of Algorithms:**

* Writing an algorithm takes a long time so it is time-consuming.
* Understanding complex logic through algorithms can be very difficult.
* Branching and Looping statements are difficult to show in Algorithms.

## Algorithm analysis

## **Algorithm analysis** is a crucial aspect of computer science that **involves evaluating and understanding the efficiency of algorithms.**

## **Definition:**

## Algorithm analysis refers to the study of algorithms behaviour and performance.

## It aims to determine how much time and memory resources an algorithm requires to solve a specific computational problem.

* Need of algorithm analysis:
  1. **Efficiency:**

Algorithm analysis helps us to understand how fast and how much space an algorithm will use, allowing us to select the most efficient one for a specific task.

* 1. **Performance comparison:**

By analysing algorithm, we can compare them to see which one works best for the particular problem. Helping us make smarter choice.

* 1. **Resource management:**

Algorithm analysis assist in managing resources effectively, ensuring we use computational resources like memory and processing power optimally.

* 1. **Scalability:**

It helps to predict how algorithm will perform as the input size increases. Aiding in designing solutions that can handles varying data sizes efficiently.

* 1. **Optimization:**

Algorithm analysis helps identify areas for improvements. Leading to better performance and efficiency of algorithm and system.

## Phase of Algorithm

## The algorithm can be examined in 2 phases. The two analyses of an algorithm are as follows:

## **Priori Analysis:**

## Priori analysis is the theoretical analysis of an algorithm performed prior to its implementation.

## Before running or executing the algorithm, other parameters might be considered, such as the speed of the processor, which has no impact on the execution phase.

* + Time complexity and space complexity are determined theoretically.
  + Often uses asymptotic notations (like Big O) to represent time complexity.

## **Posterior Analysis**:

## It is also known as the practical analysis of an algorithm.

## The algorithm is implemented in any computer language to obtain practical analysis.

## This analysis is used to determine how much running time and space the technique consumes.

* + Time and space complexity are observed during actual runs.
  + Results may vary from system to system.

| **Aspect** | **A Priori Analysis** | **A Posteriori Analysis** |
| --- | --- | --- |
| **Definition** | Theoretical analysis based on models | Empirical analysis based on actual runs |
| **Focus** | Idealized behaviour | Real-world performance |
| **Resource Independence** | Independent of specific hardware, compiler, or system conditions | Dependent on specific system conditions |
| **Accuracy** | Approximate results | Exact results |
| **Notations Used** | Asymptotic notations (e.g., Big O) | Actual execution time |
| **Credit for Speed** | Goes to the programmer | May go to compiler or hardware |
| **Timing** | Done before execution | Done after execution |
| **Cost** | Cheaper | Costlier (requires software and hardware) |
| **Maintenance Phase** | Not required | Required for tuning |

## Quick sort

* The algorithm was developed by a British computer scientist Tony Hoare in 1959.
* The name "Quick Sort" comes from the fact that, quick sort is capable of sorting a list of data elements significantly faster (twice or thrice faster) than any of the common sorting algorithms
* It is one of the most efficient sorting algorithms and is based on the splitting of an array (partition) into smaller ones and swapping (exchange) based on the comparison with 'pivot' element selected.
* Due to this, quick sort is also called as "Partition Exchange" sort.

**Algorithm :**

* **Step 1** − Choose the first index value has pivot
* **Step 2** − Take two variables to point left and right of the list excluding pivot
* **Step 3** − left points to the low index
* **Step 4** − right points to the high
* **Step 5** − while value at left is less than pivot move right
* **Step 6** − while value at right is greater than pivot move left
* **Step 7** − if both step 5 and step 6 does not match swap left and right
* **Step 8** − if left ≥ right, the point where they met is new pivot

**Example**:

*Consider: arr [] = {10, 80, 30, 90, 40, 50, 70}.*



## s

## Binary search:

Example:

void main() {

  List<int> a = [10, 20, 30, 40, 50];

  int l = 0; //lower index

  int u = 4; //upper index

  int mid = 0; //for middle value

  int f = 0; //for found or not

  int s = 50; //search this

  // loop till lower index<=upper index

  while (l <= u) {

    mid = ((l + u) / 2).floor();

    if (a[mid] == s) {

      f = 1;

      break;

    }

    if (a[mid] < s) {

      l = mid + 1;

    } else {

      u = mid - 1;

    }

  }

  if (f == 1) {

    print("$s is found at position ${mid + 1}");

  } else {

    print("$s in not found in $a");

  }

}

/\*

trace pgm

List<int> a = [10, 20, 30, 40, 50];

s=10

while:

?itration 1

l=0 u=4

mid=l+u/2= 0+4/2= 2

if(a[2]==10){

30==10

!false

}

if(a[2]<10){

!false

}

else{

u=2-1=1

}

?itratuin 2

l=0 u=1 s=10

mid=l+u/2=0+1/2=0

if(a[0]==10){

true

f=1

?exit;

}

\*/

/\*

trace pgm

l=0 u=4 s=50

?itration 1

while:

(l<=u)0<=4 //?true

{

mid=l+u/2 =0+4/2 =2

if(a[mid]==s) a[2]==50 30==50 //!false

{

!not execute

}

if(a[mid]<s) a[2]<50 =30<50 //?true

{

l=mid+1 =2+1 =3

}

else

{

!not execute

}

}

?itration 2

l=3 u=4 s=50

while:

(l<=u) 3<=4

{

mid=l+u/2 =3+4/2 =3

if(a[mid]==s) a[3]==s =40==50 //!false

{

!not execute

}

if(a[mid]<s) a[3]<50 =40<50 //?true

{

l=mid+1 l=3+1 =4

}

else

{

!not execute

}

}

?iteration 3:

l=4 u=4 s=50

while:

(l<=u) 4<=4 //?true

{

mid=l+u/2 4+4/2 =4

if(a[mid]==s) a[4]==50 50==50 //?true

{

f=1

break //?exit

}

!not execute

}

print==>50 found at position 4

\*/