RLMCA 207

DESIGN AND ANALYSIS OF ALGORITHMS

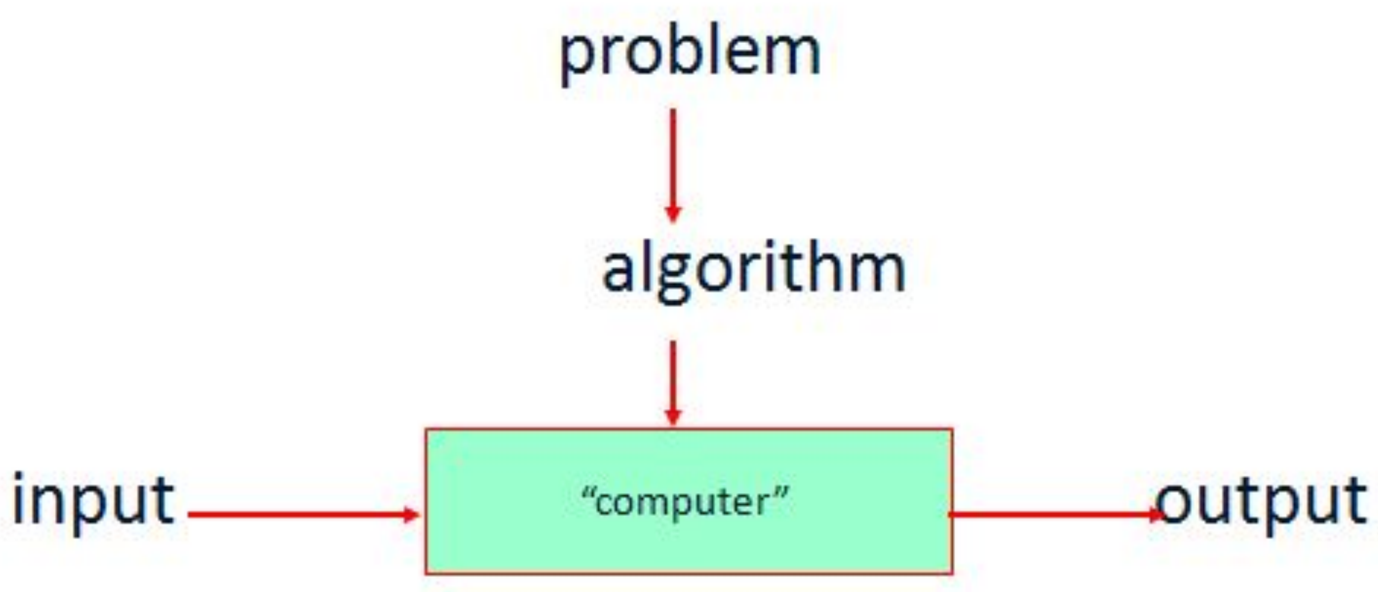
MODULE 1

dule	Contents
I	Introduction to Algorithm Analysis : Algorithm and its properties - Apriory and Aposterior analysis of algorithms - Time and Space Complexity- Elementary Operation and Complexity Estimation of Simple Algorithms - Asymptotic notations and their properties - Common Complexity functions - Recurrence Relations - Solution of Recurrence Relations - Iteration Method - Recurrence Tree Method - Master's Theorem (Proof not required)

What is an algorithm?

- An ***algorithm*** is a list of steps (sequence of unambiguous instructions) for solving a problem that transforms the input into the output.
- ie. for obtaining a required output for any legitimate input in a finite amount of time.

What is an algorithm?



Characteristics of Algorithms

- **Unambiguous** − Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** − An algorithm should have 0 or more well-defined inputs.
- **Output** − An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** − Algorithms must terminate after a finite number of steps.
- **Feasibility** − Should be feasible with the available resources.
- **Independent** − An algorithm should have step-by-step directions, which should be independent of any programming code.

Characteristics of Algorithms

- The nonambiguity requirement for each step of an algorithm cannot be compromised.
- The range of inputs for which an algorithm works has to be specified carefully.
- The same algorithm can be represented in several different ways.
- There may exist several algorithms for solving the same problem.
- Algorithms for the same problem can be based on very different ideas and can solve the problem with dramatically different speeds.

Algorithm Design and Analysis Process

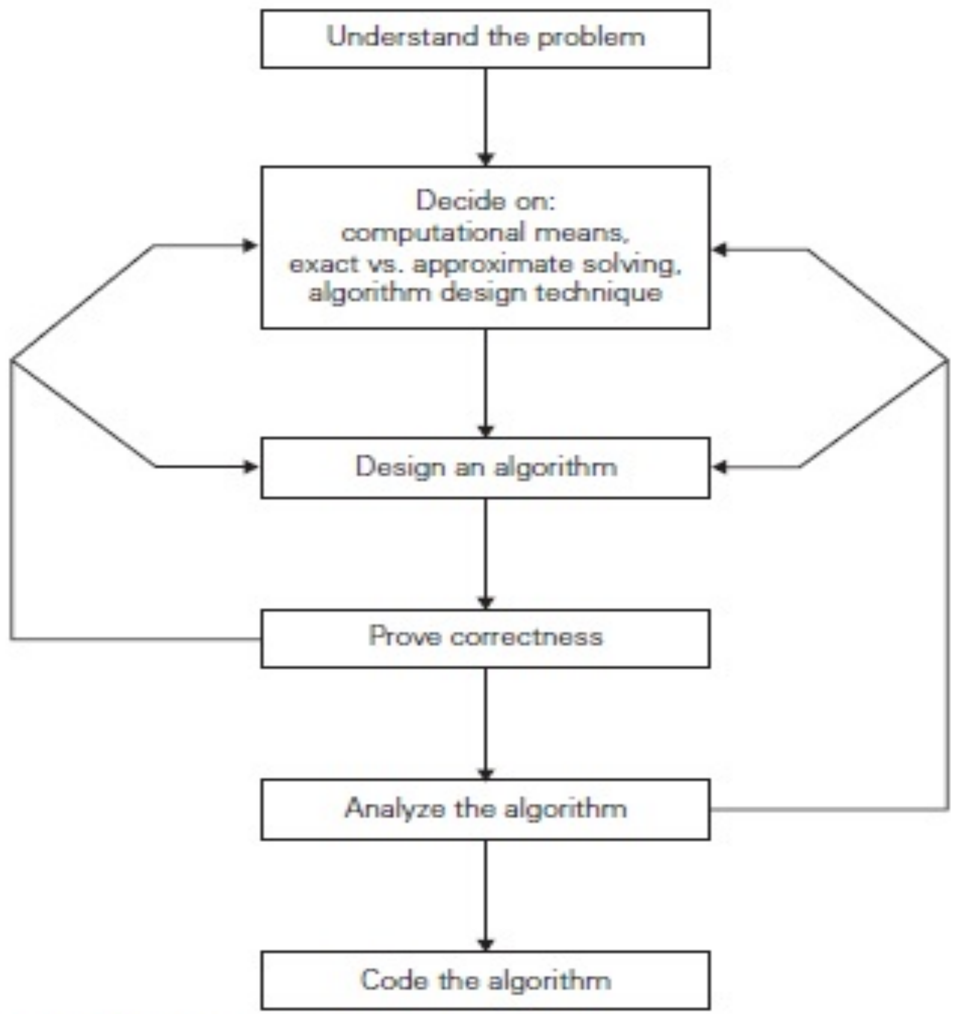


FIGURE 1.2 Algorithm design and analysis process.

Algorithm Analysis

- Efficiency of an algorithm can be analyzed at two different stages

- before implementation- ***Apriori*** **Analysis**
- after implementation- ***Aposterior*** **Analysis**

Apriori Analysis

- This is a theoretical analysis of an algorithm
- Efficiency of an algorithm is measured by assuming that all other factors are constant and have no effect on the implementation.
- for example, processor speed is constant

Aposterior Analysis

- This is an empirical analysis of an algorithm.
- The selected algorithm is implemented using programming language.
- This is then executed on target computer machine.
- In this analysis, actual statistics like running time and space required, are collected.

Comparison

Apriori Analysis	Aposterior analysis
1.Algorithm	1.program
2.Independent of language	2.Language dependent
3.Hardware Independent	3.Hardware dependent
4.Time and space function	4.Execution time and bytes

Analyze an algorithm

```
Algorithm Swap(a,b)
{
temp=a;
a=b;
b=a;
 }
```

```
Algorithm Swap(a,b)
  begin
   temp=a;     -1
    a=b;          -1
   b=a;           -1
    end
```

```
Algorithm Swap(a,b)
    begin
     temp<-a
     a <- b
     b <- a
      end
```

Criteria for Analyze Algorithms

1. Time
2. Space

Suppose **X** is an algorithm and **n** is the size of input data,the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

- **Time Factor** – Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
- **Space Factor** – Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data

$f(n) = S(n)+T(n)$

Example

- Algorithm Sum(A ,n)

```
{
 s=0;
 for(i=0;i<n;i++)
   {
s=s+A[i];
    }
  return s;
}
```

Time complexity analysis

- Algorithm Sum(A ,n)

```
{
 s=0;                          -1
 for(i=0;i<n;i++)              - n+1
   {
   s=s+A[i];                   -n
   }
 return s;                     -1
}
```

$T(n)=2n+3$

 O(n)

Space complexity

- A - n
- N -1
- S -1
- I -1

$S(n) = n+3 \quad O(n)$

```
for(i =0;i<n ; i++)
{
stmt;

 }
```

```
for(i = n;  i>n ; i--)
    {
    stmt;


     }
```

```
for(i =0;i<n ; i+2)
  {
   stmt;

    }
```

```
for(i =0;i<n ; i++)
    {
      for(j=0;j<n; j++)
     {
    stmt;
     }

     }
```

```
for(i =0;i<n ; i++)            -n+1
  {
    for(j=0;j<n; j++)          -(n+1)*(n+1)
   {
  stmt;                        - n*n
   }

   }                  o(n^2)
```

```
for(i =0;i<n ; i++)
   {
     for(j=0;j<i; j++)
    {
   stmt;
    }


    }
```

```
for(i =0;i<n ; i++)
   {
     for(j=0;j<i; j++)
    {
   stmt;
    }
   }
```

F(n)=n(n+1)/2   o(n^2)