UNIT I
Introduction to Algorithms

Algorithm Definition

An Algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms should 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 very basic so that it can be carried out, in principle, by a person using only pencil & paper.

study of Algorithms

- The way to devise or design an algorithm It includes the study of various design techniques and helps in writing algorithms using the existing design techniques like divide and conquer, backtracking etc..
- The way to validate an algorithm After the algorithm is written it is necessary to check the correctness of the algorithm i.e for each input correct output is produced, known as algorithm validation. The second phase is writing a program known as program proving or program verification.
- The way to analyse an algorithm It is known as analysis of algorithms or performance analysis, refers to the task of calculating time and space complexity of the algorithm.

study of Algorithms contd..

- The way to test a program It consists of two phases.

  1. Debugging is detection and correction of errors.

  2. Profiling or performance measurement is the actual amount of time required by the program to compute the result.

ALGORITHM SPECIFICATION

- Pseudo-Code for writing Algorithms:

  1. Comments begin with // and continue until the end of line.
  2. Blocks are indicated with matching braces { and }.
  3. An identifier begins with a letter. The data types of variables are not explicitly declared.
  4. Compound data types can be formed with records. Here is an example,

```
Node. Record
    { data type – 1 data-1;
        -------
        -------
        -------
       data type – n data – n;
       node * link;
    }
```

Here link is a pointer to the record type node. Individual data items of a record can be accessed with and period.

ALGORITHM SPECIFICATION contd..

5. Assignment of values to variables is done using the assignment statement.

    variable := expression ;

6. There are two Boolean values TRUE and FALSE. Logical Operators AND, OR, NOT Relational Operators <=,>,>=, =, !=

7. The following looping statements are employed.

    For, while and repeat-until

    While Loop:

```
While < condition >
 do {
        statement 1;
        statement 2;
         -----
         -----
        statement n;
      }
```

ALGORITHM SPECIFICATION contd..

- For Loop:

```
for variable: = value-1 to value-2 step step
do {
    statement 1;
    statement 2;
    -----
    -----
    statement n;
}
```

One step is a key word, other Step is used for increment or decrement

- repeat-until:

```
repeat{
  {
    statement 1;
    statement 2;
    -----
    -----
    statement n;
  }until < condition>
```

ALGORITHM SPECIFICATION contd..

8. A conditionalstatement has the following forms.

```
 (1) If then
(2) If then Else
 Case statement:
 Case {
           condition 1 : statements;
                         break;
            condition 2 : statements;
                         break;
             condition 3 : statements;
                         break;
              -------------------------------
              -------------------------------
              -------------------------------
              condition n : statements;


             }
```

ALGORITHM SPECIFICATION contd..

- 9. Input and output are done using the instructions read & write.
- 10. There is only one type of procedure: Algorithm, the heading takes the form,

```
Algorithm Name ( parameters list)
{
    -----------
    -----------
    -----------
    -----------
}
```

Performance Analysis:

- Performance of an algorithm is a process of making evaluative judgment about algorithms.

•Performance of an algorithm means predicting the resources which are required to an algorithm to perform its task.

•That means when we have multiple algorithms to solve a problem, we need to select a suitable algorithm to solve that problem.

•We compare all algorithms with each other which are solving same problem, to select best algorithm.

•To compare algorithms, we use a set of parameters or set of elements like memory required by that algorithm, execution speed of that algorithm, easy to understand, easy to implement, etc.

Performance Analysis contd..

Generally, the performance of an algorithm depends on the following elements...

- ✓ Whether that algorithm is providing the exact solution for the problem?
- ✓ Whether it is easy to understand?
- ✓ Whether it is easy to implement?
- ✓ How much space (memory) it requires to solve the problem?
- ✓ How much time it takes to solve the problem? Etc.,

When we want to analyze an algorithm, we consider only the space and time required by that particular algorithm and we ignore all remaining elements.

Performance Analysis contd..

- Performance analysis of an algorithm is the process of calculating space required by that algorithm and time required by that algorithm.
- Performance analysis of an algorithm is performed by using the following measures...

✓ Space required to complete the task of that algorithm (Space Complexity). It includes program space and data space

✓ Time required to complete the task of that algorithm (Time Complexity)

Performance Analysis contd..

- Performance evaluation can be divided into two major phases.

1. Performance Analysis (machine independent): It is a priori estimate, that occurs in two ways

✓ Space Complexity: The space complexity of an algorithm is the amount of memory it needs to run for completion.

✓ Time Complexity: The time complexity of an algorithm is the amount of computer time it needs to run to completion.

2 . Performance Measurement (machine dependent): It is a posteriori testing .

Space Complexity

The space complexity of an algorithm is the amount of memory it needs to run to completion.

The Space Complexity of any algorithm P is given by S(P)=C+SP (I),C is constant.

- Fixed Space Requirements (C)

  Independent of the characteristics of the inputs and outputs

✓ It includes instruction space

✓ space for simple variables, fixed-size structured variable, constants

- Variable Space Requirements (SP (I)) depend on the instance characteristics

✓ number, size, values of inputs and outputs associated with I

✓ recursive stack space, formal parameters, local variables, return address Algorithm.

Time Complexity

- The time complexity of an algorithm is the amount of computer time it needs to run to completion of algorithm and providing output .
- The time T(P) taken by a program P is the sum of the compile time and the run (or execution)time. The compile time does not depend on the instance characteristics.

  T(P)=CT+TP (I)

  It is combination of

  -Compile time (CT) independent of instance characteristics

  -Run (execution) time TP dependent of instance characteristics
- Time complexity is calculated in terms of program step as it is difficult to know the complexities of individual operations.

Computing Time Complexity

- Tabular method for computing Time Complexity :

✓ Complexity is determined by using a table which includes steps per execution(s/e) i.e amount by which count changes as a result of execution of the statement.

✓ Frequency – number of times a statement is executed.

Time complexity Analysis

- The worst-case complexity of the algorithm is the function defined by the maximum number of steps taken on any instance of size n. It represents the curve passing through the highest point of each column.
- The best-case complexity of the algorithm is the function defined by the minimum number of steps taken on any instance of size n. It represents the curve passing through the lowest point of each column.
- The average-case complexity of the algorithm is the function defined by the average number of steps taken on any instance of size n.

Ex1:Frequency table for a given algorithm for calculating time complexity

	Statement	Steps /execution	Frequency	Total steps
	Algorithm Sum ()	0	----	0
	{	0	----	0
	read(a);	1	1	1
	read(b);	1	1	1
	c := a+b;	1	1	1
	return c;	1	1	1
	}	0	---	0
	Total steps			4 → O (4)

Ex2:Frequency table for a given algorithm for calculating time complexity

	Statement	Steps /execution	Frequency	Total steps
	Algorithm ArSum (a,n)	0	---	0
	{	0	----	0
	s:=0.0;	1	1	1
	for i :=1to n do	1	n +1	n + 1
	s :=s+a[i];	1	n	n
	returns;	1	1	1
	}	0	---	0
	Total steps			2n+3 $\rightarrow$ O (n)

Ex3:Frequency table for a given algorithm for calculating time complexity

Statement	Steps/execution	Frequency	Total steps
Algorithm Sort ()	0	----	0
{	0	----	0
For I := 0 to n – 1	1	n+1	n+1
{	0	0	0
For J := 0 to n – 1	1	(n+1)*n	$n^2 + n$
{	0	0	0
If (A[J] > A[J + 1])	1	n*(n)	n^2
{	0	0	0
Temp = A[J];	1	n*n	n^2
A[J] = A[J + 1];	1	n*n	n^2
A[J + 1] = Temp;	1	n*n	n^2
}	0		
}	0		
}	0		
Total steps			$5n^2 + 2n + 1 \rightarrow O(n^2)$

Asymptotic Notations for performance analysis

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- ✓ O Notation
- ✓ Ω Notation
- ✓ θ Notation

Big ‘O’ Notation

- In the time complexity , we define the time as a function of problem size and try to estimate the growth of execution time with the growth of the problem size.
- Big ‘O’ notation is also called Landau’s symbol that describe the behavior of the functions.
- Landau’s symbol comes from the name of German mathematician Edmund Landau who invented this notation.
- Letter ‘O’ is used to rate the growth of a function is also known as order.

Big ‘O’ Notation contd…

Big ‘O’ Notation describes

- ✓ Efficiency of an algorithm.
- ✓ Time factor of an algorithm.
- ✓ Space complexity of an algorithm.

The ascending order of complexities are

From fastest to slowest

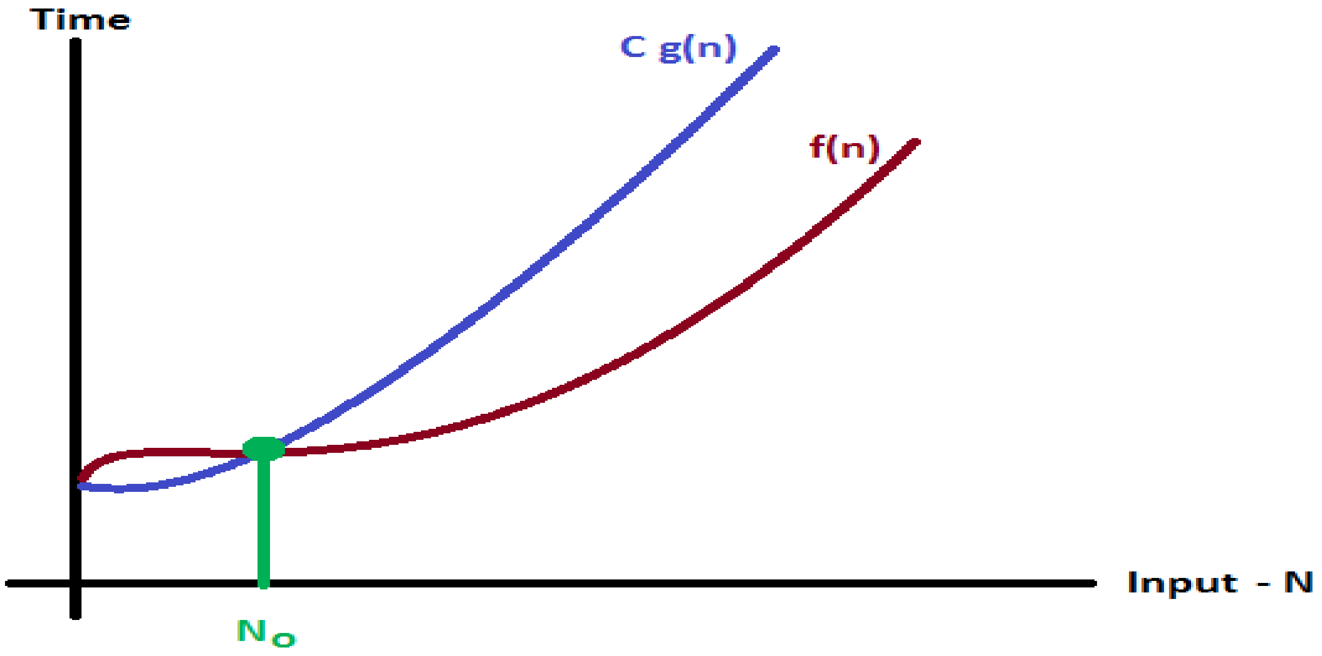
✓$O(1)$	constant (or $O(k)$ for constant k)
✓$O(\log n)$	logarithmic
✓$O(n)$	linear
✓$O(n \log n)$	"n log n”
✓$O(n^2)$	quadratic
✓$O(n^3)$	cubic
✓$O(n^k)$	polynomial (where is k is constant)
✓$O(k^n)$	exponential (where constant $k > 1$)

Big ‘O’ Notation contd…

It is a method of representing the upper bound of algorithm running time. Let f(n) and g(n) are two non-negative functions used for representing big O notations.

- **If f(n)=O(g(n)) if and only if f(n)<=c*g(n) for all n, $n>=n_0$**

  where c, n_0 are positive constants. g(n) represents the upper bound of f(n). c>0 and $n_0>=1$.



Big ‘O’ Notation contd…

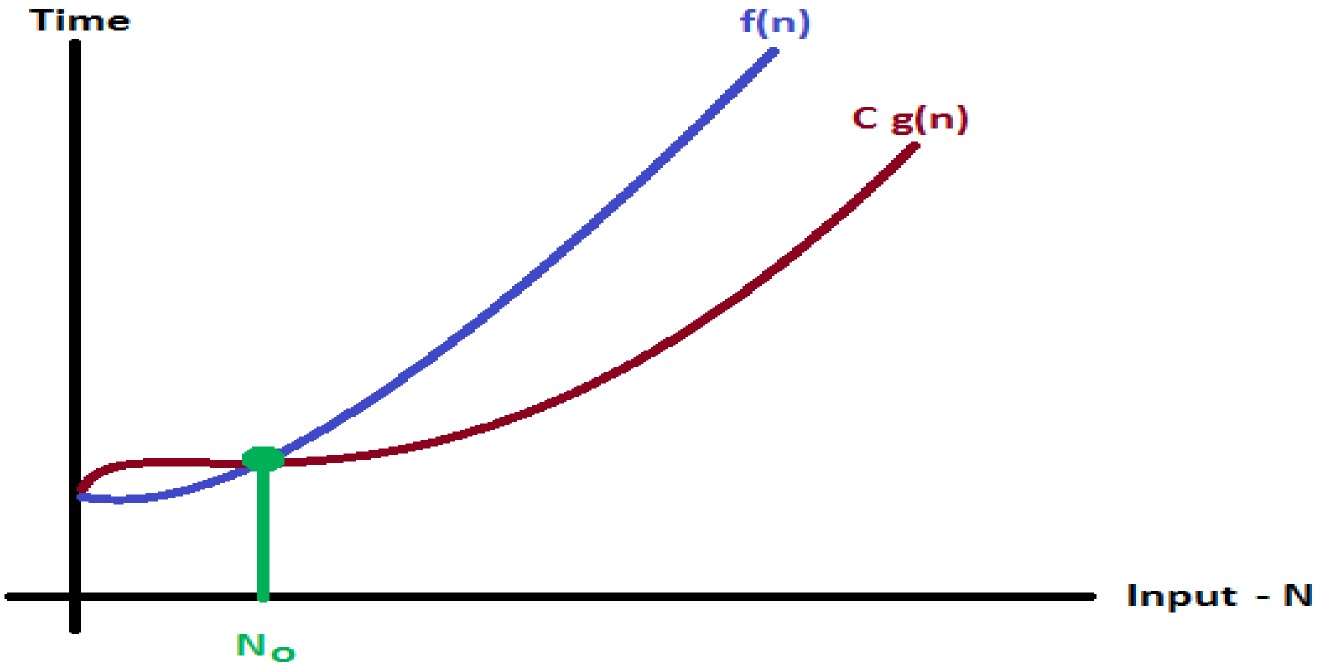
- Example for Big ‘O’ Notation :
- $3n+2 = 0(n)$ as $3n+2 <= 4n$ for all $n >=2$.
- $3n + 3 = O(n)$ as $3n + 3 <= 4n$ for all $n >= 3$.
- $10n^2+4n +2 = 0(n^2)$ as $10n^2+4n+2 <= 11n^2$ for all $n >5$
- $6*2^n+n^2 = 0(2^n)$ as $6*2^n+n^2<=7*2^n$ for $n>=4$
- With previous examples, the statement $f(n) = 0(g(n))$ states only that $g(n)$ is an upper bound on the value of $f(n)$ for all n, $n >= n_0$.

Omega notation: Ω

It is a method of representing the lower bound of algorithm running time. Let f(n) and g(n) are two non-negative functions used for representing omega notations.

If f(n)=Ω(g(n)) if and only if f(n)>=c*g(n) for all n, n>=n_0

where c, n_0 are positive constants. g(n) represents the lower bound of f(n). c>0 and n_0>=1.



Omega notation: Ω contd..

- Examples:
  - $3n+2 = \Omega(n)$ as $3n+2 >= 3n$ for $n >=1$.
  - $3n + 3 = \Omega(n)$ as $3n + 3 >= 3n$ for all $n >= 1$.
  - $10n^2+4n +2 = \Omega(n^2)$ as $10n^2+4n+2 >=n^2$ for $n >=1$
  - $6*2^n+n^2 = \Omega(2^n)$ as $6*2^n+n^2 >= 2^n$ for $n>=1$
- There are several functions $g(n)$ for which

  $f(n) = \Omega (g(n))$. The function $g(n)$ is only a lower bound $f(n)$.

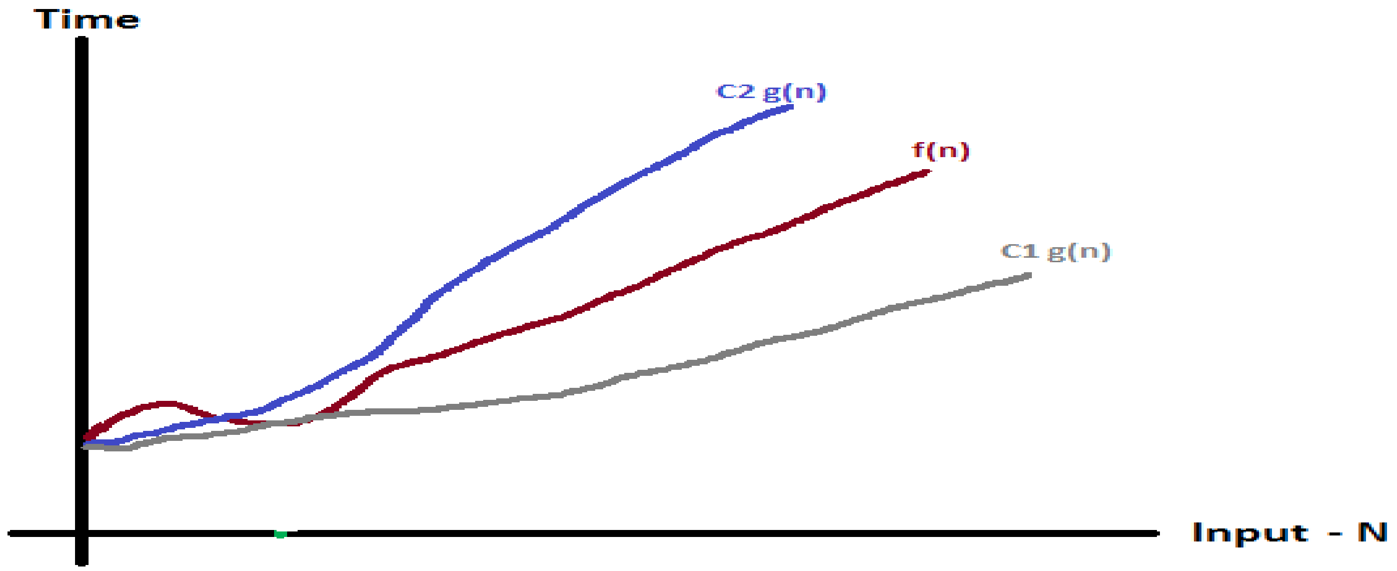
THETA 'Θ' NOTATION

It is a method of representing both lower and upper bound of algorithm running time.
Let f(n) and g(n) are two non-negative functions used for representing theta notations.

If f(n)=Θ(g(n)) if and only if c1*g(n)<=f(n)<=c2*g(n) for all n, n>=n_0

where c1,c2, n_0 are positive constants. g(n) represents the average boundary level of f(n).

c1>0,c2>0 and n_0>=1.



THETA 'Θ' NOTATION contd..

➢ Examples:

- $3n+2 = \Theta(n)$ as $3n+2 >= 3n$ for all $n >= 2$

  and $3n+2 <= 4n$ for all $n > = 2$, so $c_1 = 3$, $C_2 = 4$, and

  $n_o = 2$. So, $3n+3 = \Theta(n)$.
- $10n^2+4n+2 = \Theta(n^2)$
- $6*2^n+n^2 = \Theta(2^n)$
- $10*\log n + 4 = \Theta(\log n)$
- The theta notation is more precise than both the big oh and omega notations. The function

  $f(n) = \Theta(g(n))$ iff $g(n)$ is both an upper bound and lower bound of $f(n)$.

Little 'o' notation(little omega notation)

- The function f(n) = o(g(n)) if and only if

  $\lim_{n \to \infty} f(n)/g(n)=0$ for all n, n>=0
- Also uses the notation as f(n) = ω(f(n))
- **Example:-**

  3n+2 = o(1)

  f(n) = 3n+2 g(n) = 1

  o(1)

  $\lim_{n \to \infty} ((3n+2)/1)$

  $= \lim_{n \to \infty} 3n + \lim_{n \to \infty} 2$

  ∞+2 = 0 (which is not possible), proceed with other solution

Little 'o' notation(little omega notation) contd..

- $3n+2 = o(n)$

  $$\lim_{n \to \infty} ((3n+2)/n)$$

  $$= \lim_{n \to \infty} (3n/n) \quad + \quad \lim_{n \to \infty} (2/n)$$

  $$= \lim_{n \to \infty} 3 + 0$$

  3 = 0 (which is not possible), proceed with other solution

Little 'o' notation(little omega notation) contd..

- Assume $3n+2 = o(n^2)$

$$\lim_{n\to\infty} ((3n+2)/n^2)$$

$$= \lim_{n\to\infty} (3n/n^2) + \lim_{n\to\infty} (1/n^2)$$

$$= 0 + 0 = 0$$

- Therefore $3n+2 = o(n^2)$ is the solution.

Probabilistic Analysis

- In analysis of algorithms, probabilistic analysis of algorithms is an approach to estimate the computational complexity of an algorithm or a computational problem.
- It starts from an assumption about a probabilistic distribution of the set of all possible inputs.
- This assumption is then used to design an efficient algorithm or to derive the complexity of a known algorithm.
- This approach is not the same as that of probabilistic algorithms, but the two may be combined.

Probabilistic Analysis contd..

- For non-probabilistic, more specifically, for deterministic algorithms, the most common types of complexity estimates are

  1. The average case complexity (expected time complexity), in which given an input distribution, the expected time of an algorithm is evaluated.

  2. The almost always complexity estimates, in which given an input distribution, it is evaluated that the algorithm admits given complexity estimate that almost surely holds.

- In probabilistic analysis of probabilistic (randomized) algorithms, the distributions or averaging for all possible choices in randomized steps are also taken into an account, in addition to the input distributions.

Probabilistic Analysis example

•**Hiring Problem:-**

Hire Assistant (n):

```
1.best = 0     //Candidate 0 is at least qualified dummy candidate.
2. for i = 1 to n
3.Interview candidate i
4.If candidate i is better than candidate best
5.best = i
6.hire  candidate i
```

- **Worst Case Analysis:**

  In the worst case we actually hire every candidate we interviewed. This situation occurs if the candidates comes in strictly increasing order of quality in this case we hire ‘n’ times for a total hiring cost of O(n*c h.).

Probabilistic Analysis example contd..

- Rank Based Hiring:

  We can represent a particular input by listing in order, the rank of the candidates

  (rank(1),rank(2),. . . . .,rank(n))

  i.e., A = {1,2,3,. . . . . ,n}

- In rank based hiring we need 10 people out of 100 then we will select first 10 numbers based on rank where as in hire assistant if we need 10 people out of 100 we should interview each and every candidate.

Amortized Analysis

- The actual complexity of an operation is determined by the step count for that operation, and the actual complexity of a sequence of operations is determined by the step count for that sequence.
- The actual complexity of a sequence of operations may be determined by adding together the step counts for the individual operations in the sequence.
- Typically, determining the step count for each operation in the sequence is quite difficult, and instead, we obtain an upper bound on the step count for the sequence by adding together the worst-case step count for each operation.

Amortized Analysis contd..

1) Amortized cost of a sequence of operations can be seen as expenses of a salaried person. The average monthly expense of the person is less than or equal to the salary, but the person can spend more money in a particular month by buying a car or something. In other months, he or she saves money for the expensive month.
2) The above Amortized Analysis done for Dynamic Array example is called Aggregate Method. There are two more powerful ways to do Amortized analysis called Accounting Method and Potential Method. We will be discussing the other two methods further.
3) The amortized analysis doesn't involve probability. There is also another different notion of average case running time where algorithms use randomization to make them faster and expected running time is faster than the worst case running time. These algorithms are analyzed using Randomized Analysis. Examples of these algorithms are Randomized Quick Sort, Quick Select and Hashing.

Amortized Analysis contd..

- Example:

✓ Here the only requirement is that the sum of the amortized complexities of all the operations in any sequence of operations must be greater than or equal to their sum of the actual complexities i.e.,

$$\left(\sum_{(1 <= i <= n)} \text{amortized}(i)\right) >= \left(\sum_{(1 <= i <= n)} \text{actual}(i)\right)$$

✓ When amortized(i) and actual(i) respectively denote the amortized and actual complexities of ith operation in the sequence of operations.

Amortized Analysis contd..

Month	1	2	3	4	5	6	7	8	9
Actual	50	50	100	50	50	100	50	50	100
Amortized	75	75	75	75	75	75	75	75	75
P()	25	50	25	50	75	50	75	100	75

Month	10	11	12	13	14	15	16
Actual	50	50	200	50	50	100	50
Amortized	75	75	75	75	75	75	75
P()	100	125	0	25	50	25	50

Amortized Analysis contd..

- Relative to the actual and amortized costs of each operation in a sequence of n operations, We define the potential function as

  P(i) = amortized(i) – actual(i) + P(i-1)

- Generalized form:

∑(1 <= i <= n) P(i) = ∑(1<=i<=n) (amortized(i) – actual(i) + P(i-1))

3 types of Amortized analysis:

- Aggregate Method:-

  In this method we uniformly assign the amortized cost to the each operation.
- Accounting Method:-

  In this method we assign amortized costs to the operators by guessing what assignments will work.
- Potential Method:-

  Here we start with a potential function which is obtained by using good guess work.

End of Unit 1

Any querries?