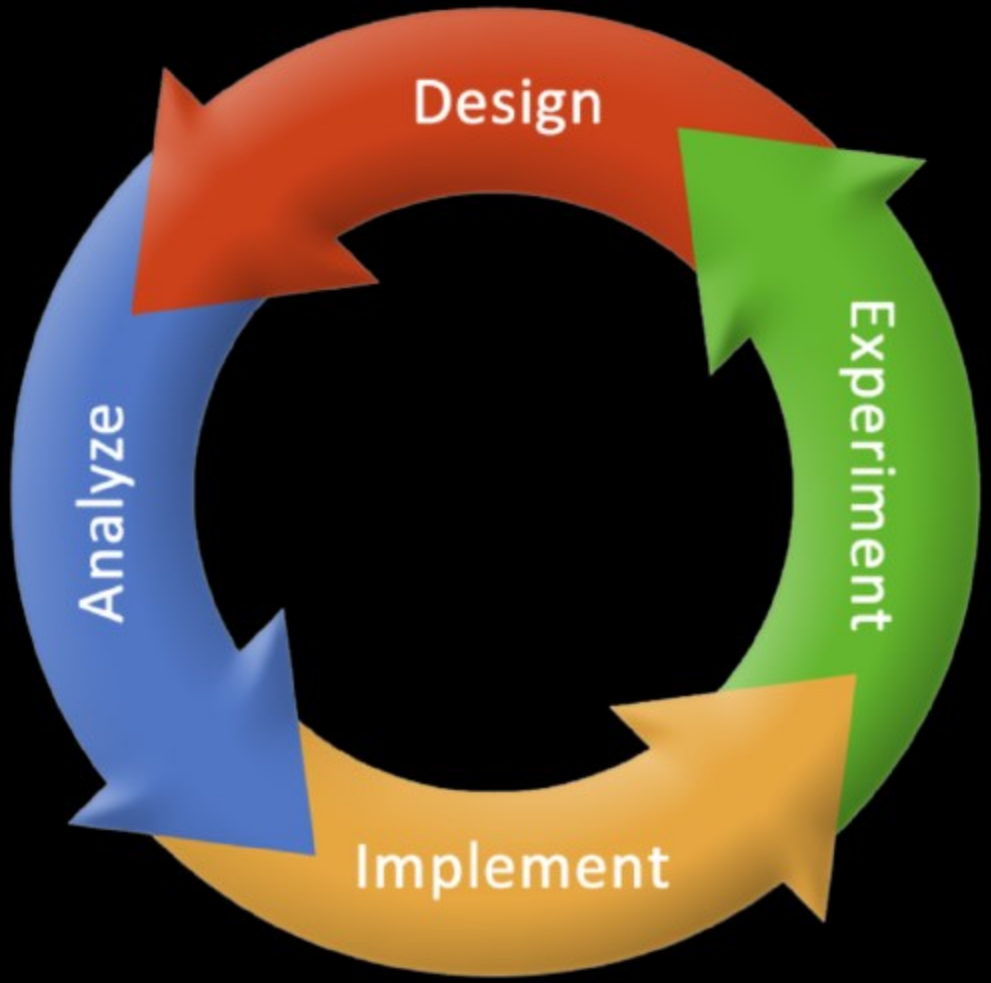


Algorithm

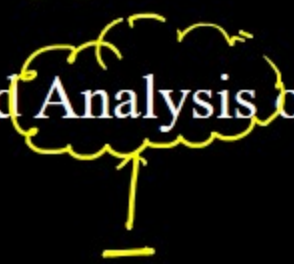
Abhishek Sharma

- 2009 —

+ • GATE
  • placement

Algorithm

- Design and Analysis of Algorithm



Introduction

Analysis

Greedy

Divide & Conquer + Mathematical Aspect tool Asymptotic Notation

Dynamic

Graph

Syllabus

Section 5: Algorithms

- Searching, sorting, hashing. Asymptotic worst case time and space complexity. Algorithm design techniques: greedy, dynamic programming and divide-and-conquer. Graph traversals, minimum spanning trees, shortest paths

Data structure

. Best case
. Average case (some)

. Backtracking
. Branch & Bound

Application - DFS/BFS

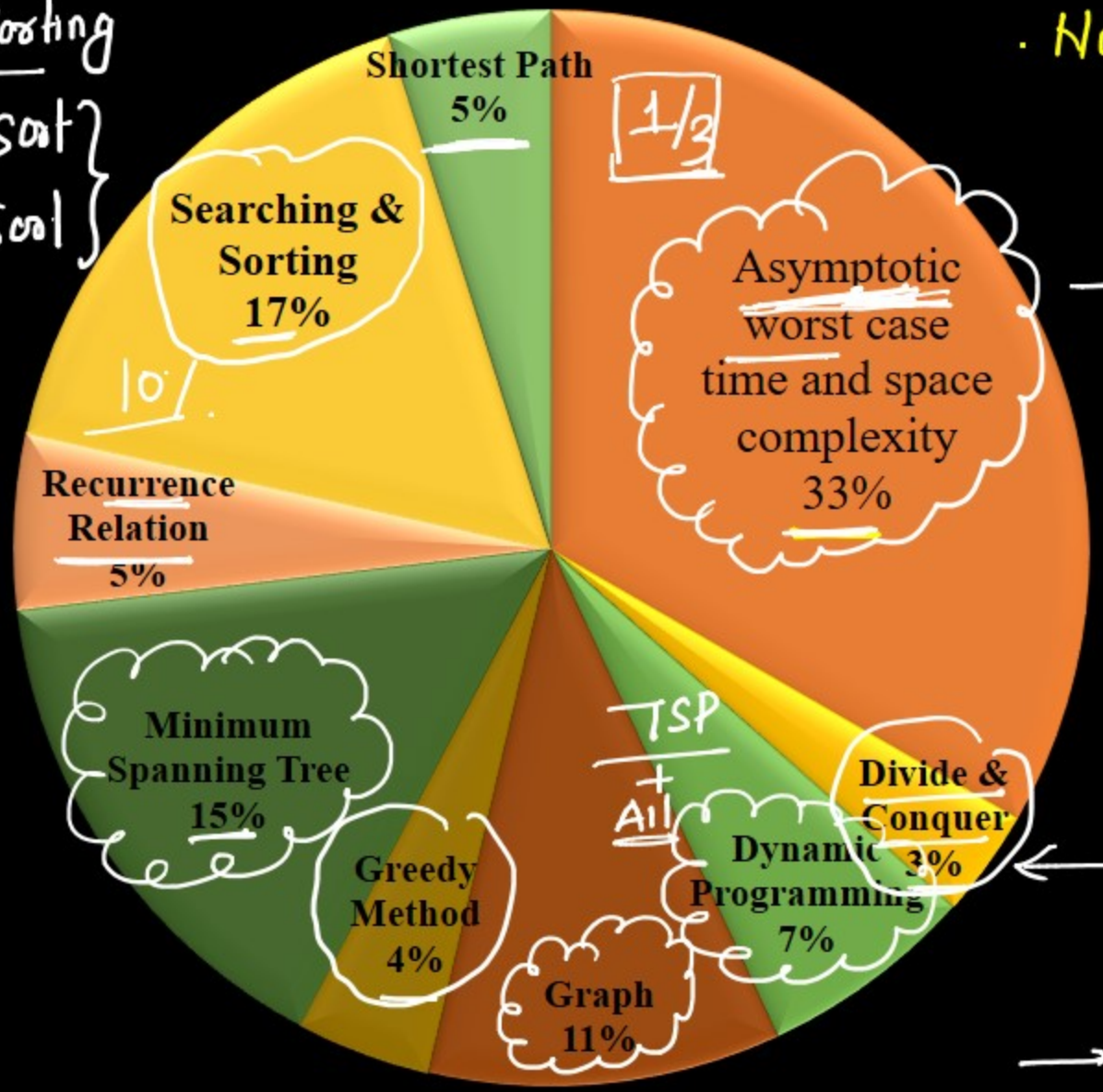
Greedy Method

Data structure :-

Explain :-

Weightage Analysis

Basic Sorting
- quick Sort
- Merge Sort



. No. of questions asked till 2021

— question from data structure

BST + Heap

Work sheet

Work + wall book + text book + GQB

Master Method

3% + quick/Merge + Recurrence Relation

18%

→ 41% + MST + Shortest path

Prerequisite

- I will cover DS - aspect
- Summation Series, Logarithms, Induction
- Probability
- Discrete Mathematics – & Graph function
  - Function
  - Graph

5 subjects 15% Aptitude

50% 13% + EM (Engineering Maths)
+ Discrete Maths

22% + Data structure + programming
+ Algorithm

DL + — 3 Question
CD + 3 Question

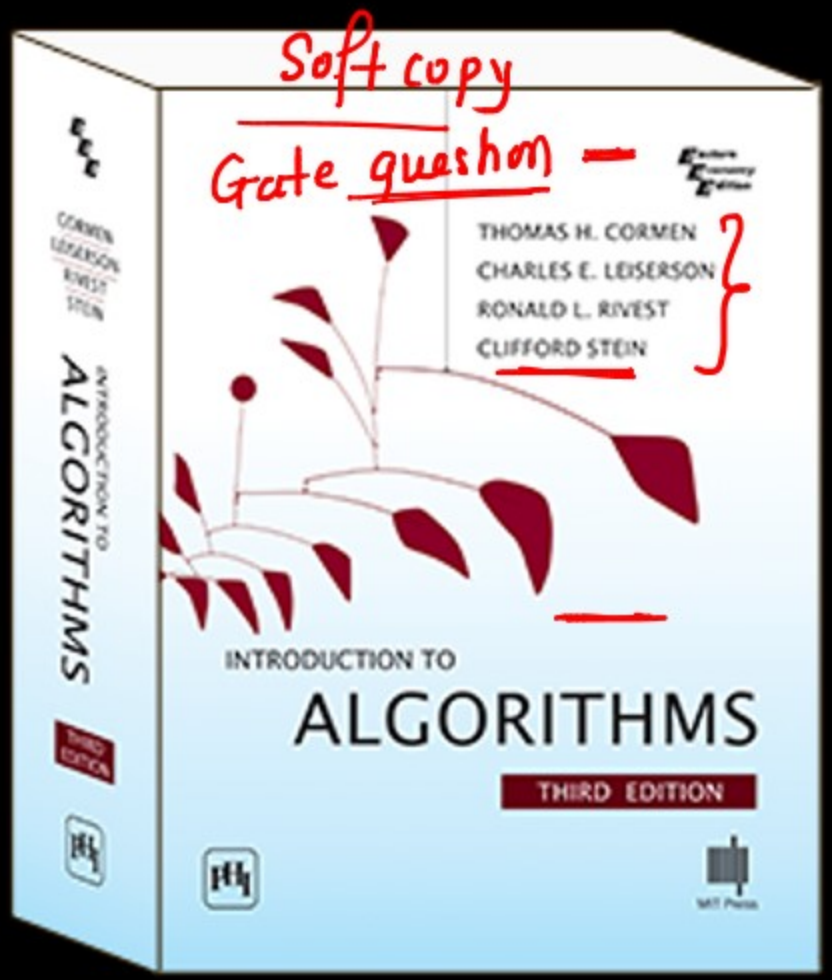
TOC + DBMS + CO + OS + CN

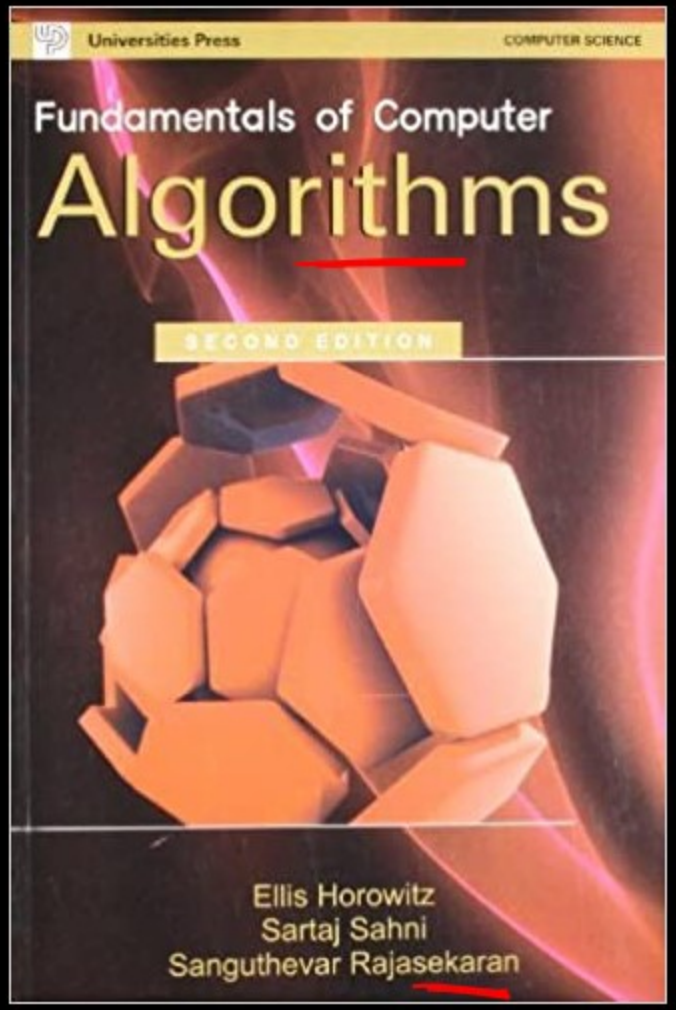
Equal weightage

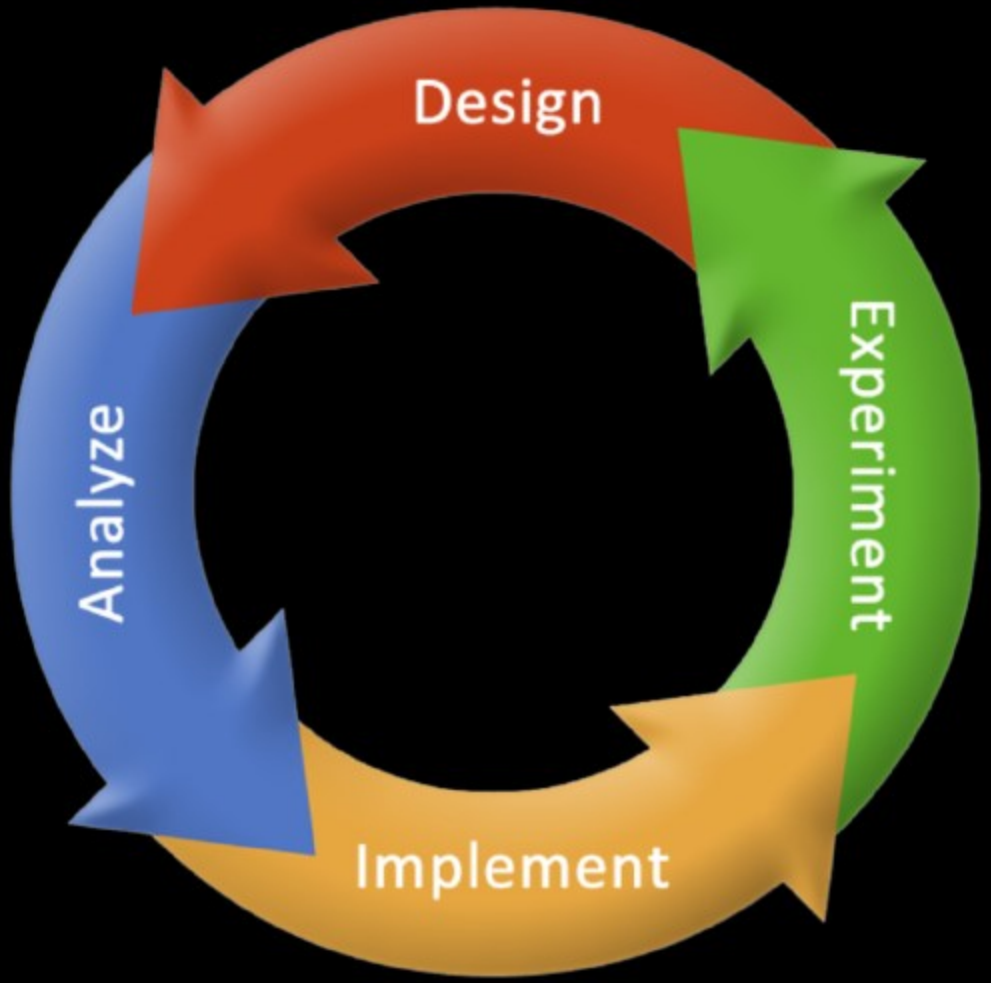
Topics to be Covered

- Design and Analysis of Algorithm

Books :- Soft


Soft copy
Gate question
THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION


Universities Press
COMPUTER SCIENCE
Fundamentals of Computer
Algorithms
SECOND EDITION
Ellis Horowitz
Sartaj Sahni
Sanguthevar Rajasekaran



Algorithmic Analysis

Analysis - Result Analysis
Paper Analysis
Syllabus Analysis

Expected No. of marks : 8-10 marks

33%

Introduction

- Definition, Properties + ~~TOC~~ theory of computation
- Why Analysis is Required
- How to Analysis
- Types of Analysis
- Framework of Analysis + Start An

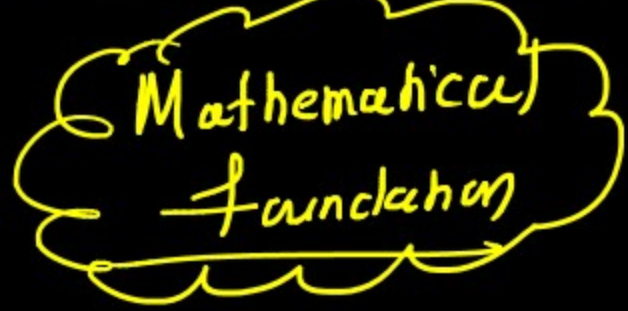
7.30

+

Asymptotic Notation + Recurrence Relation + [Summation & Series]

× × =

Comparison of function Solving +



Historical Background

One of Khwãrezm's most famous residents was Muhammad ibn Mūsa al-Khwarizmī, an influential 9th century Persian scholar, astronomer, geographer, and mathematician known especially for his contributions to the study of algebra. Indeed, the latinization of his name, which meant 'the native of Khwãrezm' in Persian, gave English the word *algorithm.*

Definition of Algorithm

- Solving a problem
- finite Sequence of Instruction/steps
  whe followed it Helps to solve a problem.

problem : Computational problem

- Decidability [Solvable] — longer time
- unsolvable — Undecidability — No Algorithm exists

[Algorithm Exists for it] — NP Hard / NP-complete

Definition of Algorithm

- An *algorithm* is a finite set of instructions that, if followed, *solves* particular Problem.

But we are talking about solving Computational problem!!

Properties of Algorithm

- Input : zero or more Input
- output : at least one output / side effect.
- Definiteness : Each statement must be clear (~~unamg~~ unambiguous)
- finiteness : The steps must be completed in finite amount of time
- Termination

Software Development Life Cycle

Why Analysis is Required

Solution :-

1. Performance - word
   Rotates Image
   1/2 min
   How much time Required to perform the task

2. Resource utilization - App - Space (Memory)
   - Battery
   - Space (Physical)
   - Data (Mobile)
   - ~~pay~~

3. Comparison of algorithm :-
   user - Sol_1 user$_2$ Sol_2

Why Analysis is Required

Analyse – Solution / Algorithm

- Performance
  - Time take to perform spell Correction, rotating an image
- Utilization of Resources
  - Power
- Comparison of Algorithms

How to Analyze

Solution / Algorithm

Software (Compiler)

- Posteriori Analysis :

Algorithm → Implement the Algorithm (C programming Language)

→ M/c (Hardware) ISA (Hardware)

Run the program. → Estimate the time

- Priori Analysis * :- Algorithm → [Mathematical tool (Asymptotic Notation) performance of Algorithm]

Theoritical Analysis

How to Analyze

The complexity of an algorithm can be calculated by using two methods:

- Posteriori Analysis
- Priori Analysis

Difference between Posteriori analysis and A Priori Analysis:

Posteriori Analysis	Priori Analysis

Difference between Posteriori analysis and A Priori Analysis:

Posteriori Analysis

- Posteriori analysis is a relative analysis.
- It is dependent on language of compiler and type of hardware.
- It will give exact answer.

Priori Analysis

- Priori analysis is an absolute analysis.
- It is independent of language of compiler and types of hardware.
- It will give approximate answer.

Difference between Posteriori analysis and A Priori Analysis:

A Posteriori Analysis

- It doesn't use asymptotic notations to represent the time complexity of an algorithm.

A Priori Analysis

- It uses the asymptotic notations to represent how much time the algorithm will take in order to complete its execution.

Difference between Posteriori analysis and A Priori Analysis:

A Posteriori Analysis

- The time complexity of an algorithm using a posteriori analysis differ from system to system.

A Priori Analysis

- The time complexity of an algorithm using a priori analysis is same for every system.

Difference between Posteriori analysis and A Priori Analysis:

A Posteriori Analysis

- If the time taken by the algorithm is less, then the credit will go to compiler and hardware.

A Priori Analysis

- If the program running faster, credit goes to the programmer.

Theoritical +

Small Break - 7.45 pm

- Coreman

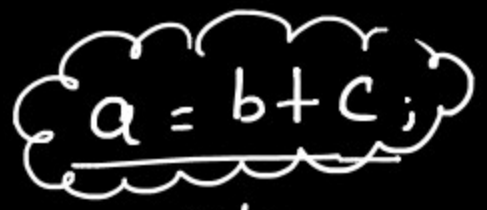
:- Soft copy

. Insertion Sort

[A Priori Analysis :]

The RAM Model of Computation

Computing time Complexity Algorithm (Priori Analysis)



M/C

✳ No. of programming steps taken by an algorithm/ program.

M/c dependent

```
Mov R1 b    R1 ← b
MOV R2 c    R2 ← c
Add R1, R2 , R1 ← R1 + R2
Mov a R1
```

a Single program

[a = b + c] → (1)

one program

unit of I/O operation

✳ Loop : multiple steps

✳ Memory Access 1 programming step

The RAM Model of Computation

Theoritical M/c

- he RAM (Random Access Machine) model of computation measures the run time of an algorithm by summing up the number of steps needed to execute the algorithm on a set of data. The RAM model operates by the following principles:

The RAM Model of Computation

The RAM Model of Computation

The RAM (Random Access Machine) model of computation measures the run time of an algorithm by summing up the number of steps needed to execute the algorithm on a set of data. The operates by RAM model he following principles

- One programming steps consists of basic logical or arithmetic operations (+, *, =, if, call can also be considered to be simple operations that take one time step).

Sum

a = b+c; ← Single programming

n+1

for i = 1 to n

n+1+n

a = b+c;

2n+1

n+1

for i = 1 to n

fun() - 2

1 step

fun() {

for (i = 1 to m)

m+1+1

}

fun() {

a = b+c

return b;

}

The RAM Model of Computation

- Loops and subroutines are complex operations composed of multiple time steps.
- All memory access takes exactly one time step.

framework

Count No. of programming steps taken by a program/ Algorithm

Framework of Analysis

Step count for given Input

↑

{programming step}

Framework of Analysis

- Recursive and non-Recursive Program
- Step count
  - The primitive operation (The basic Operation)

Time Complexity

Time Complexity == Step count

- Time complexity of the program/ Algorithm is calculate as number of *program steps* needed to complete the algorithm.

total No. of programming steps represents the time complexity

Step count

Number of Steps Taken By An Algorithm

n – Size of the array

Sum of an Array Elements	Total Steps
Algorithm Sum (a,n){	
s:= 0.0;	1
for i:=1 to n do{	n+1 ← Last time condition is false
s = s + a[i];	n
} return s;	1
}	Sum+1
total	2n+3 ←

Name of the array

1

Size

for (i=1; i<=n; i++)

Number of Steps Taken By An Algorithm

Sum of Matrix	Total Steps
Algorithm Add(a,b,c,n,n){	
for i := 1 to n do	
for j := 1 to n do	
c[i,j] :=a[i,j] + b[i,j];	
}	
Total	

Step count

if( )

Sahani

Minimum step count :- $2n^2+2n+1$

Maximum step count :- $2n^2+2n+1$

Exact Bound

upper bound = Lower bound

= Exact bound

Primitive

Number of Steps Taken By An Algorithm

Asymtoh'c

$O(n^2)$

Steps Count

Sum of Matrix	Total Steps
`Algorithm Add(a,b,c,n,n){`	
`for i := 1 to n do {` → n	n+1 = n+1
`for j := 1 to n do` {	n × (n+1) = n^2+n
`c[i,j] :=a[i,j] + b[i,j];`	n^2 = n^2
}	$2n^2+2n+1$ Minimum = Maximum
Total	↑ Maximum

Number of Steps Taken By An Algorithm

Addition of two matrixes

```
Algorithm Add(a,b,c,n,n)
{
      for i := 1 to n do
            for j := 1 to n do
                  c[i,j] :=a[i,j] + b[i,j];
}
```

H.W

Number of Steps Taken By An Algorithm

	Total steps	
	n = 0	n > 1
Algorithm Rsum (a,n){		
if (n <= 0) then	1	1
return 0.0;	1	0
else		
return RSum(a, n - 1) +a		
}		

Recursive Algorithm

Number of Steps Taken By An Algorithm

	Total steps	
	n = 0	n > 1
`Algorithm Rsum (a,n){`	0	0
`if (n <= 0) then`	1	1
`return 0.0;`	1	0
`else`	0	0
`return RSum(a, n - 1) +a`	0	1+1
`}`	2	2 +x
	x = t *rsum* (n-1)	

The Time Complexity of the Program

Summation of step count , function of Input size

The time complexity of the program (Number of program step taken by algorithm) can be expressed an function of input $T(n)$, Where n is the input size

Summation — $2n+3$

Addition of matrix — $2n^2+2n+1$

$n+1$
$n(n+1)$
n^2

$\frac{1}{n}(1+2+3+4+\cdots+n)$

$\frac{1}{n} * \frac{n(n+1)}{2} = \frac{(n+1)}{2}$

Types of Analysis

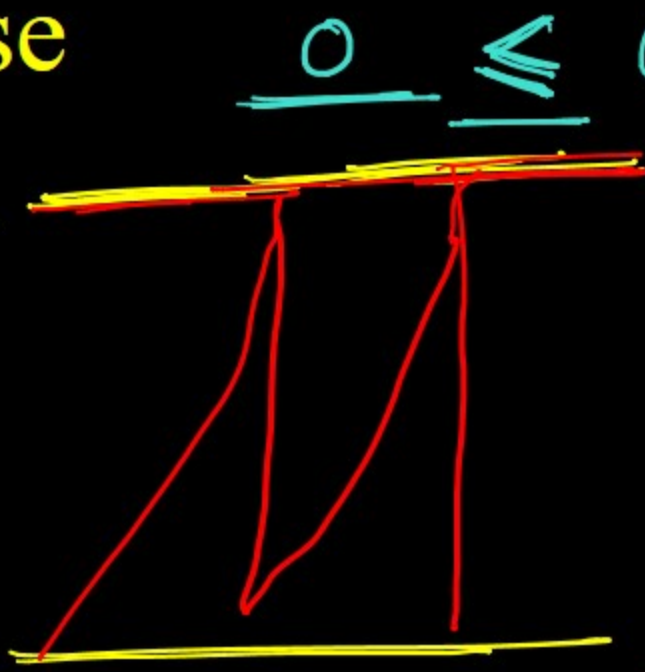
- Best Case Analysis : Minimum no. of programming step required for termination.
- Average Case Analysis $\frac{1}{n}[1+2+3+\cdots+n] = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$
- Worst Case Case Analysis Maximum No. of programming step required for termination
- Amortized Analysis

Best Case

- Linear Search

R programming
Step count / time complexity

Clear -



O ≤ worst case
upper bound
Maximum

Best case
Lower bound
Minimum time

Ω ≤

- upper bound = Lower bound = Exact bound

Worst Case

Average Case

Amortized Analysis

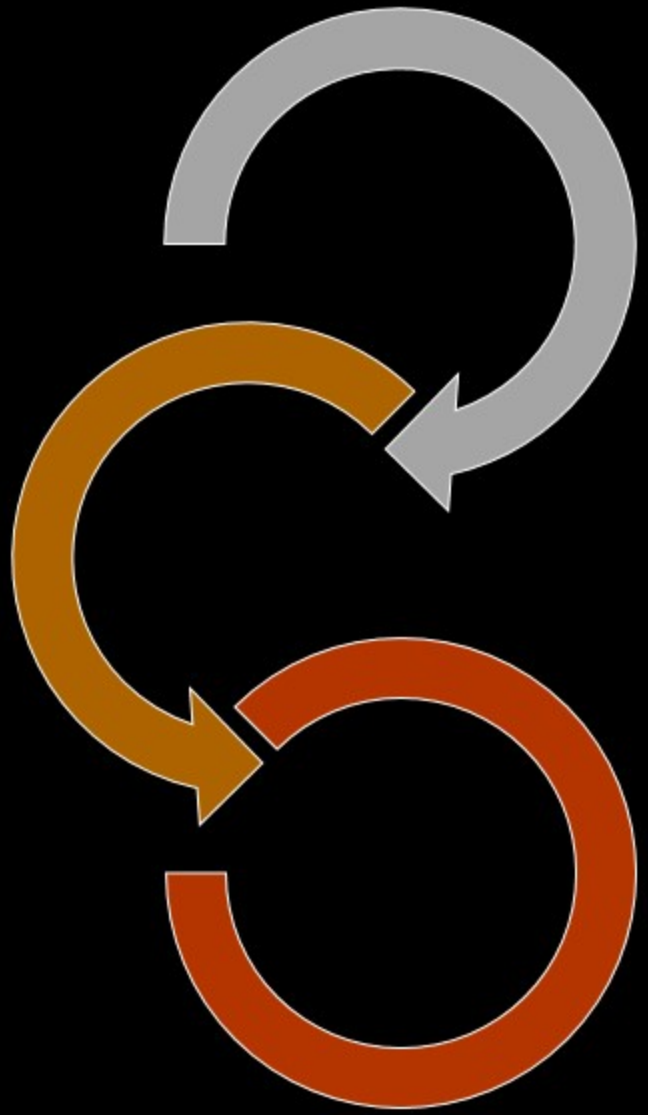
Amortized Analysis

- This analysis is used when the occasional operation is very slow, but most of the operations which are executing very frequently are faster. Data structures we need amortized analysis for Hash Tables, Disjoint Sets etc.

Amortized Analysis

- Aggregate Analysis

Relationship Among Best, Average and Worst Case



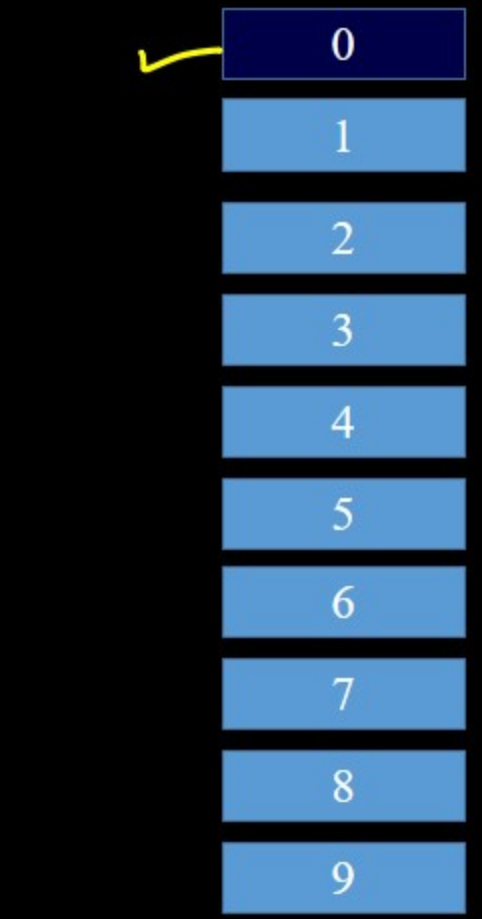
Algorithmic Complexity

Linear Search

C- program – unordered Array

0 - ... n-1

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++) {
        if (arr[i] == x)
            return i;  ← terminate
    }
    return -1;
}
```



ordered Array : Binary Search

Linear Search

Step count : Smallest No. of steps required for termination of Search (4)

1+11+10+10 32 ⇒ 33
+1

```
int search(int arr[], int n, int x){
    int i;①      ⑪      ⑩      10
    for (i=0; i<n; i++)
        if (arr[i] == x)  ③  10
            return i;④
    return -1; ←0
}
```

- Comparison –

  1 Comparison

  Sean Element that is been Serched is in 1st posion

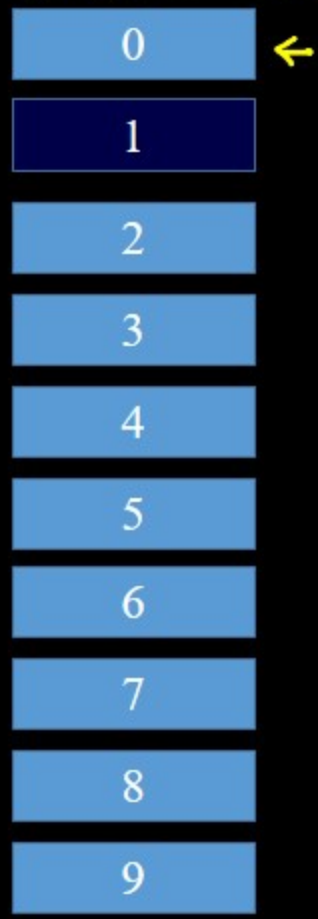
- Worst case Analysis :

  Maximum No. of comparison – N

  Last position – element is Not present

  N

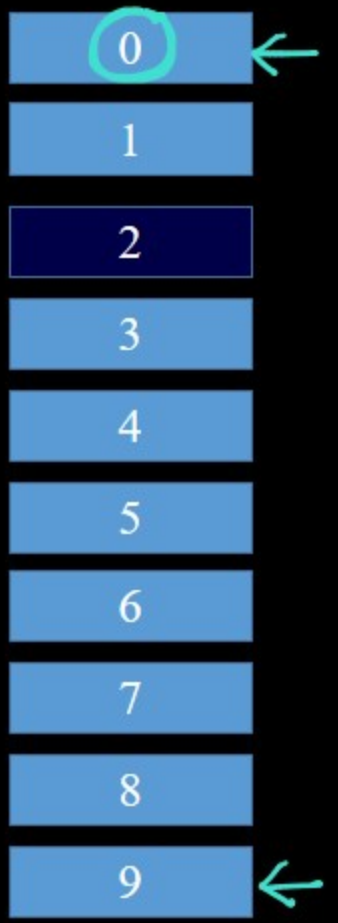
- Best case



Linear Search

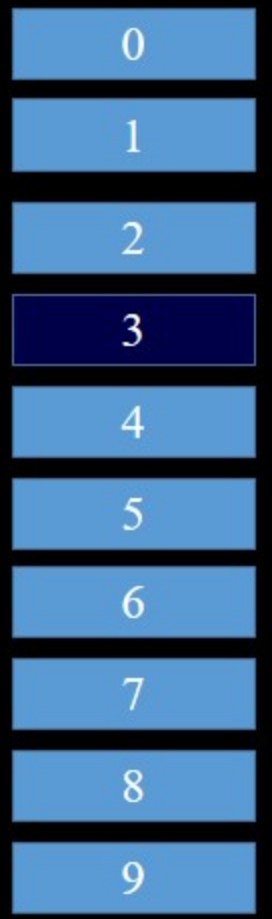
```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```

Condition check



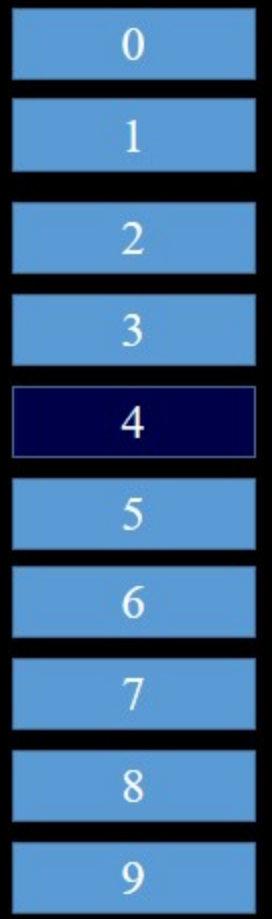
Linear Search

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



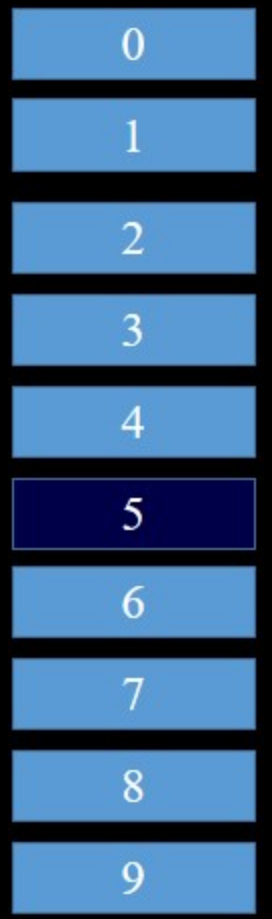
Linear Search

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



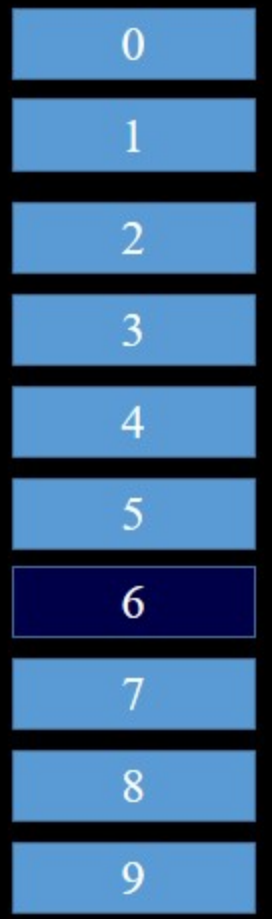
Linear Search

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



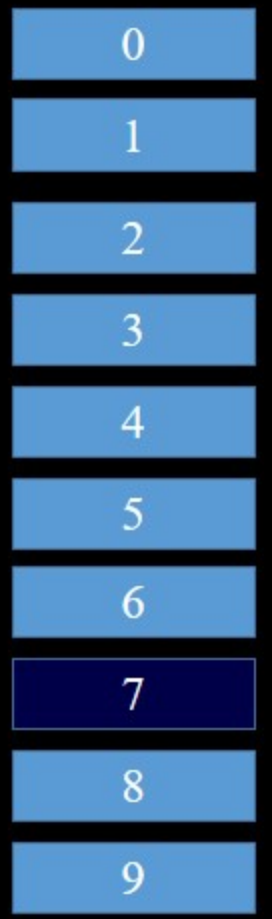
Linear Search

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



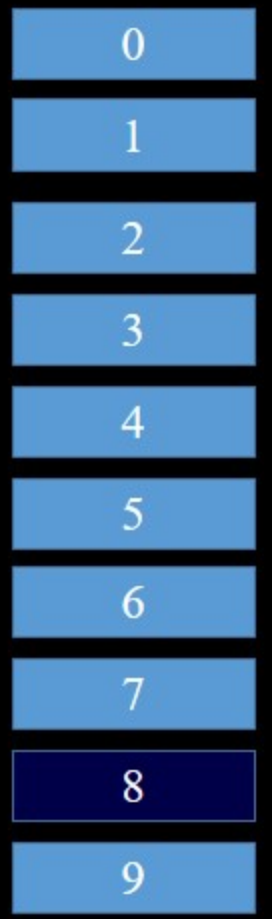
Linear Search

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



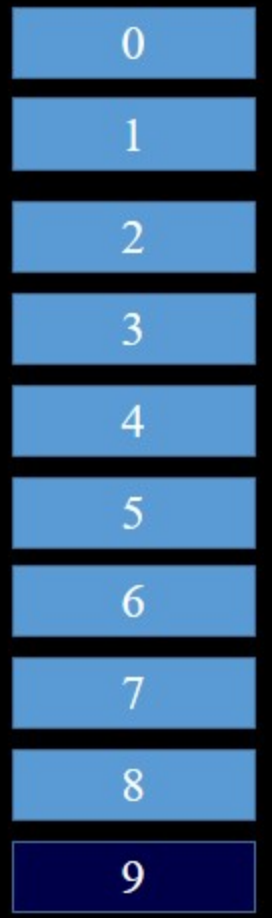
Linear Search

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



Linear Search

```
int search(int arr[], int n, int x){
    int i;
    for (i=0; i<n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```



Bound

Step count

- Upper bound – Maximum time
- Lower Bound – Minimum
- Exact bound – Max = min

Running

Worst case – W(n)

Average case – A(n)

Best case – B(n)

W(n), A(n), B(n)

Related with each other

$$B(n) \leq A(n) \leq W(n)$$