16IT302 – DESIGN AND ANALYSIS OF ALGORITHM

- Pre- Requisite for DAA – Algorithm / DS
- What you are going to Study in DAA
  - Recipe for food preparation
  - Algorithms (steps) are instructions for building programs
  - Designing Algorithm
  - Analyzing Algorithm
- Why Designing and Analyzing Algorithm is important.
  - Without a proper blueprint you cannot construct a house
  - Proper design and analyzing of algorithm will give a best solution for a problem
  - Requirement (Algorithm should be designed)

Problem how to solve steps to solve Analyze

Why Designing and Analyzing Algorithm? Example

- Example: searching
- Search 1

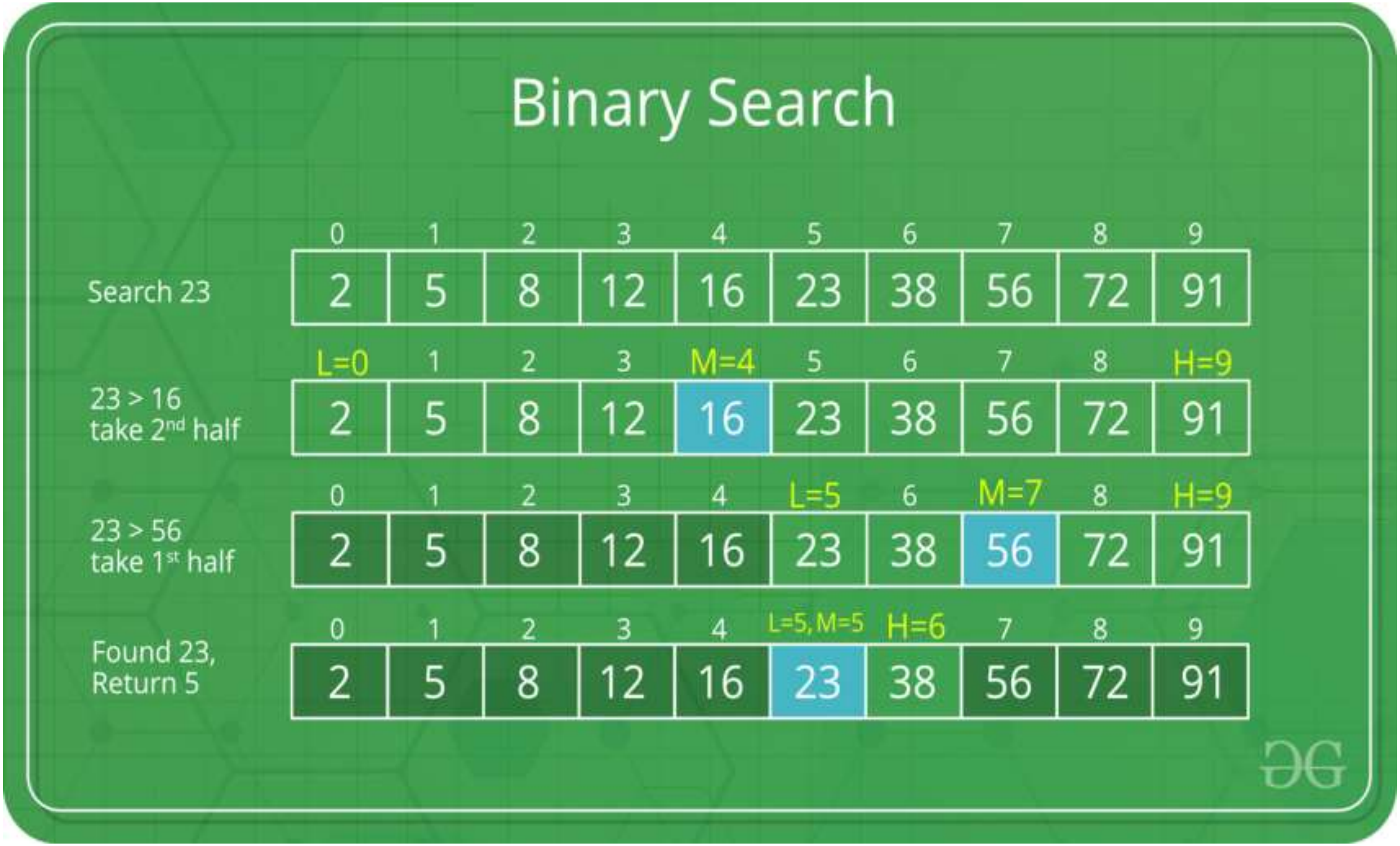
1	2	3	4	5

- Search 5

1	2	3	4	5

- Search technique
  - Google - 500-600 times each year search algorithm is changed
  - MS Word - Boyer – Moore algorithm

Binary Search



SYLLABUS

UNIT I	INTRODUCTION	9+6
Notion of an Algorithm – Fundamentals of Algorithmic Problem Solving – Important Problem Types – Fundamentals of the Analysis of Algorithm Efficiency – Analysis Framework – Asymptotic Notations and its properties – Mathematical analysis for Recursive and Nonrecursive algorithms.		
UNIT II	BRUTE FORCE AND DIVIDE-AND-CONQUER	9+6
Brute Force: Insertion Sort, Bubble Sort, Sequential Search, Closest-Pair and Convex-Hull Problems-Traveling Salesman Problem – Knapsack Problem - Assignment problem. Divide and conquer methodology: Merge sort – Quick sort – Binary search – Multiplication of Large Integers – Strassen's Matrix Multiplication		
UNIT III	DYNAMIC PROGRAMMING AND GREEDY TECHNIQUE	9+6
Dynamic Programming: Computing a Binomial Coefficient – Warshall's and Floyd's algorithm – Optimal Binary Search Trees – Knapsack Problem and Memory functions. Greedy Technique Prim's algorithm-Kruskal's Algorithm - Dijkstra's Algorithm-Huffman Trees – Job Sequence Scheduling		
UNIT IV	ITERATIVE IMPROVEMENT	9+6
The Simplex Method-The Maximum-Flow Problem – Maximum Matching in Bipartite Graphs- The Stable marriage Problem.		
UNIT V	COPING WITH THE LIMITATIONS OF ALGORITHM	9+6
Limitations of Algorithm - Lower-Bound Arguments-Decision Trees-P, NP and NP-Complete Problems – Coping with the Limitations – Backtracking: n-Queens problem – Hamiltonian Circuit Problem – Subset Sum Problem-Branch and Bound: Assignment problem – Knapsack Problem – Traveling Salesman Problem-Approximation Algorithms for NP Hard Problems		

TEXT BOOKS	
1	Anany Levitin, "Introduction to the Design and Analysis of Algorithms", Pearson Education, 3rd Edition, 2012. (Unit I,II,III,IV,V)

UNIT I – NOTION OF ALGORITHM

- Algorithm
  - unambiguous instructions to solve a problem
  - Solution to a problem / procedure for getting that solution
  - Different forms
  - Single problem – multiple solutions – multiple algorithms – requirements
- instructions – computers / human beings
- Example :

greatest common divisor of 2 numbers (GCD) – 3 methods

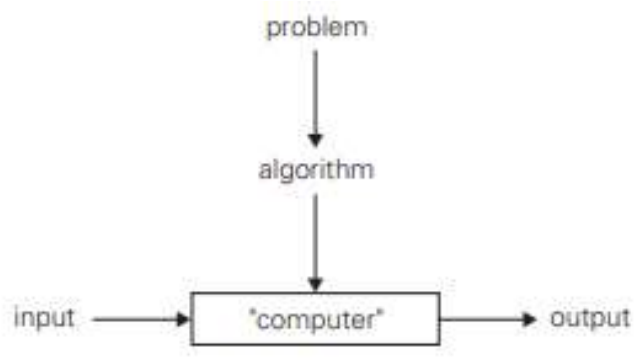


Fig: Notion of Algorithm

GCD of two numbers – Euclid's Algorithm

- GCD of two numbers
  - Euclid's algorithm
  - Consecutive integer checking algorithm
  - Middle school procedure
  - ***Euclid's algorithm***

gcd(m,n)= gcd(n, m mod n)

Example1: gcd (60,24) = gcd (24, 60 mod 24)
= gcd (24, 12)
= gcd (12, 24 mod 12)
= gcd (12,0)

Example 2: gcd (70, 35)

Example 3: gcd (30,14) = gcd (n, m mod n)
= gcd (14, 30 mod 14)
= gcd (?)

Euclids Algorithm

Iteration	m	n	r = m % n
1	50	35	15
2	35	15	5
3	15	5	0
4	5 (GCD)	0 (Stop)	

GCD of two numbers – Euclid's Algorithm

Euclid's algorithm for computing gcd(m, n)

Step 1 If $n = 0$, return the value of m as the answer and stop; otherwise, proceed to Step 2.

Step 2 Divide m by n and assign the value of the remainder to r.

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

Alternatively, we can express the same algorithm in pseudocode:

ALGORITHM *Euclid*(m, n)

```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

GCD of two numbers – Consecutive Integer Checking Algorithm

- GCD – common divisor cannot be greater than the smaller of these numbers ***t = min {m, n}***
- gcd (60,24) □ 24 □ decrease 24 by 1□ 23□ 22□ …….□ 12

m	***n***	***t***
60	24	24
60	24	23
60	24	22
60	24	21
60	24	20
60	24	19
60	24	18

m	***n***	***t***
60	24	17
60	24	16
60	24	15
60	24	14
60	24	13
60	***24***	***12***

Consecutive Integer Checking Algorithm

Step 1 Assign the value of min{m, n} to t.

Step 2 Divide m by t. If the remainder of this division is 0, go to Step 3; otherwise, go to Step 4.

Step 3 Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to Step 4.

Step 4 Decrease the value of t by 1. Go to Step 2.

GCD of two numbers – Middle School procedure

Step 1 Find the prime factors of m.

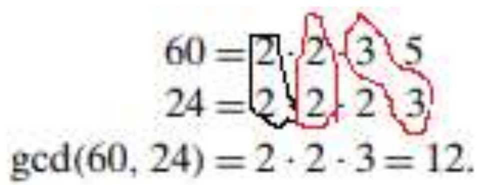
Step 2 Find the prime factors of n.

Step 3 Identify all the common factors in the two prime expansions found in Step 1 and Step 2. (If p is a common factor occurring p_m and p_n times in m and n, respectively, it should be repeated $\min\{p_m, p_n\}$ times.)

Step 4 Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

Thus, for the numbers 60 and 24, we get

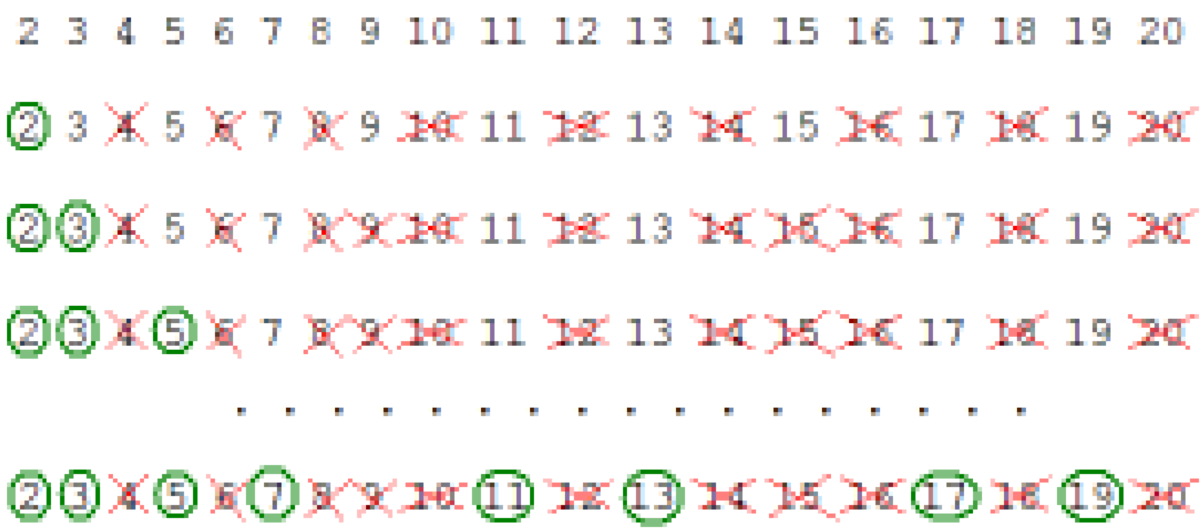
$$60 = 2 \cdot 2 \cdot 3 \cdot 5$$
$$24 = 2 \cdot 2 \cdot 2 \cdot 3$$
$$\gcd(60, 24) = 2 \cdot 2 \cdot 3 = 12.$$



- Middle school procedure – Sieve of Eratosthenes
- *Euclid's Algorithm is Simpler and fast*

GCD of two numbers – Middle School procedure

- Sieve of Eratosthenes – prime factors



Fundamentals of Algorithmic Problem Solving

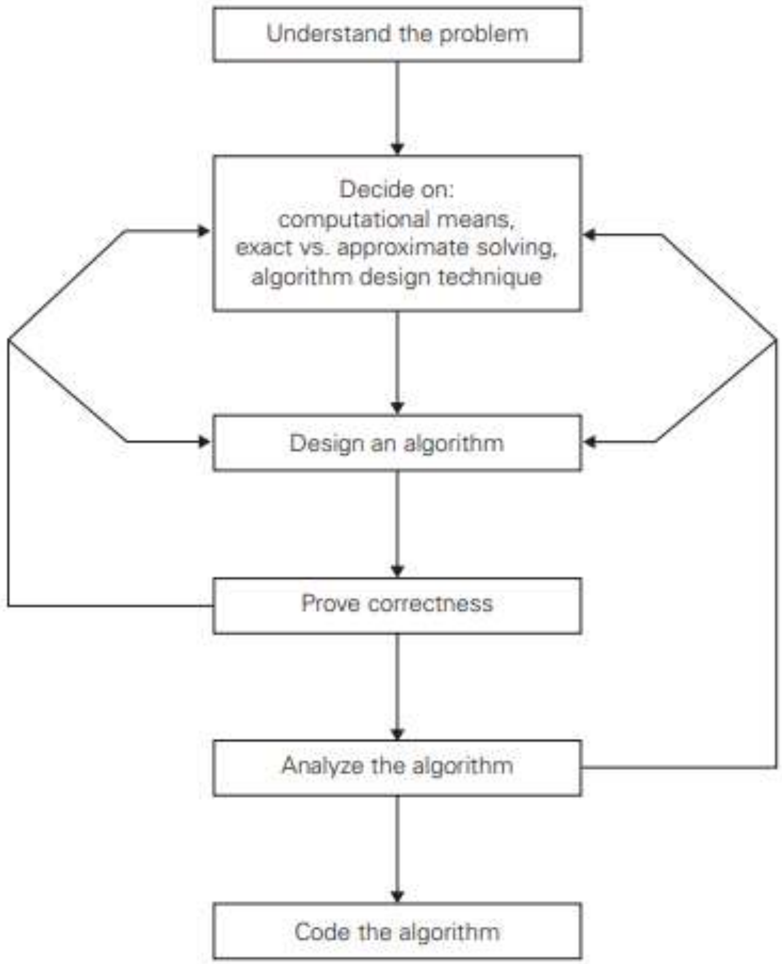


Fig: Algorithm Design and Analysis Process

Fundamentals of Algorithmic Problem Solving

- Understanding the problem
  - What, doubts, examples, use cases
  - Inputs – ***instance of the problem***
- Ascertaining the capabilities of a computational device
  - Random Access Machine – ***Sequential Algorithm***
  - Instructions – concurrent – ***Parallel algorithm***
  - ***Speed and memory*** of computer system – Depends on Application type
- Choosing between exact and approximate problem solving
  - Exact algorithm
  - Approximation algorithm
- Deciding on Appropriate data structures
  - Data Structure – representing the data

Fundamentals of Algorithmic Problem Solving

- Algorithm design techniques
  - Methods/ process to solve a problem
  - Example : Linear (Linear programming)

    vs

    Binary serach (Divide and Conquer programming)
- Methods to specifying an algorithm
  - Natural language
  - Pseudo code (Natural language + programming constructs)
  - Flowchart

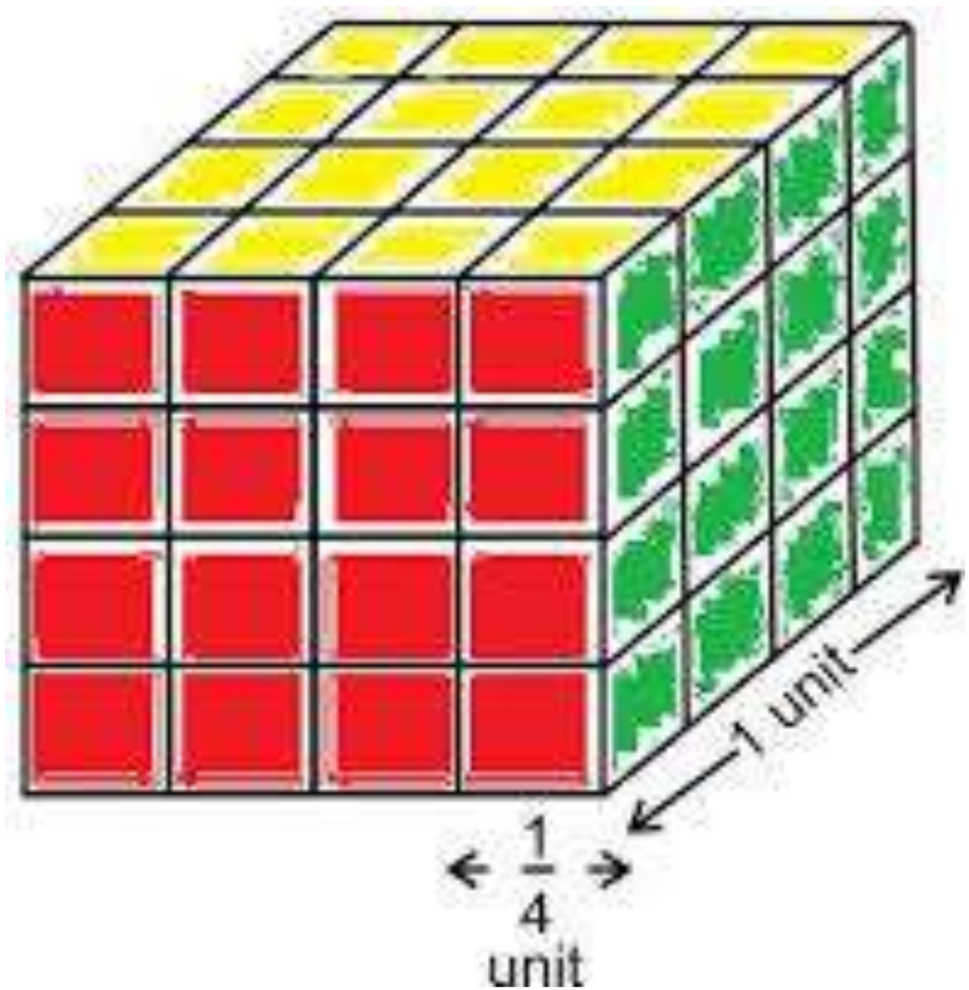
Fundamentals of Algorithmic Problem Solving

- Proving an algorithm's correctness
  - Correctness – GCD (Euclids algorithm) n value decreases and last reaches 0
  - Complex – mathematical induction (iteration)
  - Algorithm incorrect – 1 instance
- Analyzing an algorithm
  - Time efficiency
  - Space efficiency
  - Simplicity – easier to understand and program
  - Generality
- Coding an algorithm

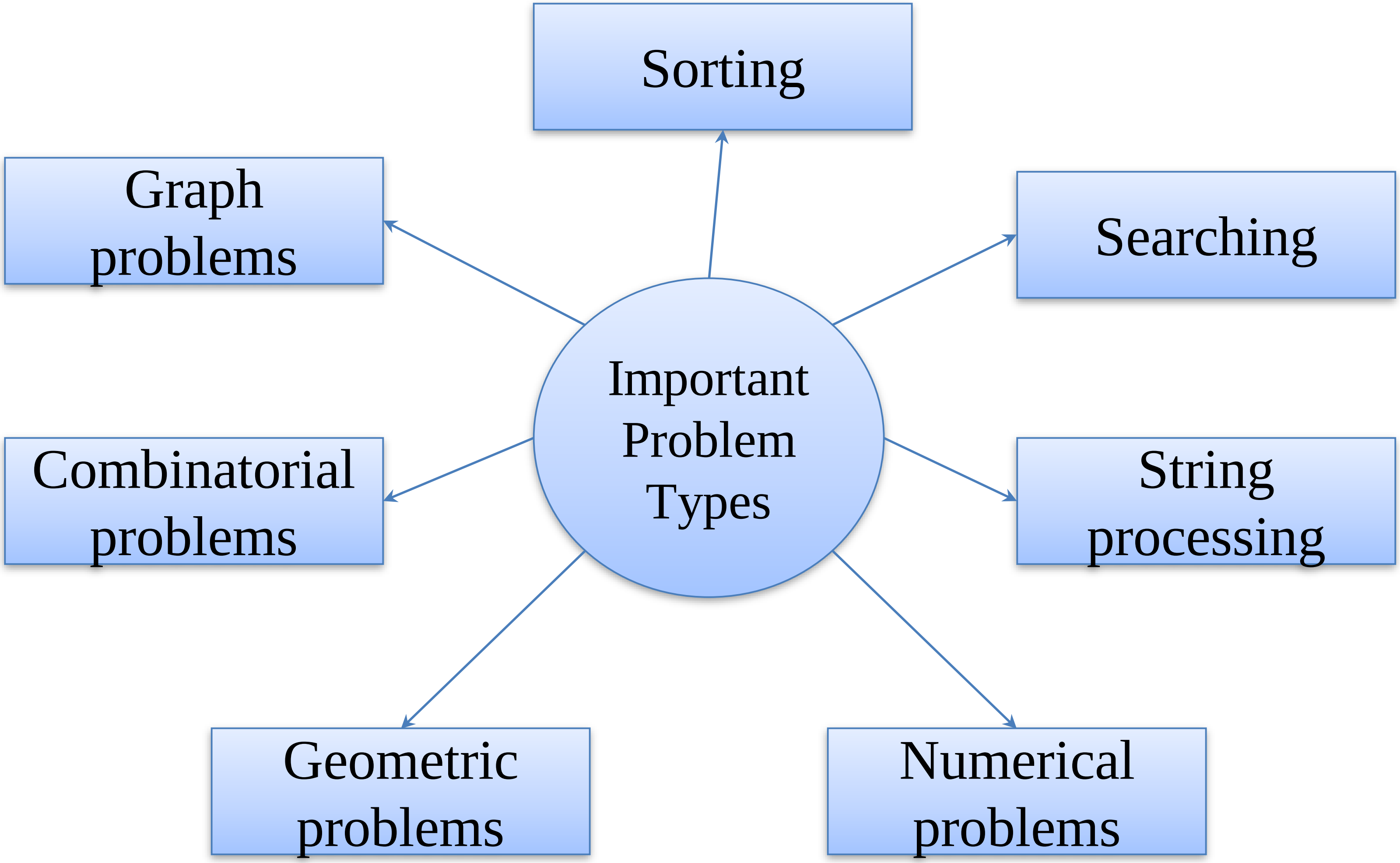
A cube painted red in two adjacent sides and opposite to red it is painted green. The remaining sides painted black.

This cube is divided into 64 equal sized smaller cubes.

How many smaller cubes will be there with no sides colored?


1 unit
1
4
unit

IMPORTANT PROBLEM TYPES

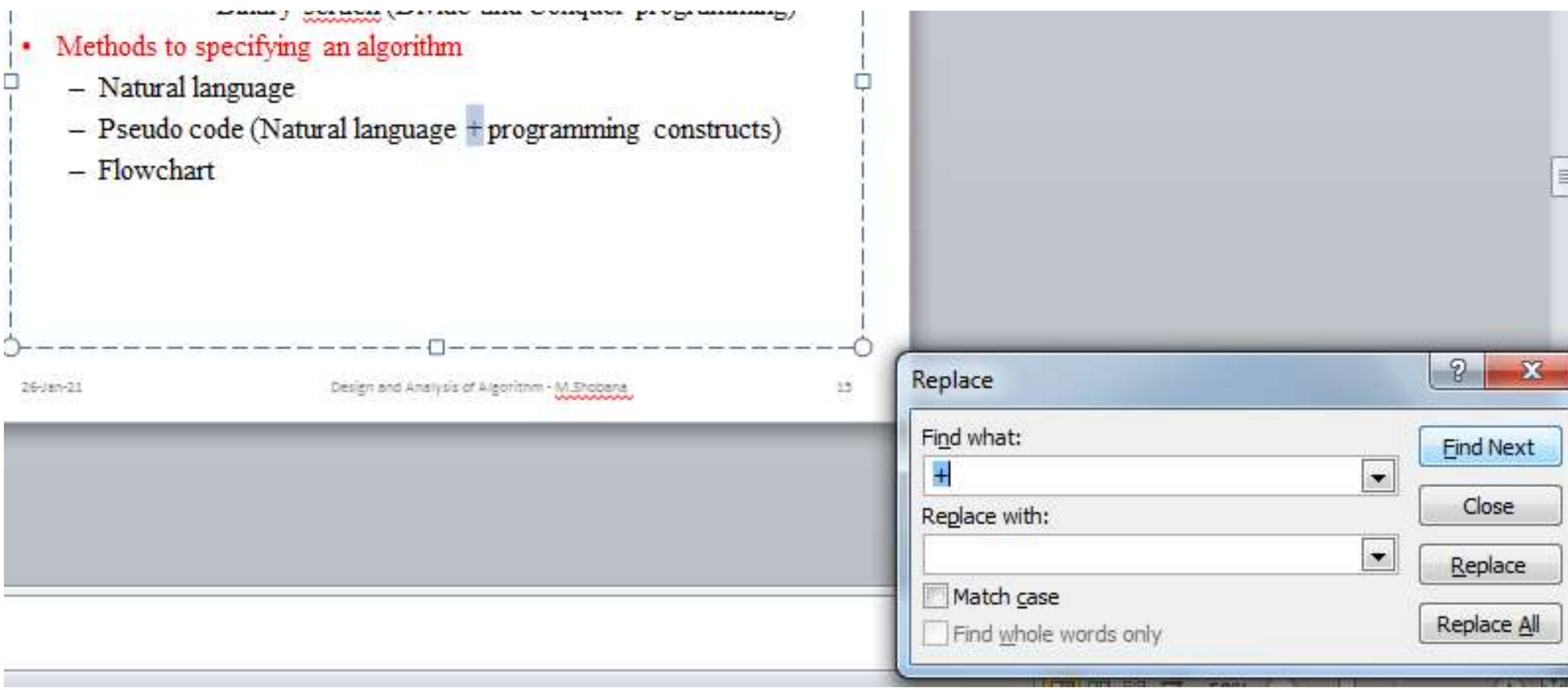


IMPORTANT PROBLEM TYPES

- ***Sorting***
  - Key
  - Colleges, hospitals, office
  - Ease of search - dictionaries, telephone books, class list
  - Several algorithm – not good for all the situations
  - Searching is made easier
  - Properties of sorting algorithm
    - Stable
    - In place

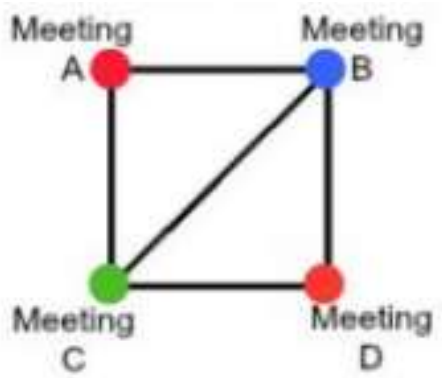
IMPORTANT PROBLEM TYPES

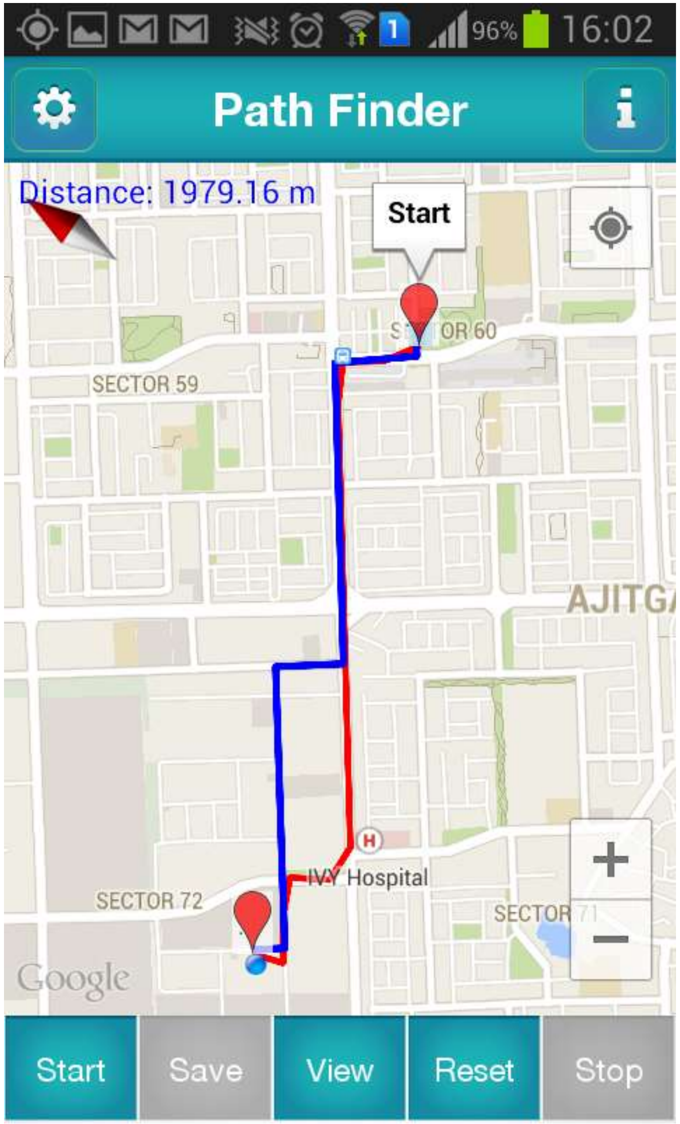
- ***Searching***
  - Search key
  - Several algorithm
- ***String processing***
  - String – string matching



IMPORTANT PROBLEM TYPES

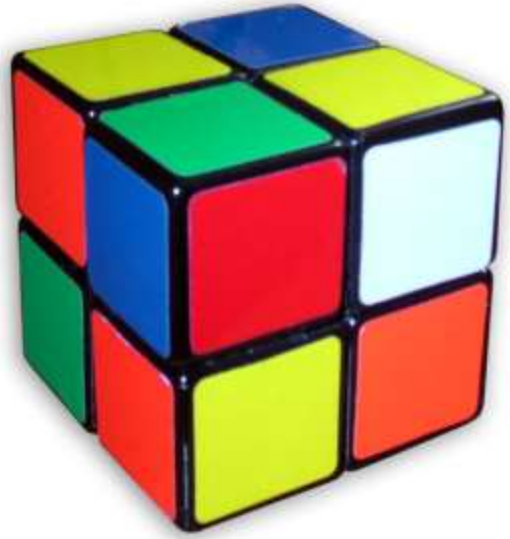
- ***Graph problems***
  - Vertices, edges
  - Graph traversal, shortest path
  - Flight network, Google map – shortest path
  - Ex: travelling salesman problem,
  - Graph coloring – event scheduling





IMPORTANT PROBLEM TYPES

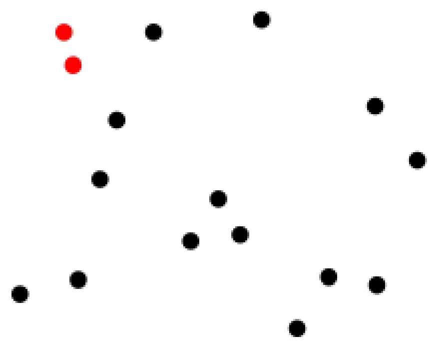
- ***Combinatorial problems***
  - Finding optimal object from a finite set of objects (permutation, combination, subset from a finite set)
  - ***Example:***
    - How many ways are there to make a 2-letter word
    - How many ways are there to select 5 integers from {1, 2, …., 20}



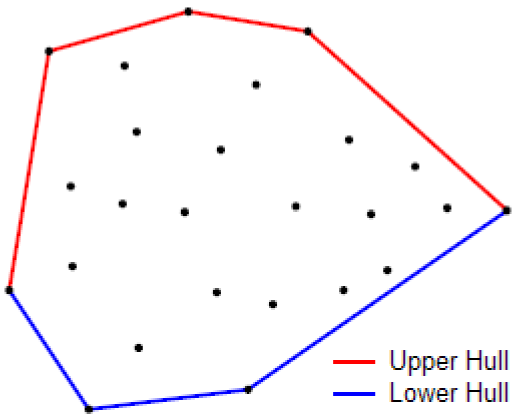
IMPORTANT PROBLEM TYPES

- ***Geometric Problems***
  - Points, lines, polygons
  - Computer graphics (circle,smiley)
  - Example

Closest pair problem



Convex hull problem



Real-time application
Nuclear/chemical leak Evacuation
Tracking Disease epidemic

IMPORTANT PROBLEM TYPES

- ***Numerical Problems***
  - Integrals, functions
  - Approximate
  - Real numbers