Data Structures and Algorithms

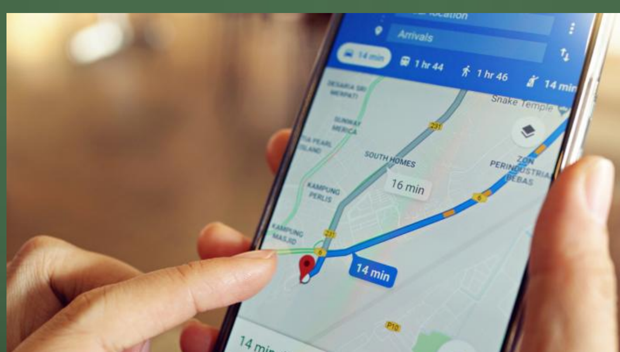
Lecture# 1

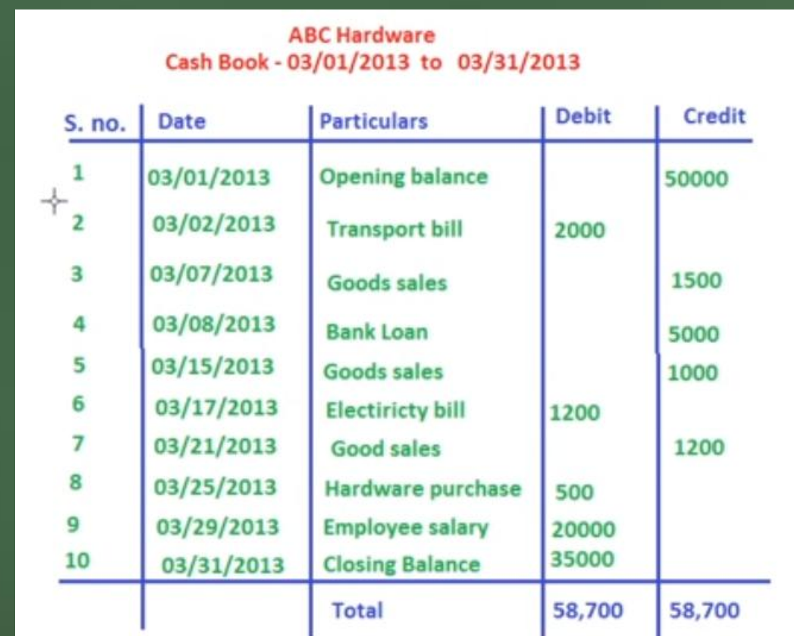
Shikha Mehrotra

Introduction to Data Structures and Algorithms

Introduction to Data Structures and Algorithms

Dictionary

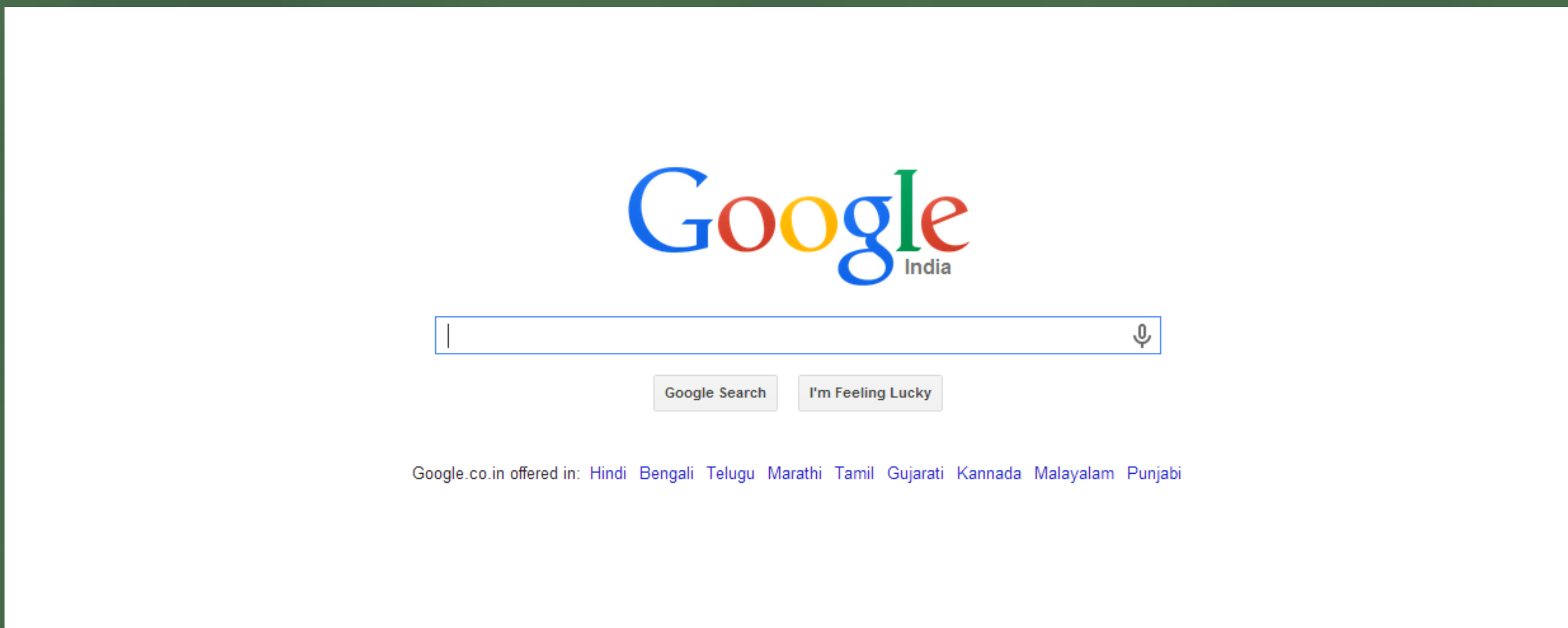




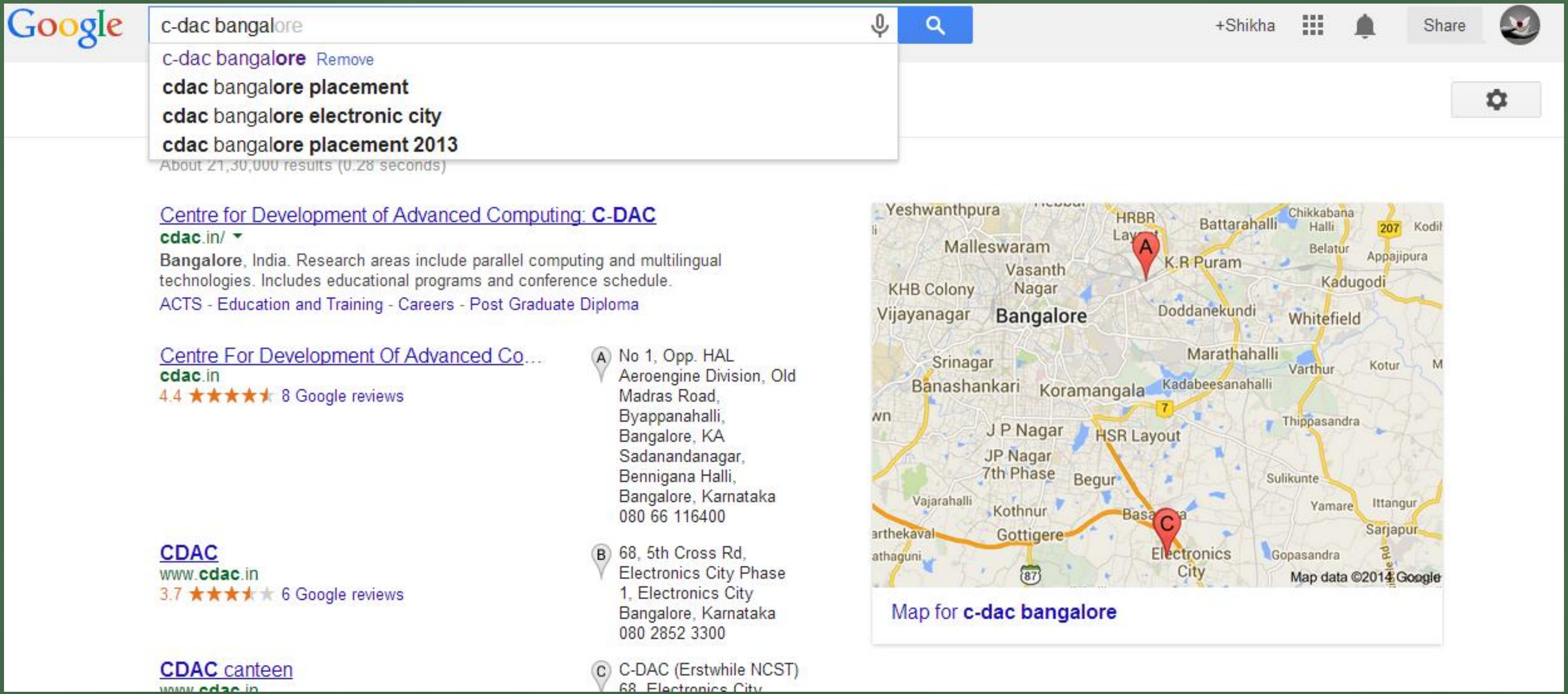
ABC Hardware
Cash Book - 03/01/2013 to 03/31/2013

S. no.	Date	Particulars	Debit	Credit
1	03/01/2013	Opening balance		50000
2	03/02/2013	Transport bill	2000	
3	03/07/2013	Goods sales		1500
4	03/08/2013	Bank Loan		5000
5	03/15/2013	Goods sales		1000
6	03/17/2013	Electiricty bill	1200	
7	03/21/2013	Good sales		1200
8	03/25/2013	Hardware purchase	500	
9	03/29/2013	Employee salary	20000	
10	03/31/2013	Closing Balance	35000	
		Total	58,700	58,700

Example


Google
India
Google Search
I'm Feeling Lucky
Google.co.in offered in: Hindi Bengali Telugu Marathi Tamil Gujarati Kannada Malayalam Punjabi

Example


Google
c-dac bangalore
+Shikha
Share
c-dac bangalore Remove
cdac bangalore placement
cdac bangalore electronic city
cdac bangalore placement 2013
About 21,30,000 results (0.28 seconds)
Centre for Development of Advanced Computing: C-DAC
cdac.in/
Bangalore, India. Research areas include parallel computing and multilingual technologies. Includes educational programs and conference schedule.
ACTS - Education and Training - Careers - Post Graduate Diploma
Centre For Development Of Advanced Co...
cdac.in
4.4 8 Google reviews
No 1, Opp. HAL Aeroengine Division, Old Madras Road, Byappanahalli, Bangalore, KA Sadanandanagar, Bennigana Halli, Bangalore, Karnataka 080 66 116400
CDAC
www.cdac.in
3.7 6 Google reviews
68, 5th Cross Rd, Electronics City Phase 1, Electronics City Bangalore, Karnataka 080 2852 3300
CDAC canteen
C-DAC (Erstwhile NCST)
Map data ©2014 Google
Map for c-dac bangalore

How is it possible?

With the help of efficient

Data Structures and Algorithms

For Developing Software

Media Player

Games

Multimedia

Stock Analysis

Browser

Data Structures

A data structure is a way to store and organize data in a computer, so it can be used efficiently

We talk about data structures as :

1. Mathematical / Logical models (ADT)
2. Implementation

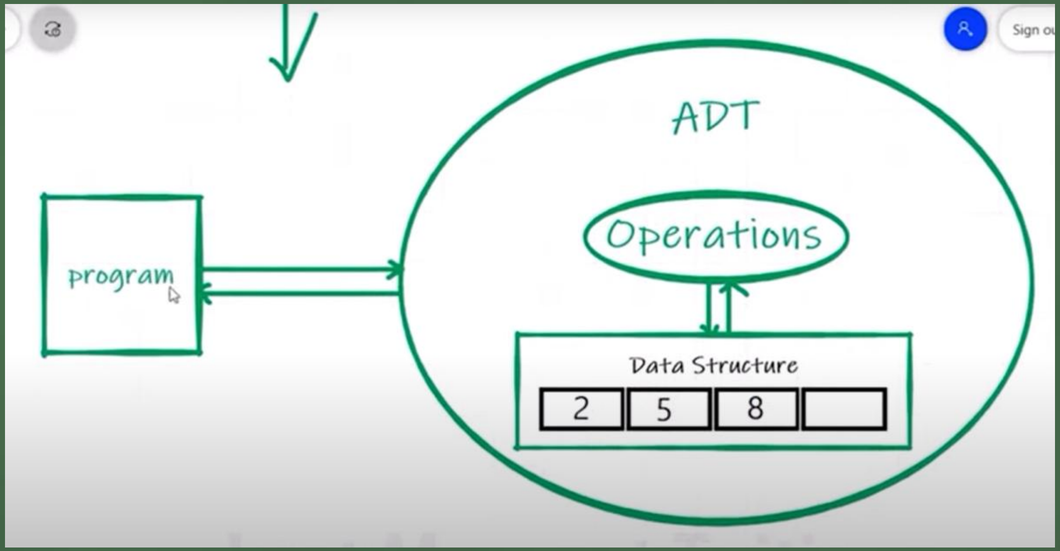
ADT (Abstract Data Type)

- <u>Car</u>
- Engine,
- Gear.
- Petrol tank,
- Tires, etc.

- <u>Operations:</u>
- StartEngine()
- Drive()
- ApplyBreaks()
- IsPetrolAllow()

Array

- Create()
- Get()
- Set()
- Remove()



Data Structures

✓ Turned On/Off
✓ Receive Signals
✓ Play Audio / Video

Abstract View

ADT

↓

List

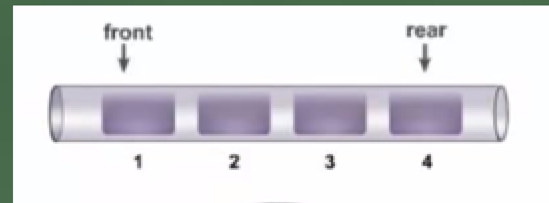
- Store a given number of elements of given typ
- Read Elements by position
- Modify elements by position

Arrays

Concrete Implementation

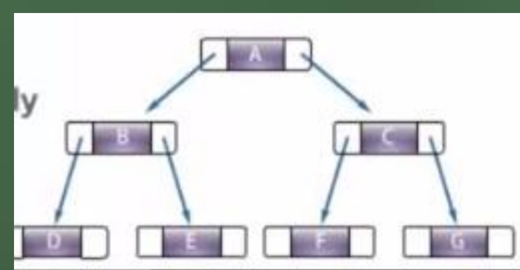
Data Structure Organize the Data

- Linearly



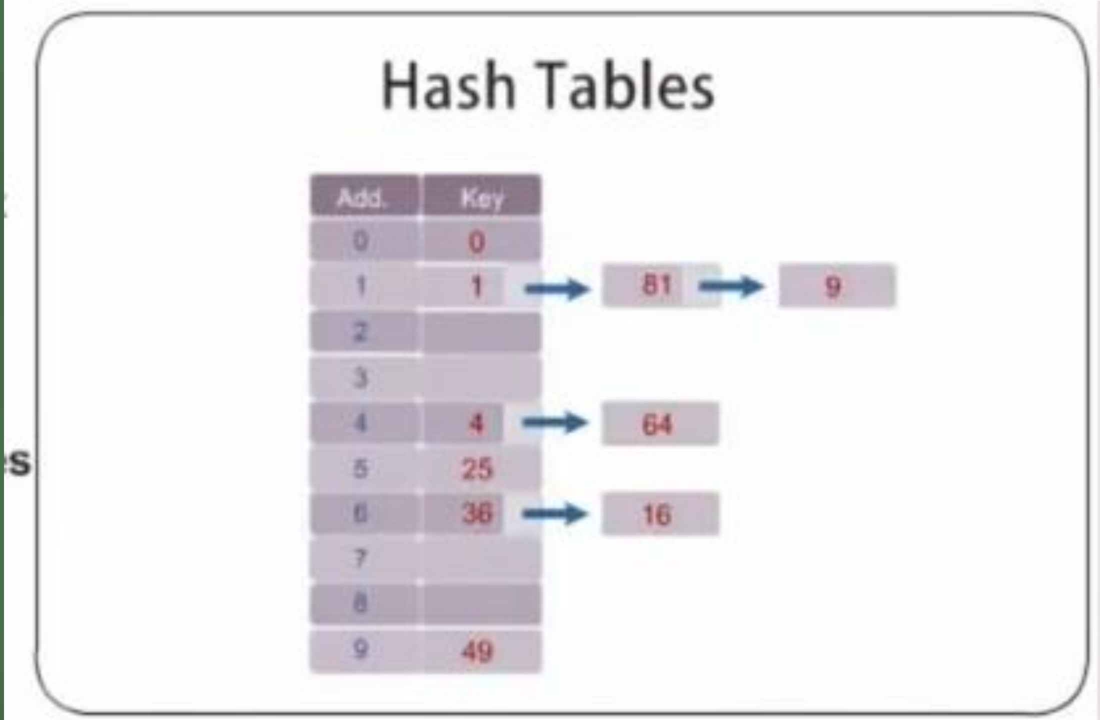
- Circularly

- Hierarchically



Different types of Data Structures

- Arrays
- Stacks
- Queues
- Linked List
- Graphs
- Trees
- Hash Tables



Different types of Data Structures

- Arrays
- Stacks
- Queues
- Linked List
- Graphs
- Trees
- Hash Tables

Algorithms

- Traversal
- Searching
- Sorting

Complexity of Program

Complexity/ Efficiency of Program

- When a program/algorithm is <u>better than</u> other ?

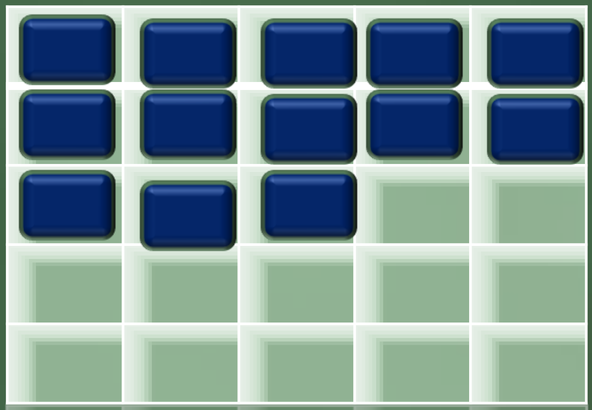
- What are the measures of comparison?

1. Space Required - Space Complexity
2. Time Required – Time Complexity

Space Complexity

Program

RAM



The amount of memory it requires to execute.

Space Complexity

Memory α Parameter

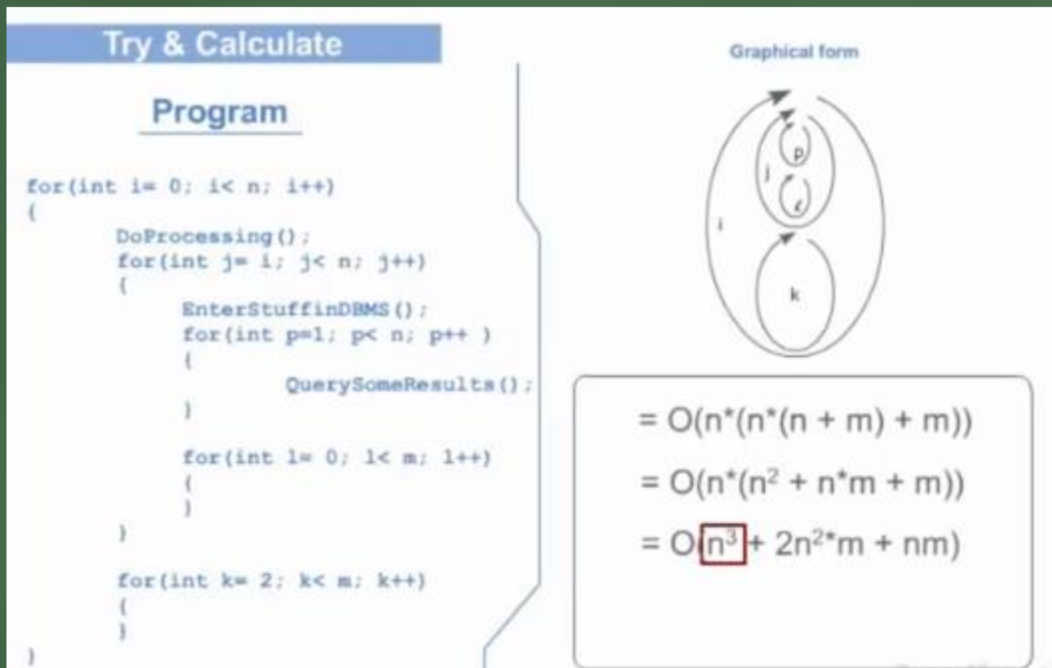
Linear

Square

Cubic

Time Complexity

The amount of computer time it requires to execute the program.



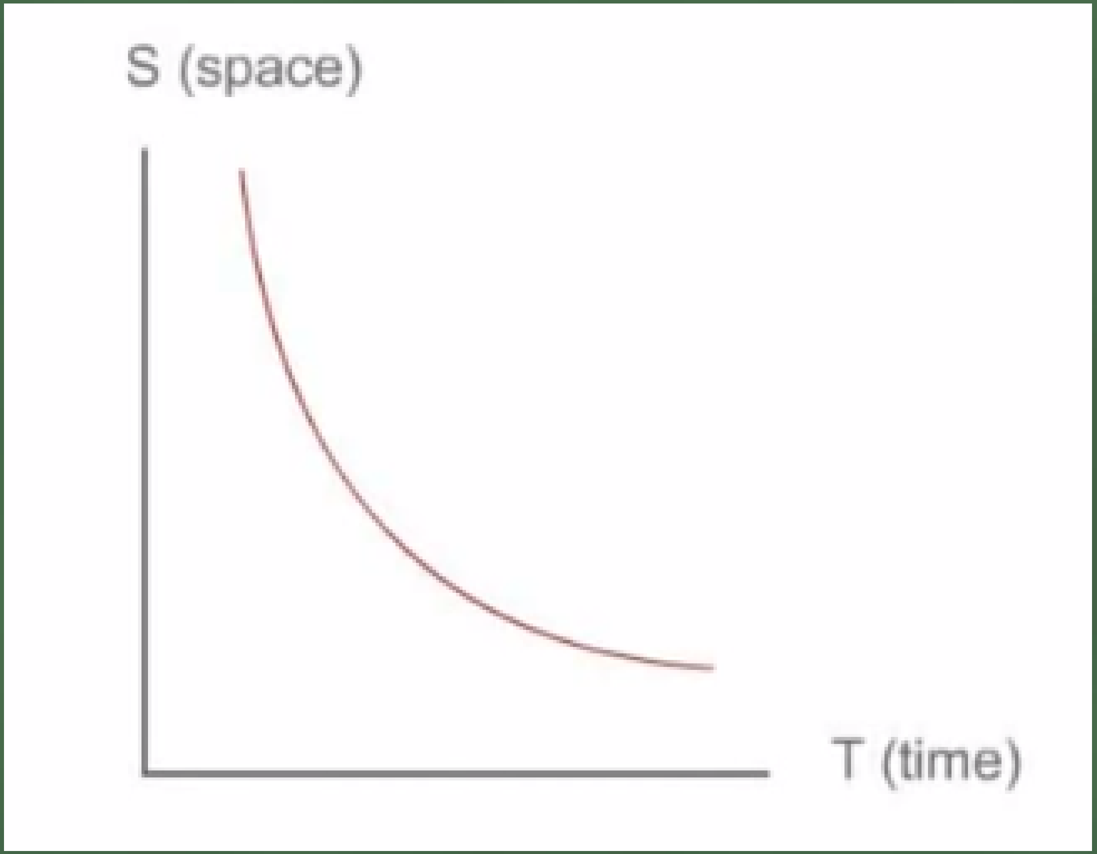
$O(n^3)$

40	50	60	70	80

40	50	60	70	80	90	100	10	20	30

Time Space Trade Off

Less time to execute the program requires more space and vice versa



So there is a trade off between time and space complexity

Complexity Analysis

1. Worst Case analysis

   The maximum time needed over all inputs

2. Best Case analysis

   The minimum time needed

3. Average Case analysis

   average time needed

- Usually only worst-case information is given since average case is much harder to estimate

•Complexity Notations

- ❑ O (The Big O)
- ❑ Ω (Omega)
- ❑ Θ (Theta)

The Big O Notation

- Is used for worst cast analysis

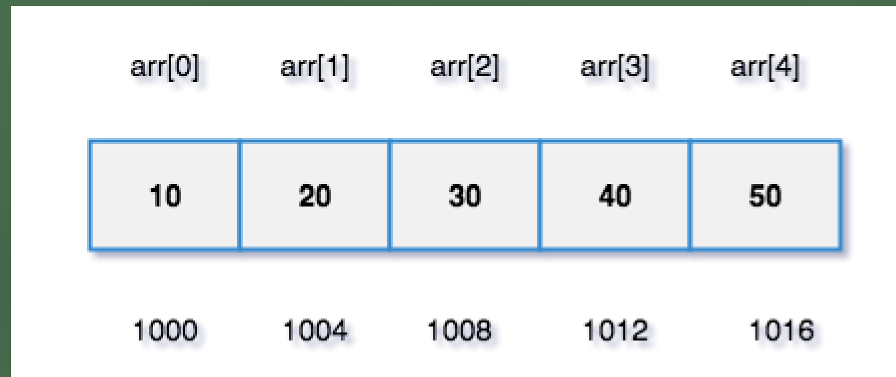
  An algorithm is O(f (N)) if there are constants c and N_0, such that for $N \geq N_0$ the time to perform the algorithm for an input size N is bounded by t(N) < c f(N)

- Consequences
  - O(f(N)) is identically the same as O(a f(N))
  - O(aNx + bNy) is identically the same as O(N max(x,y)
  - O(Nx) implies O(Ny) for all y ≥ x

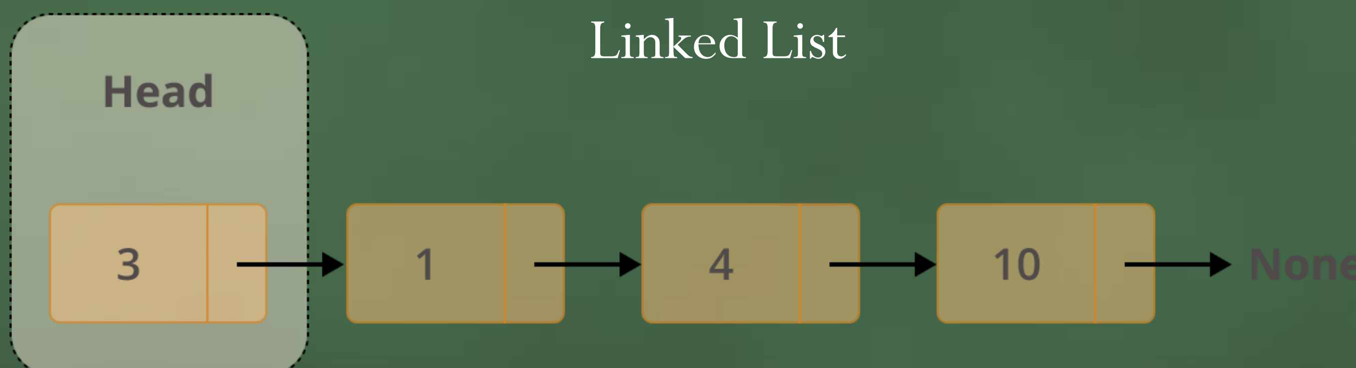
Ω and Θ Notations

- Ω is used for best case analysis:
  - An algorithm is Ω(f (N)) if there are constants c and N0, such that for N≥N0 the time to perform the algorithm for an input size N is bounded by
    t(N) > c f(N)
- Θ is used if worst and best case scale the same
  - An algorithm is Θ(f (N)) if it is Θ(f (N)) and O(f (N))

Array



Linked List



How to analyze Time Complexity

Running time depends upon

- x ➢ Single vs. Multi Processor
- x ➢ Read/Write Speed to Memory
- x ➢ 32 bit vs. 64 bit
- ➢ Input

How to analyze Time Complexity

Running time depends upon

- ~~Single vs Multi Processor~~
- ~~Read/Write Speed to Memory~~
- ~~32 bit vs 64 bit~~
- Input
    - Rate of growth of time

We analyse time complexity for

-Very large input size

- worst case scenario

Algo 1: $T(n) = 5n^2 + 7$

Quardratic rate of growth

Algo 2: $T(n) = 17n^2 + 6n + 8$

Useful rules

- simple statements (read, write, assign)
  - $O(1)$ (constant)
- simple operations (+ - * / == > >= < <=)
  - $O(1)$
- sequence of simple statements/operations
  - rule of sums
- for, do, while loops
  - rules of products

Two important rules

- Rule of sums
  - if you do a number of operations in sequence, the runtime is dominated by the most expensive operation

i.e.

- Rule of products
  - if you repeat an operation a number of times, the total runtime is the runtime of the operation multiplied by the iteration count

```
Function ( )
{  int a;
   a = 5
   a++;
   for(i=0;i<n;i++)
   {
      // simple statements
   }
   for(i=0;i<n; i++)
   {
      for(j=0;j<n;j++)
      {
         // simple statements
      }
   }
}
```

$$T(n) = O(1) + O(n) + O(n2)$$
$$= O(n2)$$

```
Function ( )
{
  if (some Condition)
  {
     for(i=0; i<n; i++)
     {  //simple statements        O(n)
     }
  }
  else
  {  for(i=0; i<n; i++)
     {  for(j=0; j<n; j++)         O(n^2)
        {  // simple statements
        }
     }
  }
}
```

T(n)	keep one	drop coef
$3n^2+4n+1$	$3\,n^2$	n^2
$101\,n^2+102$	$101\,n^2$	n^2
$15\,n^2+6n$	$15\,n^2$	n^2
$a\,n^2+bn+c$	$a\,n^2$	n^2

How to analyze Time Complexity-Example-1

Sum (a,b)

{

return a+b

}

1 1

Tsum = 2

Constant time

•How to analyze Time Complexity-Example-2

```
SumOfList(A,n)
{
   total =0
  for ( i=0, i<n, i++)
            total=total + Aj

   return total

}
```

	Cost	No.OfTimes
total =0	1	1
for (i=0, i<n, i++)	2	n+1
total=total + A_j	2	n
return total	2	1

$$T_{sumOfList} = 1+2(n+1) + 2n + 2$$

$$= 4n + 4$$

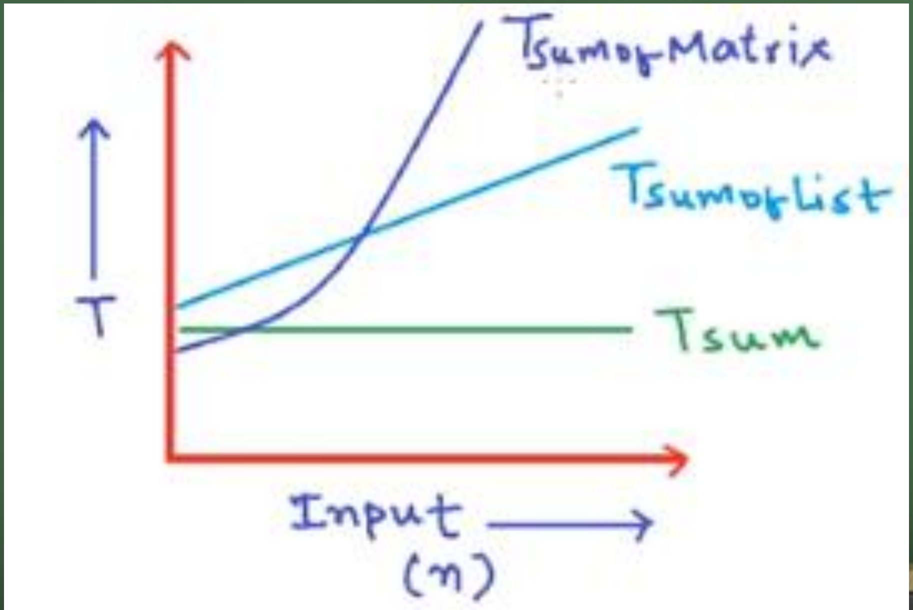
$$\boxed{Cn + C`}$$

- **How to analyze Time Complexity**

$T_{sum} = k$ — $O(1)$

$T_{sumOflist} = Cn+C`$ — $O(n)$

$T_{sumofMatrix} = an^2+bn+c$ — $O(n^2)$

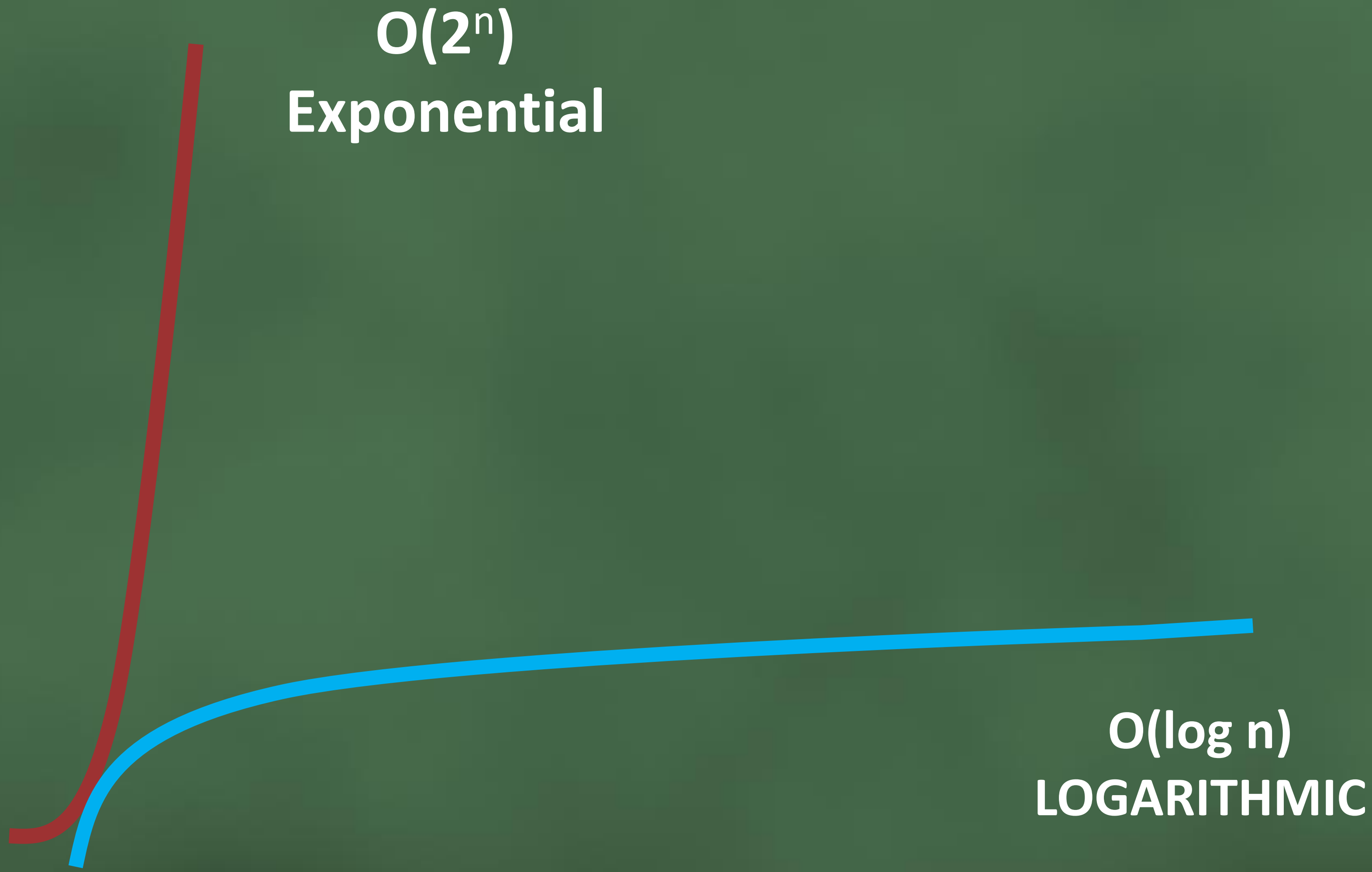


Time Complexity Analysis
– Fibonacci Sequence - comparison

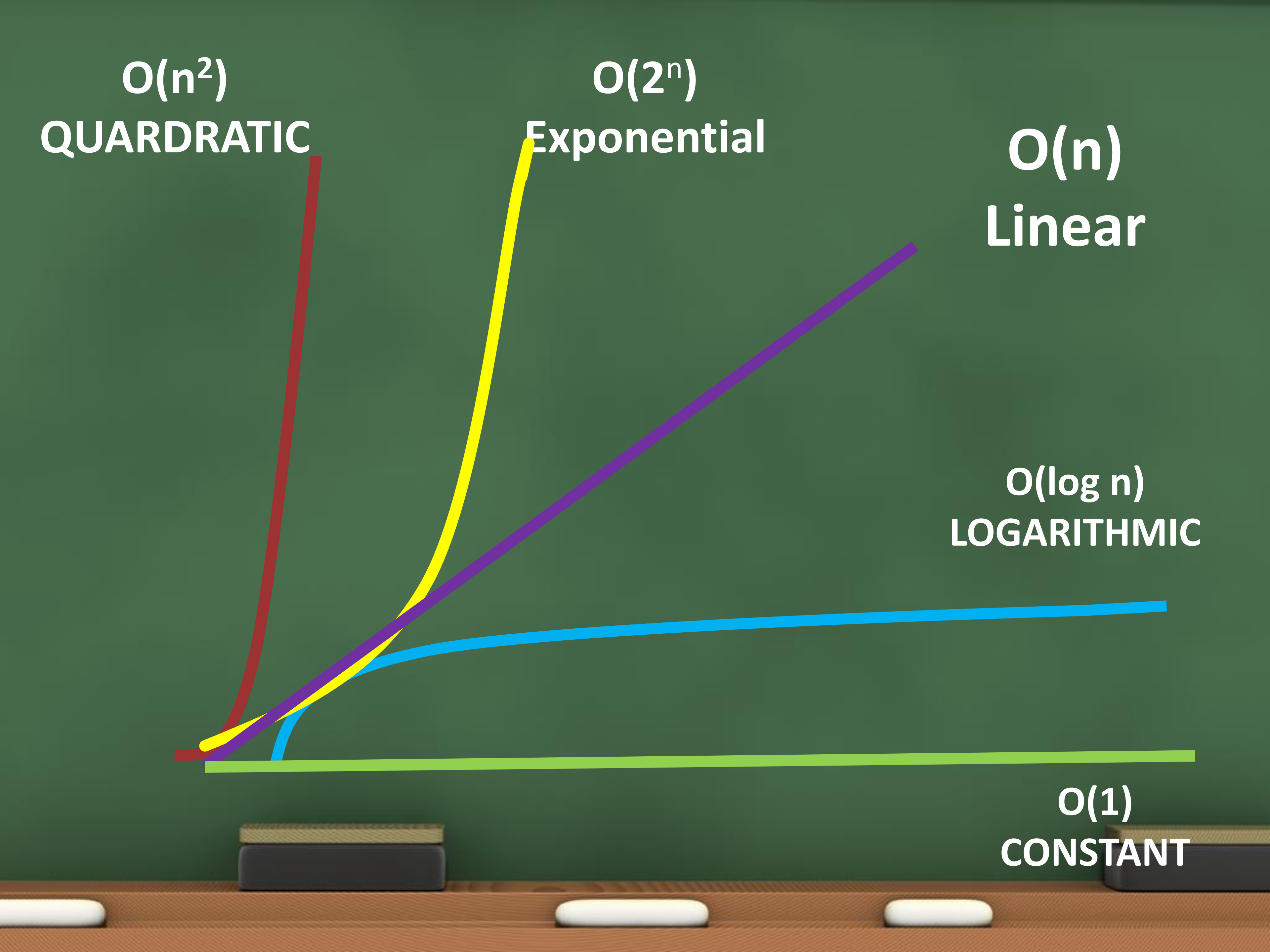
Recursion	$O(2^n)$
Iterative	$O(n)$


$O(n^2)$
QUARDRATIC
$O(n)$
Linear


O(n)
Linear
O(log n)
LOGARITHMIC


O(2^n)
Exponential
O(log n)
LOGARITHMIC

Constant	$O(1)$
Logarithmic	$O(\log n)$
Linear	$O(n)$
Quadratic	$O(n^2)$
Exponential	$O(2^n)$


$O(n^2)$
QUARDRATIC
$O(2^n)$
Exponential
O(n)
Linear
O(log n)
LOGARITHMIC
O(1)
CONSTANT

Which Algorithm do you prefer ?

Algorithm A	Algorithm B	Which do you pick?
O(log N)	O(N)	A
O(log N)	O(Nlog N)	A
O(Nlog N)	O(N)	B
O(log N)	$O(N^2)$	A
O(N!)	$O(2^N)$	B
$O(2^N)$	$O(N^2)$	B

•Complexity – Exercise-1

```
{
        max=0;
        for(i=0; i<n; i++)
        {
                read(number)
                if ( max < number )
                        max=number;
        }
        print(max);
}
```

Hash Tables

Queues

Linked List

Trees

Graphs

Stacks

Array