Algorithm Analysis

Analysis of Algorithms

- An Algorithm is a set of instructions to perform a task or to solve a given problem.
- There are several different algorithms to solve a given problem.
- Analysis of algorithm deals in finding best algorithm which runs fast and takes in less memory.
- For example –

  Q. Find sum of first n natural numbers.

  a) Input :- n = 4

  Output :- 10 i.e. (1 + 2 + 3 + 4)

  b) Input :- n = 5

  Output :- 15 i.e. (1 + 2 + 3 + 4 + 5)

Analysis of Algorithms

Ramesh

```
public int findSum(int n) {
    return n * (n + 1) / 2;
}
```

Suresh

```
public int findSum(int n) {
    int sum = 0;
    for(int i = 1; i <= n; i++) {
        sum = sum + i;
    }
    return sum;
}
```

1. Time Complexity
2. Space Complexity

Time Complexity

Time Complexity

- Its amount of time taken by algorithm to run.
- The input processed by an algorithm helps in determining the time complexity.

Ramesh

```
public int findSum(int n) {
    return n * (n + 1) / 2;
}
```

Suresh

```
public int findSum(int n) {
    int sum = 0;
    for(int i = 1; i <= n; i++) {
        sum = sum + i;
    }
    return sum;
}
```

Space Complexity

Space Complexity

- Its amount of memory or space taken by algorithm to run.
- The memory required to process the input by an algorithm helps in determining the space complexity.

Ramesh

```
public int findSum(int n) {
    return n * (n + 1) / 2;
}
```

Suresh

```
public int findSum(int n) {
    int sum = 0;
    for(int i = 1; i <= n; i++) {
        sum = sum + i;
    }
    return sum;
}
```

Asymptotic Analysis

Asymptotic Analysis

- Asymptotic analysis helps in evaluating performance of an algorithm in terms of input size and its increase.
- Using asymptotic analysis we don't measure actual running time of algorithm.
- It helps in determining how time and space taken by algorithm increases with input size.

Asymptotic Notation

Asymptotic Notations

- Asymptotic Notations are the mathematical tools used to describe the running time of an algorithm in terms of input size.
- Example – Performance of car in 1 litre of petrol

Highway (min traffic) – 25 km/litre
City (max traffic) – 15 km/litre
City + Highway (avg traffic)– 20 km/litre

Asymptotic Notations help us in determining –
1. Best Case
2. Average Case
3. Worst Case

Order of Growth: how an algorithm's time taken grows as input size increases. It can be constant, linear, quadratic etc

Asymptotic Notation Types

Best - Exact or lower

Omega(Ω) Notation

- It is the formal way to express the lower bound of an algorithm's running time.
- Lower bound means for any given input this notation determines best amount of time an algorithm can take to complete.

- For example –

  If we say certain algorithm takes 100 secs as best amount of time. So, 100 secs will be lower bound of that algorithm. The algorithm can take more than 100 secs but it will not take less than 100 secs.

Worst - Exact or upper

Big O(O) Notation

- It is the formal way to express the upper bound of an algorithm's running time.
- Upper bound means for any given input this notation determines longest amount of time an algorithm can take to complete.

- For example –

  If we say certain algorithm takes 100 secs as longest amount of time. So, 100 secs will be upper bound of that algorithm. The algorithm can take less than 100 secs but it will not take more than 100 secs.

Average - Exact

Theta(Θ) Notation

- It is the formal way to express both the upper and lower bound of an algorithm's running time.
- By Lower and Upper bound means for any given input this notation determines average amount of time an algorithm can take to complete.

- For example –

  If we run certain algorithm and it takes 100 secs for first run, 120 secs for second run, 110 for third run and so on. So, theta notations gives an average of running time of that algorithm.

Big O notation rules

Rules of Big O(O) Notation

- It's a Single Processor
- It performs Sequential Execution of statements
- Assignment operation takes 1 unit of time
- Return statement takes in 1 unit of time
- Arithmetical operation takes 1 unit of time
- Logical operation takes 1 unit of time
- Other small/single operations takes 1 unit of time
- Drop lower order terms
- Drop constant multipliers

$$T = 3n^2 + 6n + 1 \Longrightarrow O(n^2)$$

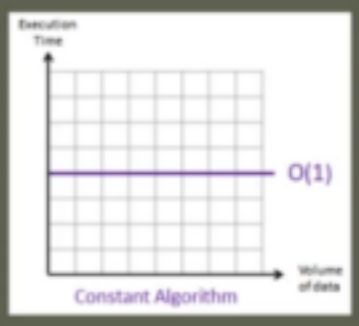
Time Complexity of constant algorithm

```
1  public int sum(int x, int y) {
2      int result = x + y;
3      return result;
4  }
```

line no.	operations	unit time
2	1 + 1 + 1 + 1	4
3	1 + 1	2

T = 4 + 2 = 6

T ≈ C (constant)




```
public int get(int[ ] arr, int i) {
    return arr[i];
}
```

Time Complexity = O(1)

Credit: 101computing.net

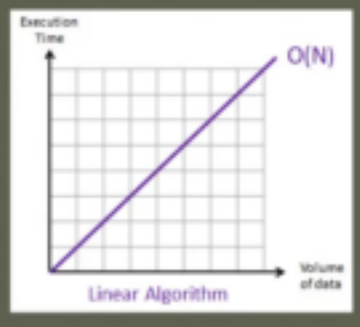
Time Complexity of linear algorithm

```
1  public void findSum(int n) {
2      int sum = 0;  // 1 step
3      for(int i = 1; i <= n; i++) {
4          sum = sum + i; // n steps
5      }
6      return sum; // 1 step
7  }
```

line no.	operations	unit time
2	1	1
3	1 + 3n + 3 + 3n	6n + 4
4	n(1 + 1 + 1 + 1)	4n
6	1 + 1	2

T = 1 + 6n + 4 + 4n + 2

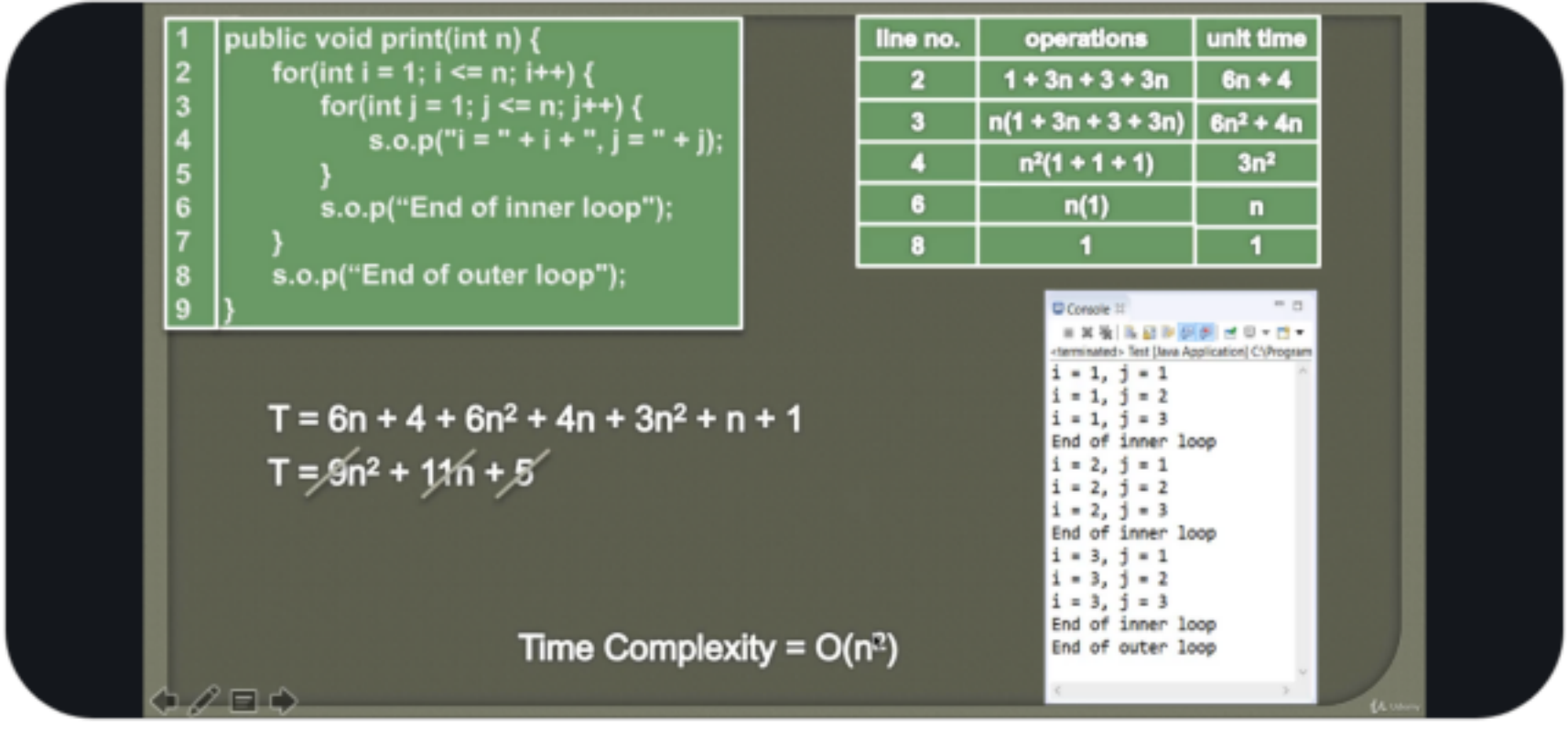
T = ~~10n + 7~~



Time Complexity = O(n)

Credit: 101computing.net

Time Complexity of polynomial algo



Auxiliary space

Order of Growth of extra space or temporary space in terms of input size

Order of Growth hierarchy:

Constant < log(logn)) < logn < n < nlogn < (n)2 < (n)3 < (n)4 < (2)n < (n)n

- If loops goes like i*2 or i/2 then time complexity will be O(logn)
- If loops goes like i^2 then time complexity will be O(log(logn))

==========

Data Structure

Tries

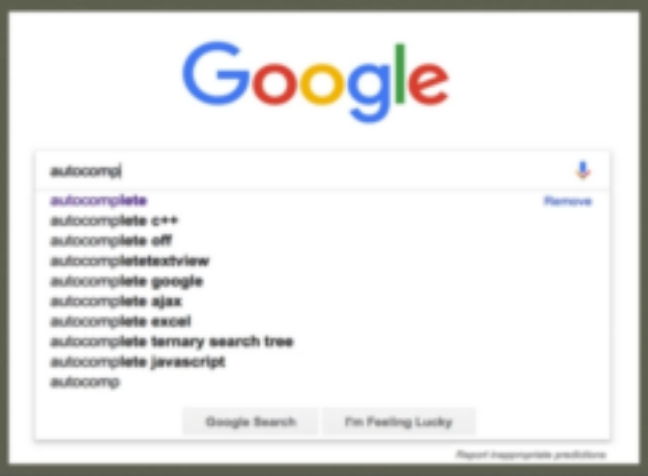
What is a Trie?

- Trie basically comes from the word Re*trie*val.
- The main purpose of this data structure is to retrieve stored information very fast.

- A Trie with 4 words -
  1. dog
  2. dust
  3. hat
  4. home

Applications - Auto-Complete words

- Autocomplete feature is implemented by Tries.
- Many websites have used autocomplete feature, which suggest user rest of the word, while user is typing.

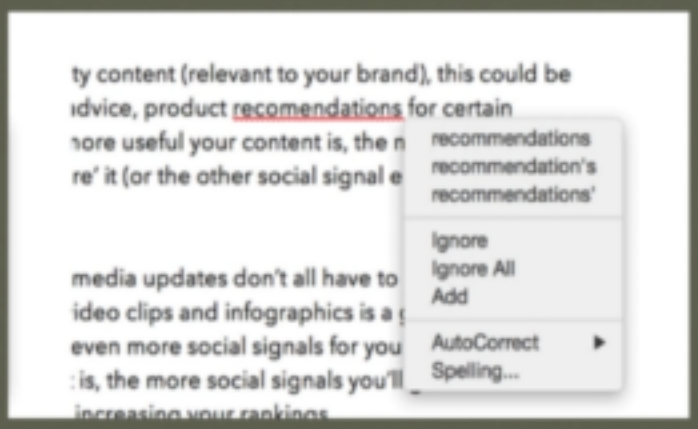


Applications - Search Contacts in phone

- Searching a person contact number in contact list is efficiently implemented by Trie. As soon as user enters letters the application auto suggest the name of the person.

Applications - Spell Checking

- Tries help to check and correct word spelling entered by user. In case user doesn't know exact spelling it auto suggest the correct spelling.

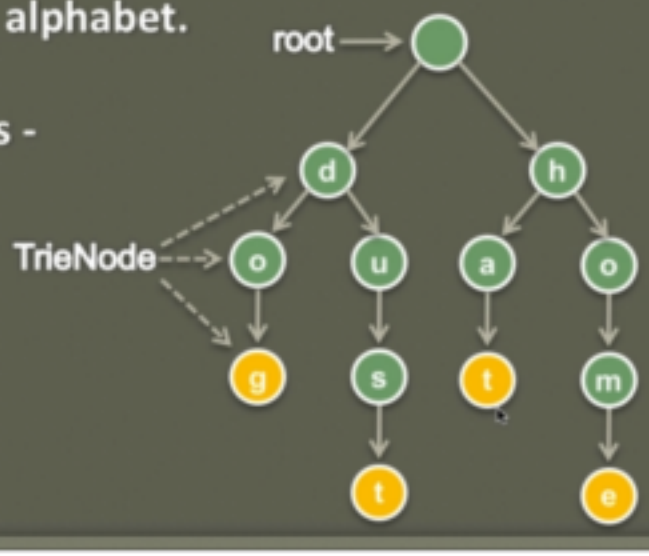


Trie Node

What is a TrieNode?

- A TrieNode in a Trie represents a single alphabet of the word.
- In below example, in order to insert word "dog" 3 TrieNode are used, one for each alphabet.

- A Trie with 4 words -
  1. dog
  2. dust
  3. hat
  4. home



Representation of a Trie Node

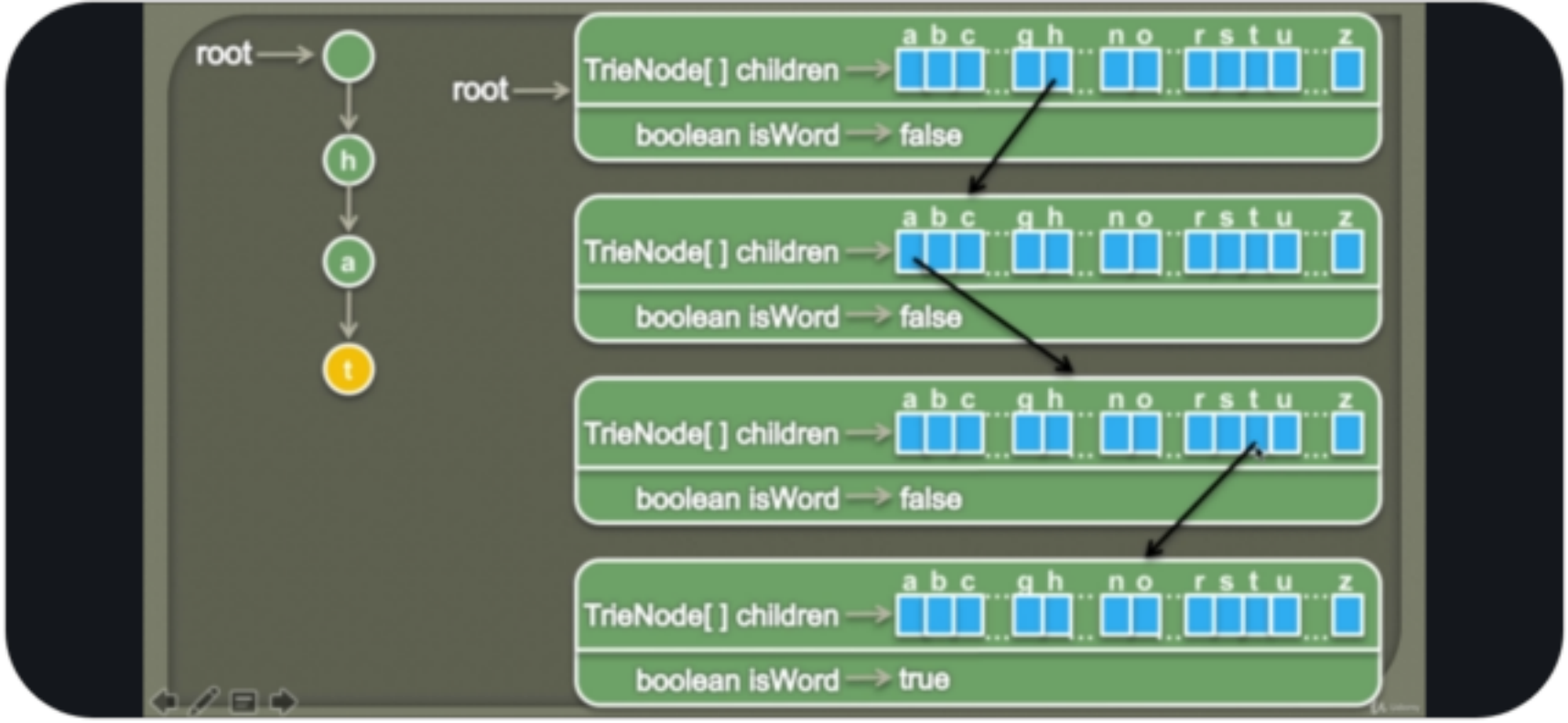
Representation of a TrieNode in Trie

A TrieNode class in Trie consists of two data members.

1. TrieNode[ ] children - An array which refers to other TrieNodes in Trie, also called as child nodes of a TrieNode. The size of array is usually taken as 26 (if we are storing English words)
2. boolean isWord - A boolean value to indicate the end of word. This value is set as true when a word is inserted completely.

TrieNode

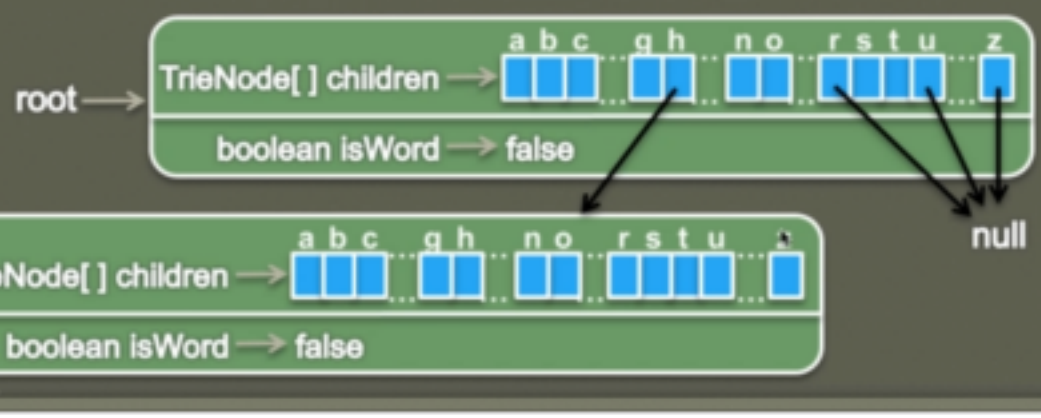
```
TrieNode[ ] children
boolean isWord
```

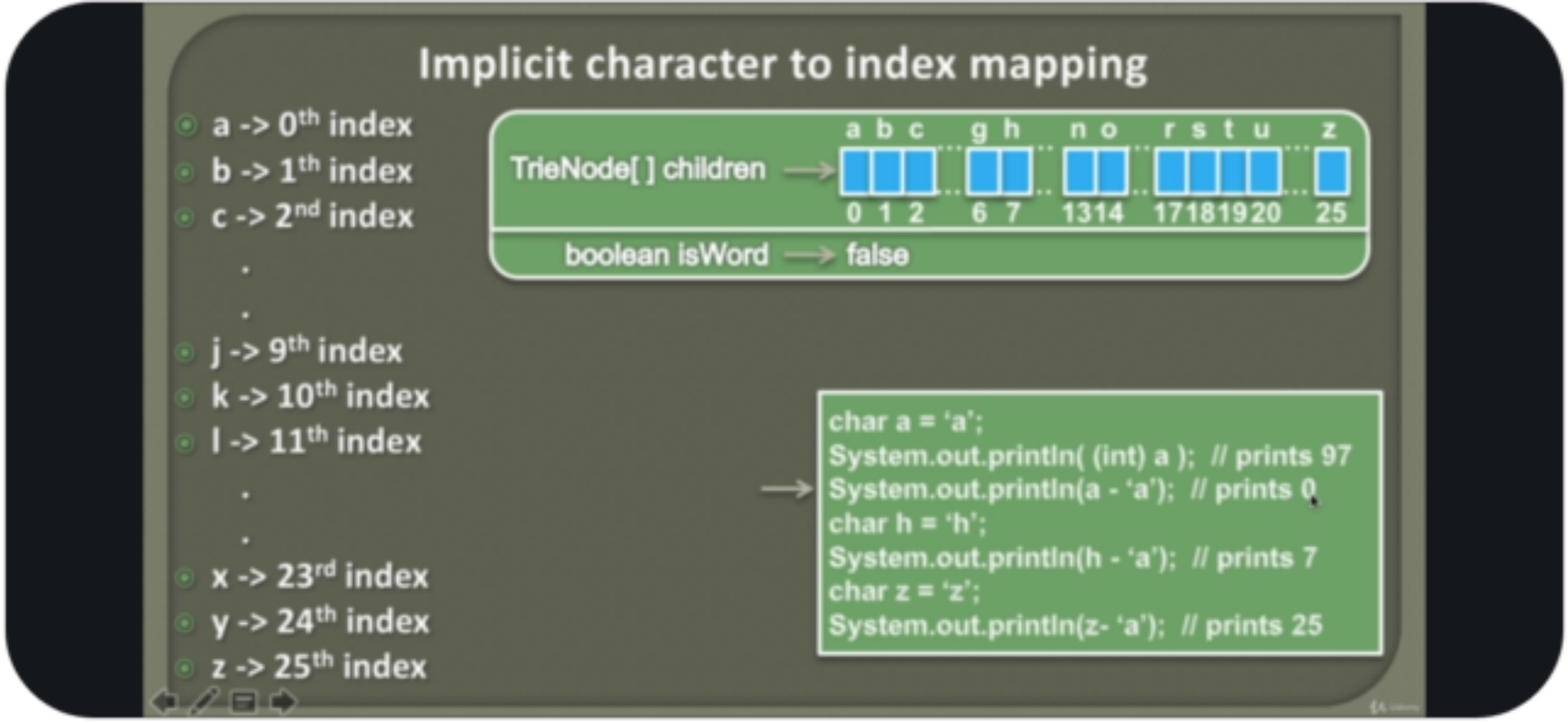


Implementation of Trie Node

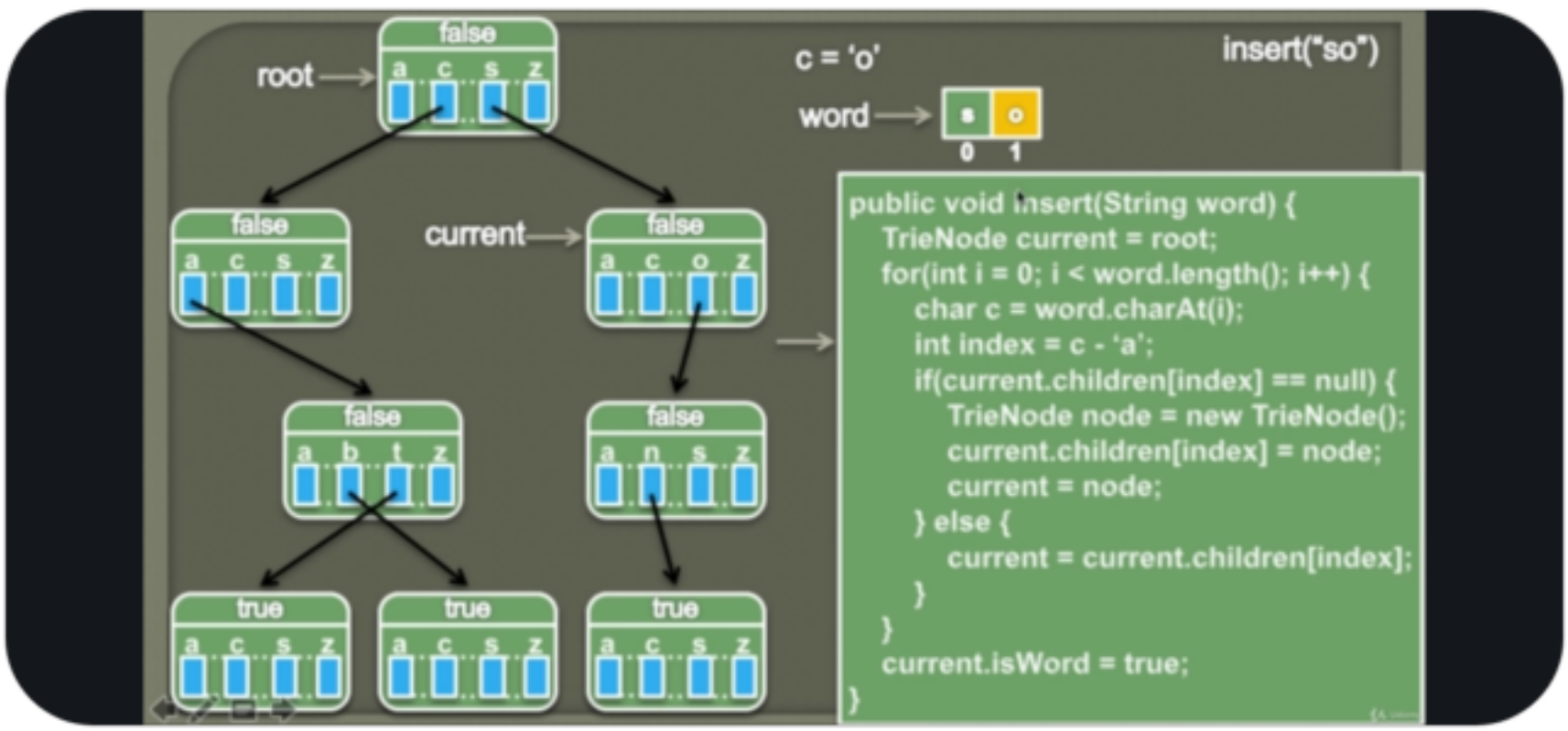
Implementation of the Trie class

- A Trie will be implemented using TrieNode class.
- A root TrieNode is at top with empty value having 26 links (one per alphabet).
- The links are either null or points to another TrieNode.



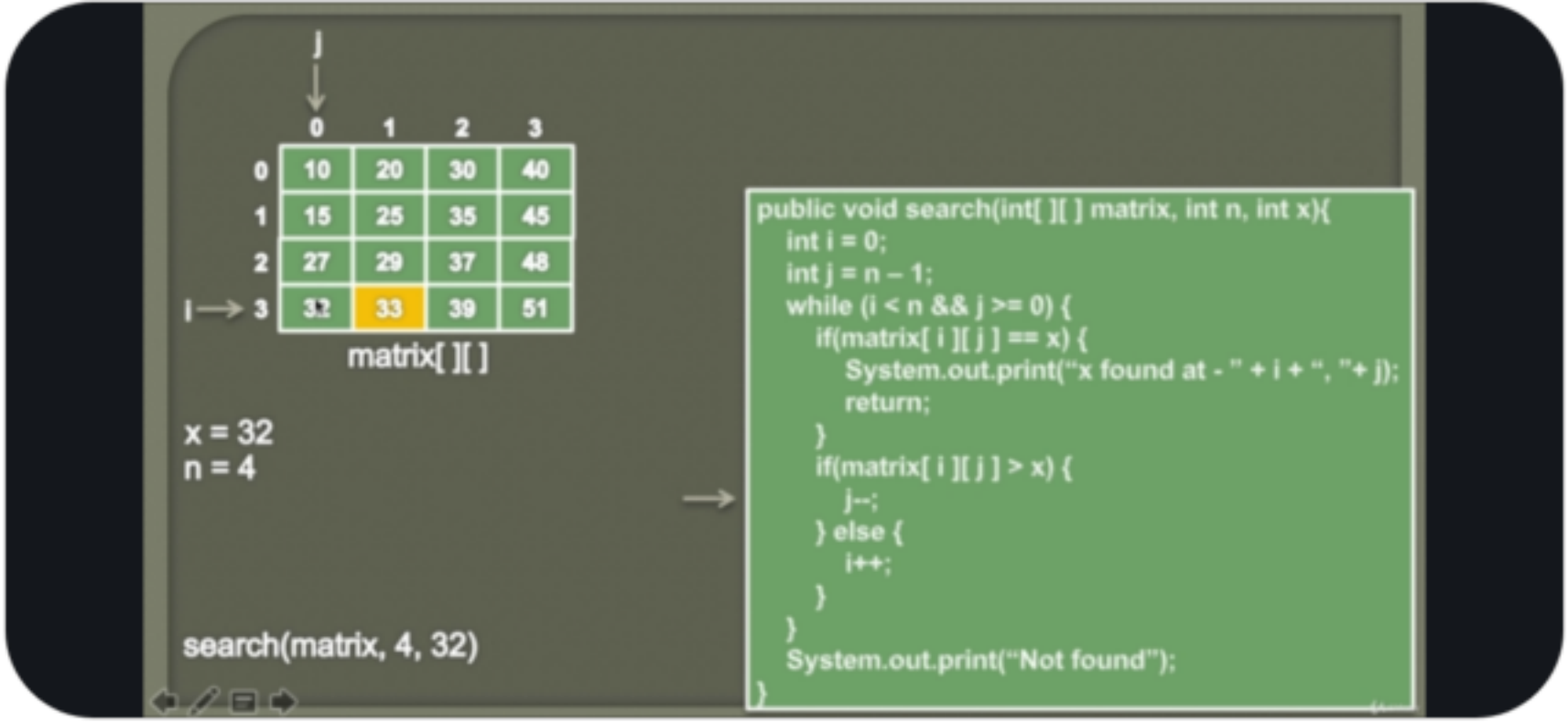


How to insert a word



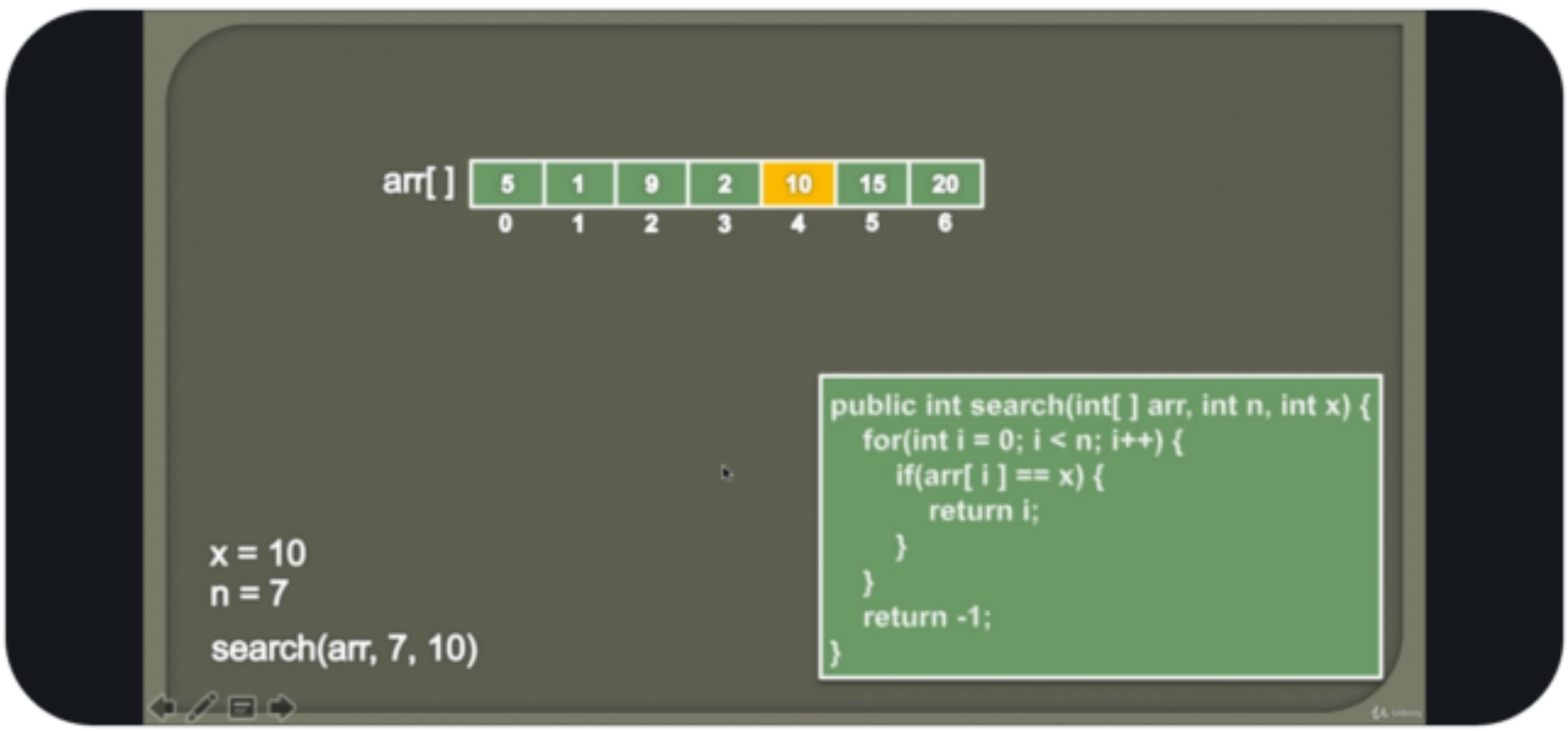
Matrix

Search for a key in sorted matrix

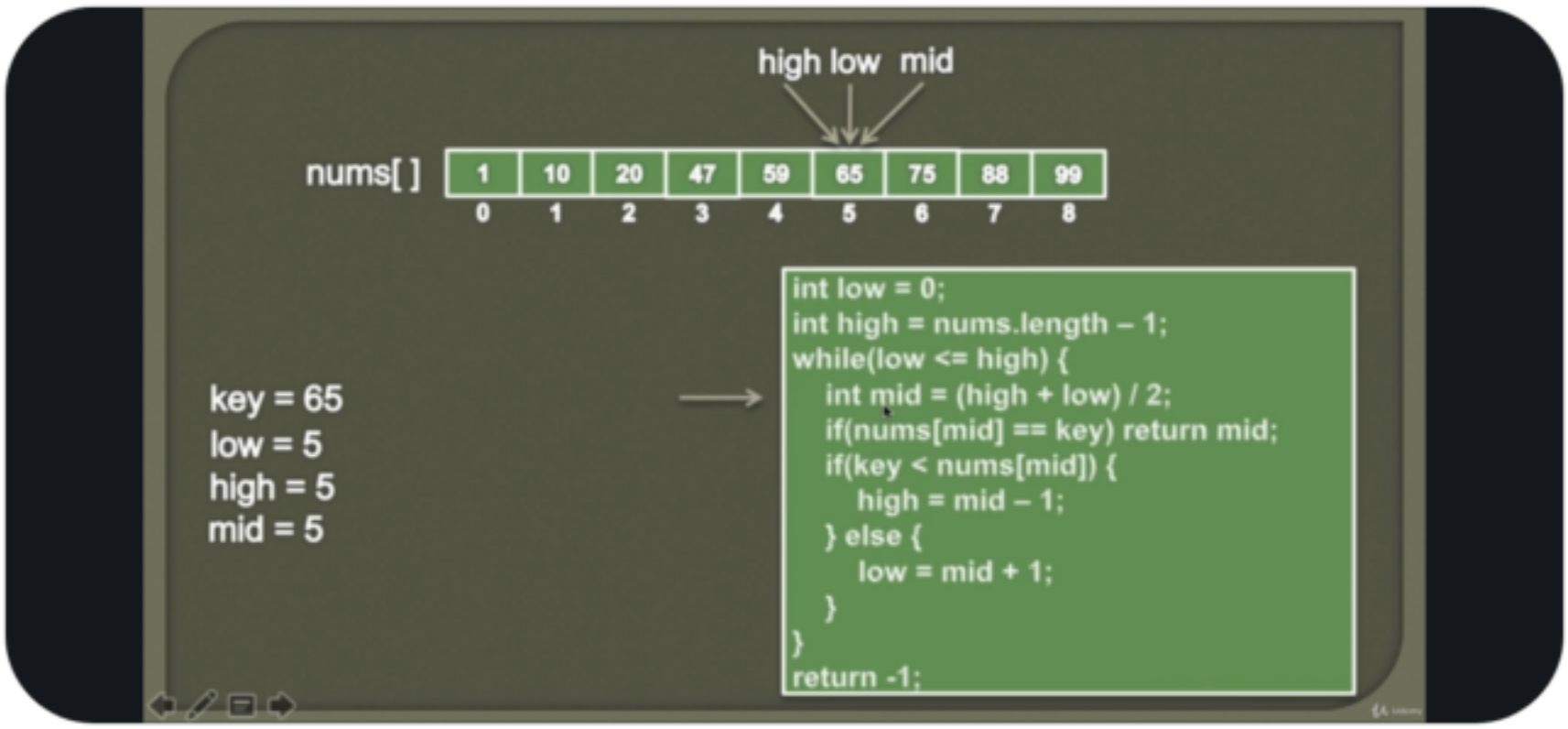


Searching

Linear Search



Binary Search



Sorting

Bubble Sort

Insertion Sort