K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

Course: Data Structure

(Course Code : ENCS205)

UNIT-1: Foundations of Data Structures

School of Engineering & Technology
K.R. Mangalam University

SESSION 6:
Recurrence Relation

Recapitulation (Previous Session)

Quiz

Q1: Which complexity class indicates algorithms with quadratic time complexity?
a) O(n)
b) O(log n)
c) O(n^2)
d) O(1)

Q2: What does it mean when an algorithm has exponential time complexity?
a) It runs in constant time.
b) Its running time increases linearly with the size of the input.
c) Its running time doubles with each increase in input size.
d) Its running time grows rapidly with the size of the input.

Recapitulation (Previous Session)

Quiz

Q3: Which complexity class represents algorithms that are considered intractable and are not feasible to solve efficiently for large input sizes?

a) O(log n)

b) O(n)

c) O(n log n)

d) O(2^n)

Q4: What does it mean when an algorithm has logarithmic space complexity?

a) Its memory usage grows logarithmically with the size of the input.

b) Its memory usage remains constant regardless of input size.

c) Its memory usage grows exponentially with the size of the input.

d) Its memory usage decreases with the size of the input.

Recapitulation (Previous Session)

Quiz (Answers)

A1: c) O(n^2)

A2: d) Its running time grows rapidly with the size of the input.

A3: d) O(2^n)

A4: a) Its memory usage grows logarithmically with the size of the input.

Recurrence Relation

- Mathematical expression that defines a sequence previous terms.
- Used to model the time complexity.
- General form : a_{n} = f(a_{n-1}, a_{n-2},....,a_{n-k})
- f defines relationship between current term and previous terms.

Recurrence Relation (Cont..)

- Equation or inequality that describes a function values on smaller inputs.

- To solve obtain a function defined on the natural numbers that satisfy the recurrence.

- Recurrence relation is represented as:

$$T(n) = \theta(1) \text{ if } n=1$$
$$2T(n/2) + \theta(n) \text{ if } n>1$$

Recurrence Relation (Cont..)

- Equation or inequality that describes a function values on smaller inputs.
- To solve obtain a function defined on the natural numbers that satisfy the recurrence.
- Recurrence relation is represented as:

$$T(n) = \theta(1) \text{ if } n=1$$
$$2T(n/2)+ \theta(n) \text{ if } n>1$$

Significance of Recurrence Relations in Data Structure

- In analyzing and optimizing the complexity of algorithms.

- Some of the common uses :

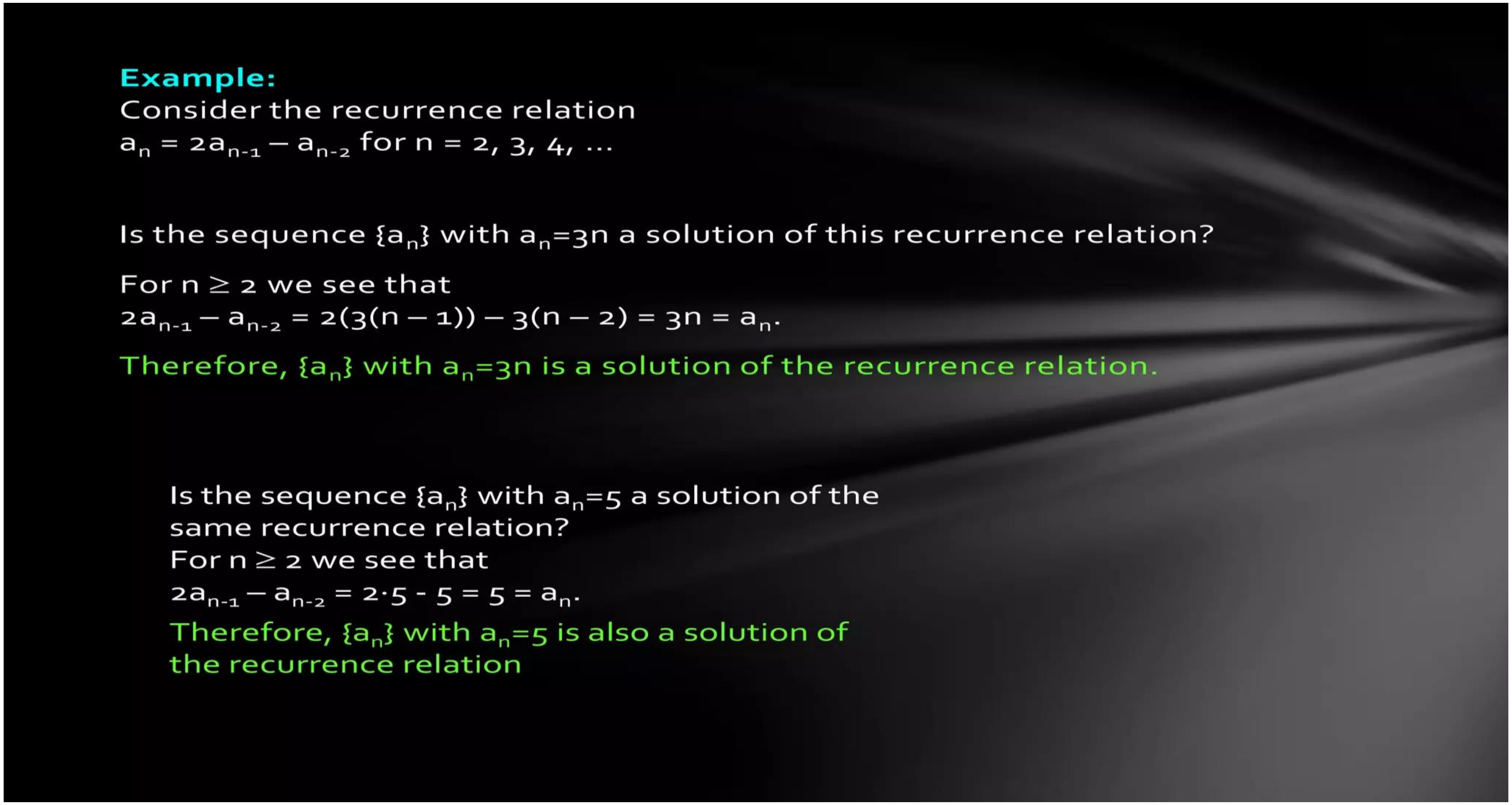
  a. Time Complexity Analysis

  b. Generalizing Divide and Conquer Algorithms

  c. Analyzing Recursive Algorithms

  d. Defining State and Transitions for Dynamic Programming.

Example of Recurrence Relations



Example:
Consider the recurrence relation
$a_n = 2a_{n-1} - a_{n-2}$ for $n = 2, 3, 4, \ldots$

Is the sequence $\{a_n\}$ with $a_n=3n$ a solution of this recurrence relation?

For $n \geq 2$ we see that
$2a_{n-1} - a_{n-2} = 2(3(n-1)) - 3(n-2) = 3n = a_n$.

Therefore, $\{a_n\}$ with $a_n=3n$ is a solution of the recurrence relation.

Is the sequence $\{a_n\}$ with $a_n=5$ a solution of the same recurrence relation?
For $n \geq 2$ we see that
$2a_{n-1} - a_{n-2} = 2\cdot5 - 5 = 5 = a_n$.
Therefore, $\{a_n\}$ with $a_n=5$ is also a solution of the recurrence relation

Some more examples of Recurrence Relations

Example	Recurrence Relation
Fibonacci Sequence	F(n) = F(n-1) + F(n-2)
Factorial of a number n	F(n) = n * F(n-1)
Merge Sort	T(n) = 2*T(n/2) + O(n)
Tower of Hanoi	H(n) = 2*H(n-1) + 1

Different methods to solve Recurrence Relations

- By Substitution Method
- By Iteration Method
- By Recurrence Tree
- By Master Method

Different methods to solve Recurrence Relations (Cont..)

1. Substitution Method

- It follows three steps:

  Step 1: Guess the form of the solution.

  Step 2: Verify by induction.

  Step 3: Solve for constants

Different methods to solve Recurrence Relations (Cont..)

Example of Substitution Method

$T(n)$=4T(2n)+n

Soln: We'll now prove the general case.

$$T(n)=4T(\mathrm{n}/2)+n$$

$$\leq 4c(n/2)^3+n$$

$$(c/2)n^3+n$$

$$cn^3-((c/2)n^3-n)$$

$$\leq cn^3$$

K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

Different methods to solve Recurrence Relations (Cont..)

Explanation :

T*(*n*)=4*T*(2*n*)+*n

Soln: We will assume that $T(1)=\Theta(1)$T(1)=Θ(1).

1. Let's guess that $T(n)\in O(n3)$T(n)∈O(n3).
   We will assume that $T(k)\leq ck3$T(k)≤ck3 for $k<n$k<n.

2. Now we need to prove that $T(n)\leq cn3$T(n)≤cn3 by induction.

3. The base case is $T(n)=\Theta(1)$T(n)=Θ(1) for all $n<n0$n<n0
   where $n0$n0 is a suitable constant.

Different methods to solve Recurrence Relations (Cont..)

Explanation :

$T(n)=4T(2n)+n$

4.Our desired form was $cn3$cn3, so we tried to put the equation in the form desired – residual: $cn3-((c/2)n3-n)$cn3−((c/2)n3−n). This is certainly less then our desired form cn^3cn^3 whenever $(c/2)n^3-n\geq(c/2)n^3-n\geq1$ (*residual*), for example, if $c\geq2c\geq2$ and $n\geq1n\geq1$.

5. For $1\leq n<n01\leq n<n0$, we have $T(n)\leq cn3T(n)\leq cn3$ if we pick cc large enough.
$T(n)\in O(n3)$T(n)∈O(n3).

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method

- Tree is a graphical representation that illustrates execution flow recursive function.
- Visual breakdown of recursive calls
- Showcasing the progression of the algorithm
- Helps analyzing time complexity
- Understanding recursive process .

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

- Node represents a particular recursive call.
- Initial call at the top,
- Subsequent calls branching.
- Tree grows downward, forming a hierarchical structure.
- The branching factor of each node depends on the no. of recursive calls made within the function.
- Additionally, the depth of the tree corresponds to the no. of recursive calls before reaching the base case.

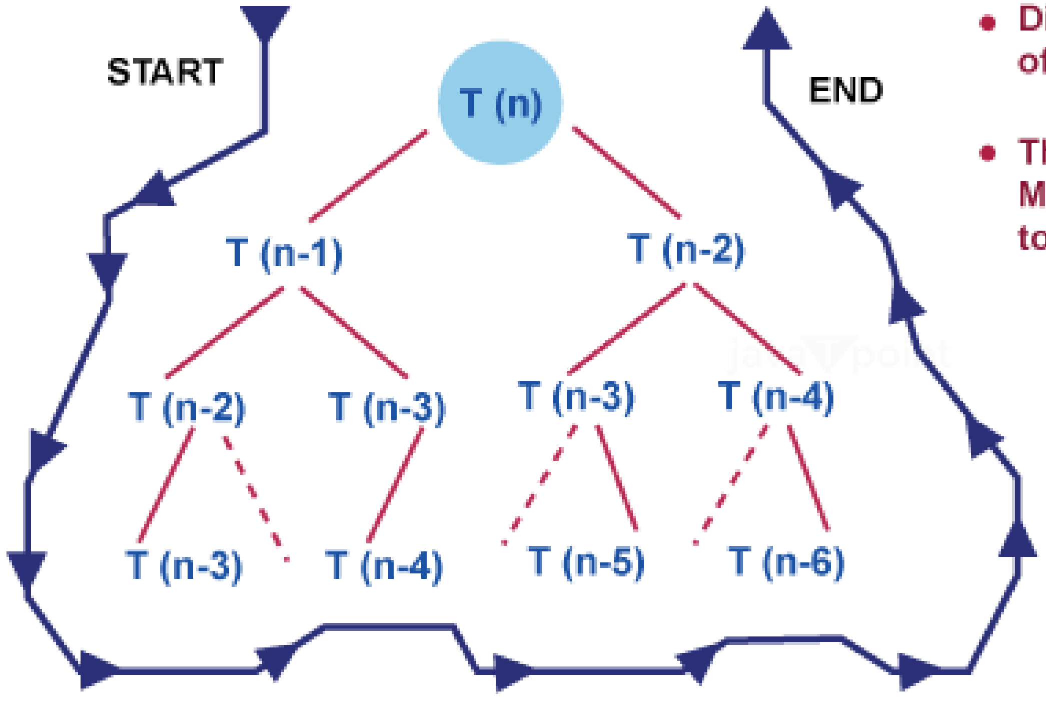
Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

- **Base Case:**
- The base case serves as the termination condition for a recursive function.
- It defines the point at which the recursion stops and the function starts returning values.
- In a recursion tree, the nodes representing the base case are usually depicted as leaf nodes, as they do not result in further recursive calls.

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

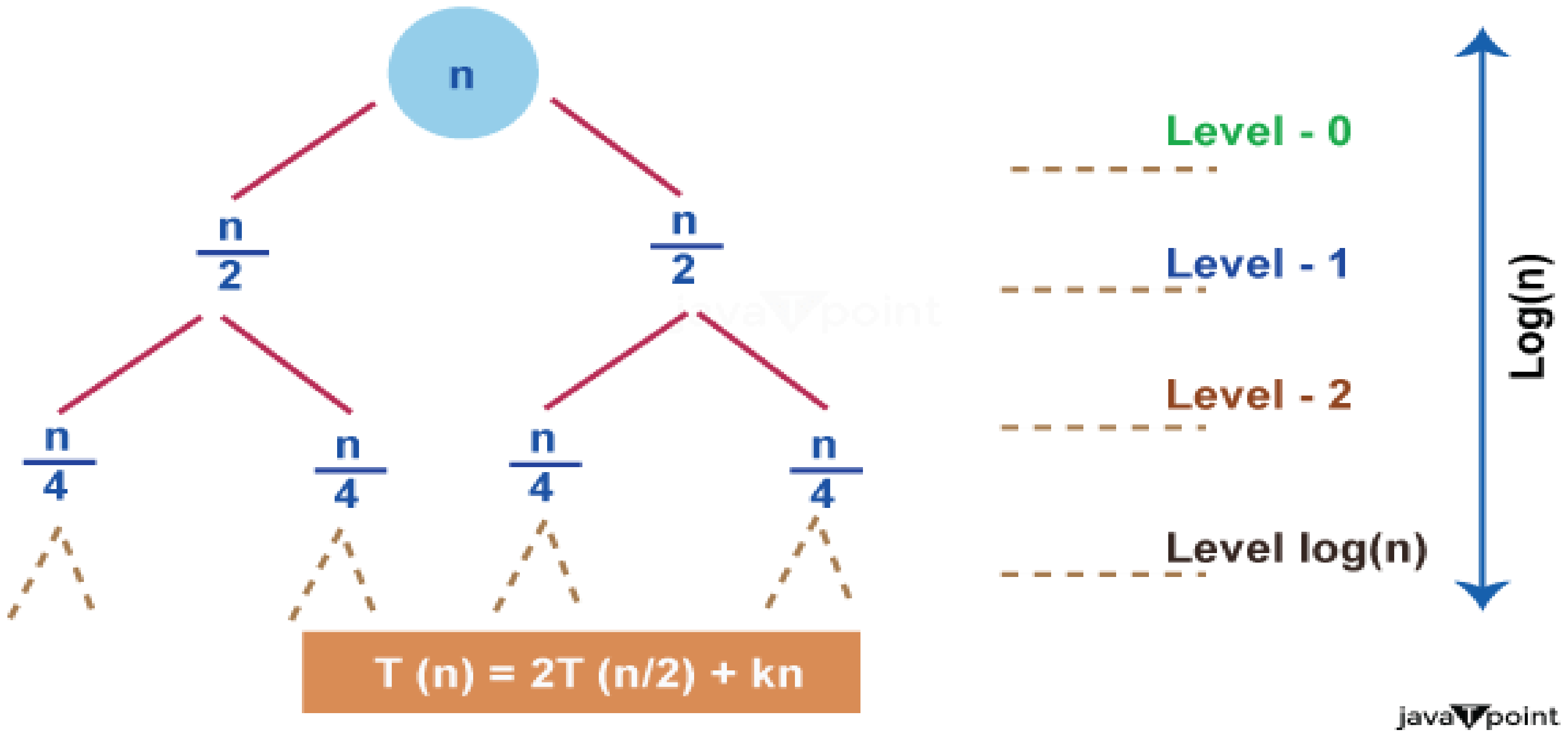


- Direction of Arrows is the Order of Execution of Function Calls.
- They execute in Depth First Manner Top to Bottom, Left to Right.

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

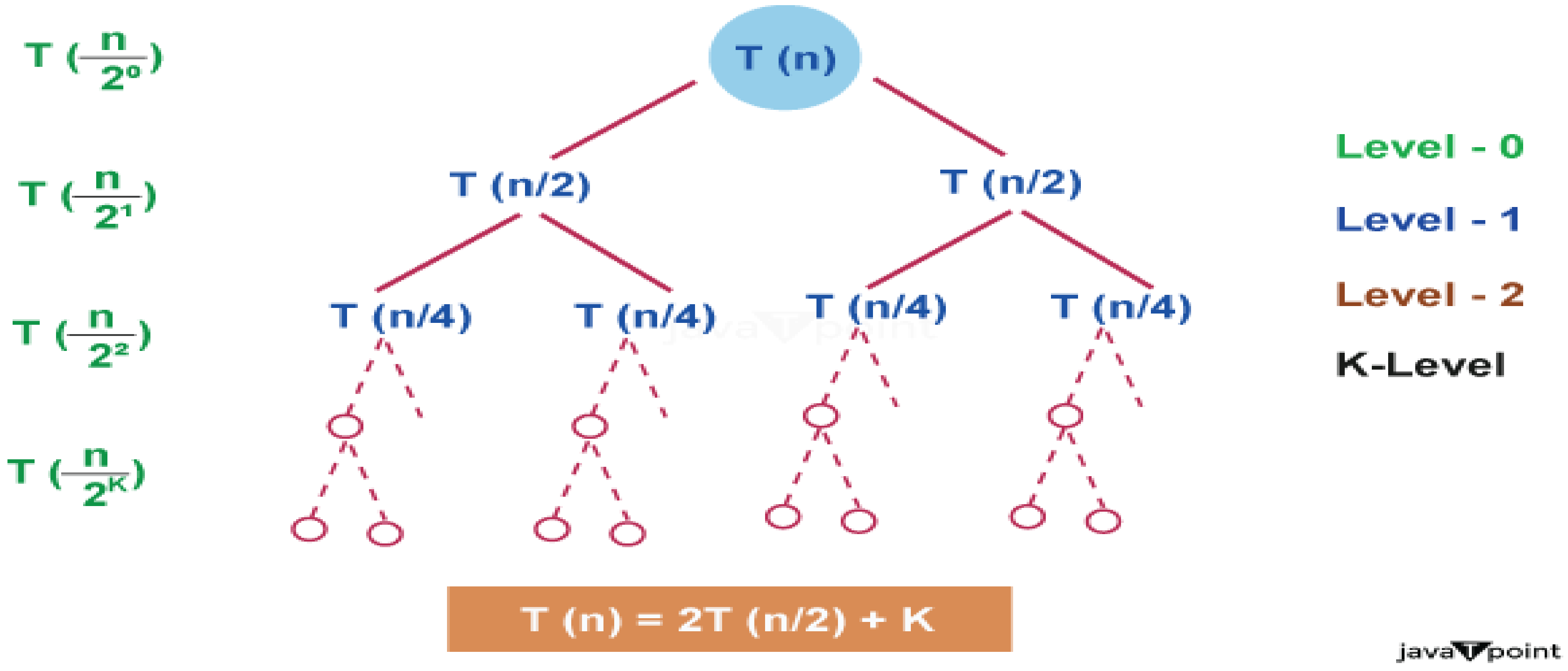
Ques : T(n) = 2T(n/2) + K



Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

Step 1: Draw the Recursion Tree



Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

Step 2: Calculate the Height of the Tree

1. Since we know that when we continuously divide a number by 2, there comes a time when this number is reduced to 1.
2. Same as with the problem size N, suppose after K divisions by 2, N becomes equal to 1, which implies, (n / 2^k) = 1
3. Here n / 2^k is the problem size at the last level and it is always equal. to 1.

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

Step 2: Calculate the Height of the Tree

1. Since we know that when we continuously divide a number by 2, there comes a time when this number is reduced to 1.
2. Same as with the problem size N, suppose after K divisions by 2, N becomes equal to 1, which implies, (n / 2^k) = 1
3. Here n / 2^k is the problem size at the last level and it is always equalto1.

log(n) = log(2^k)

log(n) = k * log(2)

•k = log(n) / log(2)

•k = log(n) base 2

So the height of the tree is log (n) base 2.

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

Step 3: Calculate the cost at each level

- Cost at Level-0 = K, two sub-problems are merged.
- Cost at Level-1 = K + K = 2*K, two sub-problems are merged two times.
- Cost at Level-2 = K + K + K + K = 4*K, two sub-problems are merged four times. and so on....

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

Step 4: Calculate the number of nodes at each level

Let's first determine the number of nodes in the last level. From the recursion tree, we can deduce this

- Level-0 have 1 (2^0) node
- Level-1 have 2 (2^1) nodes
- Level-2 have 4 (2^2) nodes
- Level-3 have 8 (2^3) nodes
- So the level log(n) should have 2^(log(n)) nodes i.e. n nodes.

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

Step 5: Sum up the cost of all the levels

- The total cost can be written as,
- Total Cost = Cost of all levels except last level + Cost of last level
- Total Cost = Cost for level-0 + Cost for level-1 + Cost for level-2 +.... + Cost for level-log(n) + Cost for last level

Different methods to solve Recurrence Relations (Cont..)

2. Recursion Tree Method (Cont..)

Step 5: Sum up the cost of all the levels

- The total cost can be written as,
- Total Cost = Cost of all levels except last level + Cost of last level
- Total Cost = Cost for level-0 + Cost for level-1 + Cost for level-2 +.... + Cost for level-log(n) + Cost for last level

Let's put the values into the formulae,

- T(n) = K + 2*K + 4*K + ... + log(n)` times + `O(1) * n
- T(n) = K(1 + 2 + 4 + ... + log(n) times)` + `O(n)
- T(n) = K(2^0 + 2^1 + 2^2 + ...+ log(n) times + O(n)

Different methods to solve Recurrence Relations (Cont..)

3. Iteration Methods

It means to expand the recurrence and express it as a summation of terms of n and initial condition.

Ex: $T(n) = 1$ if $n=1$

$= 2T(n-1)$ if $n>1$

Different methods to solve Recurrence Relations (Cont..)

3. Iteration Methods (Cont..)

Example: T (n) = 2T (n-1)

= 2[2T (n-2)] = 22T (n-2)

= 4[2T (n-3)] = 23T (n-3)

= 8[2T (n-4)] = 24T (n-4) (Eq.1)

Repeat the procedure for i times

T (n) = 2i T (n-i)

Put n-i=1 or i= n-1 in (Eq.1)

T (n) = 2n-1 T (1)

= 2n-1 .1 {T (1) =1 .....given}

= 2n-1

K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

Different methods to solve Recurrence Relations (Cont..)

4. Master Method

T (n) = a T(n/b) +f(n) with $a \geq 1$ and $b \geq 1$ be constant &
f(n) be a function

Let T (n) is defined on non-negative integers by the recurrence.

- n is the size of the problem.
- a is the number of subproblems in the recursion.
- n/b is the size of each subproblem. (Here it is assumed that all subproblems are essentially the same size.)
- f (n) is the sum of the work done outside the recursive calls, which includes the sum of dividing the problem and the sum of combining the solutions to the subproblems.

It is not possible always bound the function according to the requirement, so we make three cases which will tell us what kind of bound we can apply on the function.

Different methods to solve Recurrence Relations (Cont..)

4. Master Method (Cont..)

It is not possible always bound the function according to the requirement, so we make three cases which will tell us what kind of bound we can apply on the function.

$$T(n) = \left\{ \begin{array}{ll} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a-\varepsilon}\right) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right) \text{AND} \\ & af(n/b) < cf(n) \text{ forlarge} n \end{array} \right\} \begin{array}{l} \varepsilon > 0 \\ c < 1 \end{array}$$

Different methods to solve Recurrence Relations (Cont..)

4. Master Method (Cont..)

Compare $T(n) = 8\,T(n/2)+1000n^2$

Master Method with

$T(n) = a\,T(n/2+f(n)$ with $a>=1$ and $b>1$

$a = 8$, $b=2$, $f(n) = 1000\,n^2$, $\log_b a = \log_2 8 = 3$

Put all the values in: $f(n) = O(n^{\log_b a-e})$ Master Method

$1000\,n^2 = O(n^{3-\varepsilon})$

If we choose $\varepsilon=1$, we get: $1000\,n^2 = O(n^3\text{-}1) = O(n^2)$

Different methods to solve Recurrence Relations (Cont..)

4. Master Method (Cont..)

Compare $T (n) = 8 T(n/2)+1000n^2$

Master Method with

$T (n) = a T(n/2+f(n)$ with $a>=1$ and $b>1$

$a = 8$, $b=2$, $f (n) = 1000 n^2$, $\log_b a = \log_2 8 = 3$

Put all the values in: $f (n) = O(n^{\log_b a-e})$ Master Method

$1000 n^2 = O (n^{3-\varepsilon})$

If we choose $\varepsilon=1$, we get: $1000 n^2 = O (n^3-1) = O (n^2)$

It holds first case of the master theorem thus:

$T (n) = \Theta (^{n\log_b a})$ Master Method

Therefore: $T (n) = \Theta (n^3)$

K.R. MANGALAM UNIVERSITY
THE COMPLETE WORLD OF EDUCATION

Different methods to solve Recurrence Relations (Cont..)

Various types of Recurrence Relations

1. Linear Recurrence Relations
2. Divide and Conquer Recurrences
3. Substitution Recurrences
4. Homogeneous Recurrences
5. Non-Homogeneous Recurrences

Different methods to solve Recurrence Relations (Cont..)

1. Linear Recurrence Relations

- A function calls itself with a reduced version of the problem until it reaches a base case where it can be solved directly.
- Consider the Fibonacci sequence: $F(n)=F(n-1)+F(n-2)F(n)=F(n-1)+F(n-2)$ with base cases $F(0)=0F(0)=0$ and $F(1)=1F(1)=1$.
- This is a linear recursion relation because to find the nth Fibonacci number, we recursively call the function for $n-1$ and $n-2n-2$ until we reach the base cases.

Different methods to solve Recurrence Relations (Cont..)

1. Linear Recurrence Relations

- A function calls itself with a reduced version of the problem until it reaches a base case where it can be solved directly.
- Consider the Fibonacci sequence: $F(n)=F(n-1)+F(n-2)F(n)=F(n-1)+F(n-2)$ with base cases $F(0)=0F(0)=0$ and $F(1)=1F(1)=1$.
- This is a linear recursion relation because to find the nth Fibonacci number, we recursively call the function for $n-1$ and $n-2n-2$ until we reach the base cases.

Different methods to solve Recurrence Relations (Cont..)

1. Linear Recurrence Relations (Cont..)

Example :**$T(n) = T(n-1) + n$** for $n>0$ and $T(0) = 1$

Solution: $T(n) = T(n-1) + n$

$T(n-2) + (n-1) + n$

$T(n-k) + (n-(k-1))\ldots\ldots (n-1) + n$

Substituting $k = n$, we get

$T(n) = T(0) + 1 + 2+\ldots\ldots +n = n(n+1)/2 = O(n^2)$

Different methods to solve Recurrence Relations (Cont..)

2. Divide And Conquer

- Fundamental algorithmic paradigm.
- It involves breaking down a problem.
- More manageable subproblems.
- Solving each subproblem independently.

Then combining the solutions of the subproblems to obtain the solution to the original problem.

Different methods to solve Recurrence Relations (Cont..)

2. Divide And Conquer (Cont..)

- Divide: Break the original problem into subproblem.
- Conquer: Recursively solve the subproblems.
- Combine: Combine the solutions subproblems to obtain solution original problem.

Different methods to solve Recurrence Relations (Cont..)

2. Divide And Conquer (Cont..)

Example 1 : $T(n) = 2T(n/2) + cn$

$T(n) = 2T(n/2) + \sqrt{n}$

Soln: For recurrence relation **T(n) = 2T(n/2) + cn,**

Values of **a = 2, b = 2 and k =1**.

Here **logb(a) = log2(2) = 1 = k**.

Therefore, the complexity will be **Θ(nlog2(n))**.

Example 2: **T(n) = 2T(n/2) + √n,**

Values of **a = 2, b = 2 and k =1/2**.

Here **logb(a) = log2(2) = 1 > k**.

Therefore, the complexity will be **Θ(n)**.

Different methods to solve Recurrence Relations (Cont..)

3. Homogeneous Recurrence Relations

Mathematically, a homogeneous recurrence relation of order k is represented as:

a_{n} = f(a_{n-1}, a_{n-2},..., a_{n-k})

with the condition that the above equation equates to 0.

Example: a_{n} = 2*a_{n-1} - a_{n-2}

Different methods to solve Recurrence Relations (Cont..)

4. Non-Homogeneous Recurrence Relations

- Right-hand side is not equal to zero. It can be expressed as:

a_{n} = f(a_{n-1}, a_{n-2},...,a_{n-k}) + g(n)

where g(n) is a function that introduces a term not dependent on the previous terms.
The presence of g(n) makes the recurrence non-homogeneous.
Example: a_{n} = 2*a_{n-1}- a_{n-2} + 3^n

In this case, the term 3^n on the right-hand side makes the recurrence non-homogeneous.

Different methods to solve Recurrence Relations (Cont..)

4. Non-Homogeneous Recurrence Relations

- Right-hand side is not equal to zero. It can be expressed as:

  a_{n} = f(a_{n-1}, a_{n-2},...,a_{n-k}) + g(n)

where g(n) is a function that introduces a term not dependent on the previous terms.
The presence of g(n) makes the recurrence non-homogeneous.
Example: a_{n} = 2*a_{n-1}- a_{n-2} + 3^n

In this case, the term 3^n on the right-hand side makes the recurrence non-homogeneous.

Practice Questions

Q1: Write a recursive function to calculate the nth Fibonacci number. What is the 10th Fibonacci number?

Q2: Write a recursive function to calculate the factorial of a given number n. Calculate the factorial of 5 using this function.

Q3: Write a recursive function to calculate the sum of all elements in an array. Given the array ‘[1,2,3,4,5]’ , what is the sum?

Q4: Write a recursive function to count the number of digits in a given number. How many digits are there in the number 12345?

Practice Questions (Answers)

A1:

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

A2:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Practice Questions (Answers)

A3:
```
def sum_array(arr):
    if len(arr) == 0:
        return 0
    else:
        return arr[0] + sum_array(arr[1:])
```

A4:
```
def count_digits(n):
    if n == 0:
        return 0
    else:
        return 1 + count_digits(n // 10)
```

Practice Questions- Try Yourself

Q1: Write a recursive function to reverse a given string. What is the reverse of the string "recursion"?

Q2: Write a recursive function to solve the Tower of Hanoi problem for n disks. How many moves are required to solve the Tower of Hanoi for 3 disks?

Q3: Write a recursive function to find the sum of the digits of a given number. What is the sum of the digits of the number 1234?

Q4: Write a recursive function to find the GCD of two numbers using Euclid's algorithm. What is the GCD of 48 and 18?

THANK YOU