Data Structures and Algorithms [1]

BITS-Pilani K. K. Birla Goa Campus

[1] Material for the presentation taken from Cormen, Leiserson, Rivest and Stein, *Introduction to Algorithms, Third Edition*;

Divide the problem into a number of subproblems that are smaller instances of the same problem.

Conquer the subproblems by solving them recursively.

Combine the solutions to the subproblems into the solution for the original problem.

Minimum Element

Q. You are given an unsorted array. Find the minimum element in the array using a divide-and-conquer approach.

Minimum Element

Q. You are given an unsorted array. Find the minimum element in the array using a divide-and-conquer approach.

1: **procedure** FINDMINIMUM(arr, $left$, $right$)
2: **if** $left = right$ **then**
3: **return** $arr[left]$
4: $mid \leftarrow \lfloor (left + right)/2 \rfloor$

Minimum Element

Q. You are given an unsorted array. Find the minimum element in the array using a divide-and-conquer approach.

procedure $\textsc{findMinimum}(\mathit{arr}, \mathit{left}, \mathit{right})$

  if $\mathit{left} = \mathit{right}$ **then**

    return $\mathit{arr}[\mathit{left}]$

  $\mathit{mid} \leftarrow \lfloor (\mathit{left} + \mathit{right})/2 \rfloor$

  $\mathit{minLeft} \leftarrow \textsc{findMinimum}(\mathit{arr}, \mathit{left}, \mathit{mid})$

  $\mathit{minRight} \leftarrow \textsc{findMinimum}(\mathit{arr}, \mathit{mid} + 1, \mathit{right})$

Minimum Element

Q. You are given an unsorted array. Find the minimum element in the array using a divide-and-conquer approach.

```
procedure FINDMINIMUM(arr, left, right)
    if left = right then
        return arr[left]
    mid ← ⌊(left + right)/2⌋
    minLeft ← FINDMINIMUM(arr, left, mid)
    minRight ← FINDMINIMUM(arr, mid + 1, right)
    return min(minLeft, minRight)
```

Minimum Element

Q. You are given an unsorted array. Find the minimum element in the array using a divide-and-conquer approach.

```
procedure FINDMINIMUM(arr, left, right)
    if left = right then
        return arr[left]
    mid ← ⌊(left + right)/2⌋
    minLeft ← FINDMINIMUM(arr, left, mid)
    minRight ← FINDMINIMUM(arr, mid + 1, right)
    return min(minLeft, minRight)
```

(a) Find the recurrence equation for the above algorithm?

Minimum Element

Q. You are given an unsorted array. Find the minimum element in the array using a divide-and-conquer approach.

```
procedure FindMinimum(arr, left, right)
    if left = right then
        return arr[left]
    mid ← ⌊(left + right)/2⌋
    minLeft ← FindMinimum(arr, left, mid)
    minRight ← FindMinimum(arr, mid + 1, right)
    return min(minLeft, minRight)
```

(a) Find the recurrence equation for the above algorithm?

(b) What is the worst case running time of the algorithm?

Matrix multiplication

- $C = A \cdot B$, where A, B and C are $n \times n$ matrices

Matrix multiplication

- $C = A \cdot B$, where A, B and C are $n \times n$ matrices
- $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$

Matrix multiplication

- $C = A \cdot B$, where A, B and C are $n \times n$ matrices
- $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$

```
MATRIX-MULTIPLY(A, B, C, n)
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
```

Matrix multiplication

- MATRIX-MULTIPLY procedure runs in $\Theta(n^3)$ time.

Matrix multiplication

- MATRIX-MULTIPLY procedure runs in $\Theta(n^3)$ time.
- Volker Strassen came up with an algorithm that takes $O(n^{\lg 7})$ time, which is $o(n^{2.81})$. $(\lg 7 = 2.8074)$

Matrix multiplication

- MATRIX-MULTIPLY procedure runs in $\Theta(n^3)$ time.
- Volker Strassen came up with an algorithm that takes $O(n^{\lg 7})$ time, which is $o(n^{2.81})$. $(\lg 7 = 2.8074)$
- Strassen's algorithm uses a divide-and-conquer approach.

Matrix multiplication: divide-and-conquer

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

Matrix multiplication: divide-and-conquer

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Matrix multiplication: divide-and-conquer

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \end{aligned}$$

Matrix multiplication: divide-and-conquer

```
MATRIX-MULTIPLY-RECURSIVE(A, B, C, n)
if n == 1
// Base case.
    c11 = c11 + a11 · b11
    return
// Divide.
partition A, B, and C into n/2 × n/2 submatrices
    A11, A12, A21, A22; B11, B12, B21, B22;
    and C11, C12, C21, C22; respectively
// Conquer.
MATRIX-MULTIPLY-RECURSIVE(A11, B11, C11, n/2)
MATRIX-MULTIPLY-RECURSIVE(A11, B12, C12, n/2)
MATRIX-MULTIPLY-RECURSIVE(A21, B11, C21, n/2)
MATRIX-MULTIPLY-RECURSIVE(A21, B12, C22, n/2)
MATRIX-MULTIPLY-RECURSIVE(A12, B21, C11, n/2)
MATRIX-MULTIPLY-RECURSIVE(A12, B22, C12, n/2)
MATRIX-MULTIPLY-RECURSIVE(A22, B21, C21, n/2)
MATRIX-MULTIPLY-RECURSIVE(A22, B22, C22, n/2)
```

Matrix multiplication: divide-and-conquer

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \,, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \,, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \,, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \,. \end{aligned}$$

Matrix multiplication: divide-and-conquer

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} , \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} , \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} , \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} . \end{aligned}$$

- $T(n) = 8T(n/2) + D(n) + C(n)$

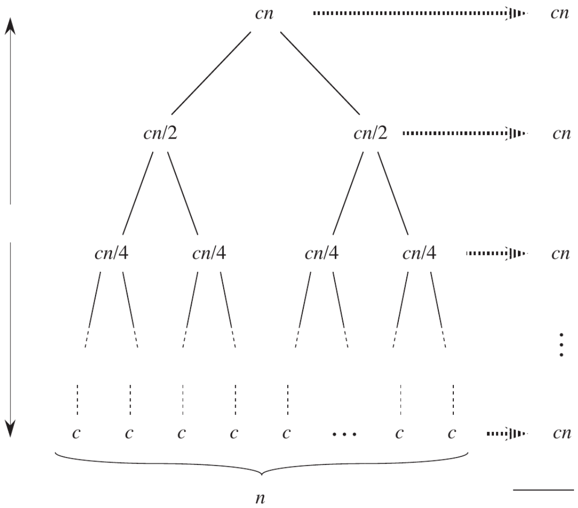
Matrix multiplication: divide-and-conquer

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \, , \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \, , \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \, , \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \, . \end{aligned}$$

- $T(n) = 8T(n/2) + D(n) + C(n)$
- $T(n) = 8T(n/2) + \Theta(1) + \Theta(1)$

Recursion tree for merge sort



Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \ldots$

Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \dots$
  $T(n) = cn^{\lg 8} + \dots$

Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \ldots$
  $T(n) = cn^{\lg 8} + \ldots$
  $T(n) = \Theta(n^3)$

Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \dots$
  $T(n) = cn^{\lg 8} + \dots$
  $T(n) = \Theta(n^3)$
- Strassen modified the divide-and-conquer approach explained above such that $T(n) = O(n^{\lg 7})$

Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \dots$
  $T(n) = cn^{\lg 8} + \dots$
  $T(n) = \Theta(n^3)$
- Strassen modified the divide-and-conquer approach explained above such that $T(n) = O(n^{\lg 7})$
- $T(n) = c7^{\lg n} + \dots$

Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \dots$
  $T(n) = cn^{\lg 8} + \dots$
  $T(n) = \Theta(n^3)$
- Strassen modified the divide-and-conquer approach explained above such that $T(n) = O(n^{\lg 7})$
- $T(n) = c7^{\lg n} + \dots$
  $T(n) = cn^{\lg 7} + \dots$

Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \ldots$
  $T(n) = cn^{\lg 8} + \ldots$
  $T(n) = \Theta(n^3)$
- Strassen modified the divide-and-conquer approach explained above such that $T(n) = O(n^{\lg 7})$
- $T(n) = c7^{\lg n} + \ldots$
  $T(n) = cn^{\lg 7} + \ldots$
  $T(n) = \Theta(n^{\lg 7})$

Matrix multiplication: divide-and-conquer

- $T(n) = c8^{\lg n} + \dots$
  $T(n) = cn^{\lg 8} + \dots$
  $T(n) = \Theta(n^3)$
- Strassen modified the divide-and-conquer approach explained above such that $T(n) = O(n^{\lg 7})$
- $T(n) = c7^{\lg n} + \dots$
  $T(n) = cn^{\lg 7} + \dots$
  $T(n) = \Theta(n^{\lg 7})$

n	0	1	2	3	4	5	6
8^n	1	8	64	512	4096	32768	262144
7^n	1	7	49	343	2401	16807	117649
2^n	1	2	4	8	16	32	64

Strassen's algorithm : Divide (Step 1)

Divide step involves finding ten $n/2 \times n/2$ matrices (S_i).

Divide step involves finding ten $n/2 \times n/2$ matrices (S_i).

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} , \\
S_2 &= A_{11} + A_{12} , \\
S_3 &= A_{21} + A_{22} , \\
S_4 &= B_{21} - B_{11} , \\
S_5 &= A_{11} + A_{22} , \\
S_6 &= B_{11} + B_{22} , \\
S_7 &= A_{12} - A_{22} , \\
S_8 &= B_{21} + B_{22} , \\
S_9 &= A_{11} - A_{21} , \\
S_{10} &= B_{11} + B_{12} .
\end{aligned}
$$

Strassen's algorithm : Conquer (Step 2)

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 \\
P_2 &= S_2 \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 \\
P_5 &= S_5 \cdot S_6 \\
P_6 &= S_7 \cdot S_8 \\
P_7 &= S_9 \cdot S_{10}
\end{aligned}
$$

Strassen's algorithm : Conquer (Step 2)

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} , \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} , \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} , \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} , \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} , \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} , \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} .
\end{aligned}
$$

Matrix multiplication: divide-and-conquer

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} , \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} , \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} , \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} . \end{aligned}$$

Strassen's algorithm : Combine (Step 3)

$$C_{11} = P_5 + P_4 - P_2 + P_6 \, .$$

$$\begin{array}{l}
A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
\qquad\qquad\qquad\qquad - A_{22} \cdot B_{11} \qquad\qquad\qquad\qquad + A_{22} \cdot B_{21} \\
\qquad\quad - A_{11} \cdot B_{22} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad - A_{12} \cdot B_{22} \\
\qquad\qquad\qquad\qquad\qquad\qquad - A_{22} \cdot B_{22} - A_{22} \cdot B_{21} + A_{12} \cdot B_{22} + A_{12} \cdot B_{21} \\
\hline
A_{11} \cdot B_{11} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + A_{12} \cdot B_{21} \, ,
\end{array}$$

Strassen's algorithm : Combine (Step 3)

$$C_{12} = P_1 + P_2 \, ,$$

$$\begin{array}{l}
A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
\qquad\qquad\quad + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
\hline
A_{11} \cdot B_{12} \qquad\qquad\qquad\quad + A_{12} \cdot B_{22} \, ,
\end{array}$$

Strassen's algorithm : Combine (Step 3)

$$C_{12} = P_1 + P_2 ,$$

$$\begin{array}{rl} & A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ & \qquad + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\ \hline & A_{11} \cdot B_{12} \qquad\qquad + A_{12} \cdot B_{22} , \end{array}$$

$$C_{21} = P_3 + P_4$$

$$\begin{array}{rl} & A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ & \qquad - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\ \hline & A_{21} \cdot B_{11} \qquad\qquad + A_{22} \cdot B_{21} , \end{array}$$

Strassen's algorithm : Combine (Step 3)

$$C_{12} = P_1 + P_2\ ,$$

$$\begin{array}{l} A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ \qquad\qquad + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\ \hline A_{11} \cdot B_{12} \qquad\qquad\qquad + A_{12} \cdot B_{22}\ , \end{array}$$

$$C_{21} = P_3 + P_4$$

$$\begin{array}{l} A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ \qquad\qquad - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\ \hline A_{21} \cdot B_{11} \qquad\qquad\qquad + A_{22} \cdot B_{21}\ , \end{array}$$

$$C_{22} = P_5 + P_1 - P_3 - P_7\ ,$$

$$\begin{array}{l} \quad A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ \qquad\qquad - A_{11} \cdot B_{22} \qquad\qquad\qquad\qquad + A_{11} \cdot B_{12} \\ \qquad\qquad\qquad\qquad - A_{22} \cdot B_{11} \qquad\qquad\qquad\qquad - A_{21} \cdot B_{11} \\ - A_{11} \cdot B_{11} \qquad\qquad\qquad\qquad\qquad - A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12} \\ \hline \qquad\qquad\qquad\qquad\qquad A_{22} \cdot B_{22} \qquad\qquad\qquad\qquad + A_{21} \cdot B_{12}\ , \end{array}$$

Strassen's algorithm

- $T(n) = 7T(n/2) + D(n) + C(n)$

Strassen's algorithm

- $T(n) = 7T(n/2) + D(n) + C(n)$
- $T(n) = 7T(n/2) + \Theta(n^2) + \Theta(n^2)$

Strassen's algorithm

- $T(n) = 7T(n/2) + D(n) + C(n)$
- $T(n) = 7T(n/2) + \Theta(n^2) + \Theta(n^2)$
- $T(n) = 7^{\lg n} + \ldots$

Strassen's algorithm

- $T(n) = 7T(n/2) + D(n) + C(n)$
- $T(n) = 7T(n/2) + \Theta(n^2) + \Theta(n^2)$
- $T(n) = 7^{\lg n} + \ldots$

  $T(n) = n^{\lg 7} + \ldots$

Strassen's algorithm

- $T(n) = 7T(n/2) + D(n) + C(n)$
- $T(n) = 7T(n/2) + \Theta(n^2) + \Theta(n^2)$
- $T(n) = 7^{\lg n} + \ldots$

  $T(n) = n^{\lg 7} + \ldots$

  $T(n) = O(n^{\lg 7}) = o(n^{2.81})$

Strassen's algorithm

- $T(n) = 7T(n/2) + D(n) + C(n)$
- $T(n) = 7T(n/2) + \Theta(n^2) + \Theta(n^2)$
- $T(n) = 7^{\lg n} + \dots$

  $T(n) = n^{\lg 7} + \dots$

  $T(n) = O(n^{\lg 7}) = o(n^{2.81})$
- Why should n be a power of 2?

Q. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2?

Strassen's algorithm

Q. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2?

$$n^{\lg 7} <= m^{\lg 7} <$$

Strassen's algorithm

Q. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2?

$$n^{\lg 7} <= m^{\lg 7} < (2n)^{\lg 7}$$

Strassen's algorithm

Q. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2?

$$n^{\lg 7} <= m^{\lg 7} < (2n)^{\lg 7} = 2^{\lg 7} n^{\lg 7}$$

Strassen's algorithm

Q. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2?

$$n^{\lg 7} <= m^{\lg 7} < (2n)^{\lg 7} = 2^{\lg 7} n^{\lg 7} = 7n^{\lg 7}$$

Q. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2?

$$n^{\lg 7} <= m^{\lg 7} < (2n)^{\lg 7} = 2^{\lg 7} n^{\lg 7} = 7 n^{\lg 7}$$

Q. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2?

$$n^{\lg 7} <= m^{\lg 7} < (2n)^{\lg 7} = 2^{\lg 7} n^{\lg 7} = 7n^{\lg 7}$$

The above implies that $m^{\lg 7} = \Theta(n^{\lg 7})$.

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Find an upper bound for $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Find an upper bound for $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- We will first guess a solution: $O(n \lg n)$ (using Recursion tree)

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Find an upper bound for $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- We will first guess a solution: $O(n \lg n)$ (using Recursion tree)
- To show : $T(n) = O(n \lg n)$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Find an upper bound for $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- We will first guess a solution: $O(n \lg n)$ (using Recursion tree)
- To show : $T(n) = O(n \lg n)$

  $T(n) \leq dn \lg n$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- Find an upper bound for $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- We will first guess a solution: $O(n \lg n)$ (using Recursion tree)
- To show : $T(n) = O(n \lg n)$

  $T(n) \leq dn \lg n$
- We will assume that $T(n/2) \leq d(n/2)\lg(n/2)$. Then show that $T(n) \leq dn \lg n$. (Inductive step)

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \le d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \le d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \le d(n/2)\lg(n/2)$$

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\le 2d(n/2)\lg(n/2) + n \end{aligned}$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2d(n/2)\lg(n/2) + n \\ &= dn(\lg n - \lg 2) + n \end{aligned}$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2d(n/2)\lg(n/2) + n \\ &= dn(\lg n - \lg 2) + n \\ &= dn(\lg n - 1) + n \end{aligned}$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \le d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \le d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \le d(n/2)\lg(n/2)$$

$$\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\le 2d(n/2)\lg(n/2) + n \\
&= dn(\lg n - \lg 2) + n \\
&= dn(\lg n - 1) + n \\
&= dn\lg n - dn + n
\end{aligned}$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2d(n/2)\lg(n/2) + n \\ &= dn(\lg n - \lg 2) + n \\ &= dn(\lg n - 1) + n \\ &= dn\lg n - dn + n \\ &\leq dn\lg n \text{ , for } d \geq 1 \end{aligned}$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

$$\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\leq 2d(n/2)\lg(n/2) + n \\
&= dn(\lg n - \lg 2) + n \\
&= dn(\lg n - 1) + n \\
&= dn\lg n - dn + n \\
&\leq dn\lg n \text{ , for } d \geq 1
\end{aligned}$$

Base case is problematic

$$T(1) = 1$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

$$\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\leq 2d(n/2)\lg(n/2) + n \\
&= dn(\lg n - \lg 2) + n \\
&= dn(\lg n - 1) + n \\
&= dn\lg n - dn + n \\
&\leq dn\lg n \text{ , for } d \geq 1
\end{aligned}$$

Base case is problematic

$$T(1) = 1 \not\leq d\,1\lg 1$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Inductive step : Assume $T(n/2) \leq d(n/2)\lg(n/2)$

$$T(\lfloor n/2 \rfloor) \leq d(\lfloor n/2 \rfloor)\lg(\lfloor n/2 \rfloor) \leq d(n/2)\lg(n/2)$$

$$\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\leq 2d(n/2)\lg(n/2) + n \\
&= dn(\lg n - \lg 2) + n \\
&= dn(\lg n - 1) + n \\
&= dn\lg n - dn + n \\
&\leq dn\lg n \text{ , for } d \geq 1
\end{aligned}$$

Base case is problematic

$$T(1) = 1 \not\leq d\,1\lg 1 = 0$$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 4$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 4 \leq d2 \lg 2$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 4 \leq d2\lg 2$
- $T(3) = 2T(\lfloor 3/2 \rfloor) + 3$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 4 \leq d2 \lg 2$
- $T(3) = 2T(\lfloor 3/2 \rfloor) + 3 = 5$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 4 \leq d2\lg 2$
- $T(3) = 2T(\lfloor 3/2 \rfloor) + 3 = 5 \leq d3\lg 3$

Substitution method : $T(n) = 2T(\lfloor n/2 \rfloor) + n$

- $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 4 \leq d2 \lg 2$
- $T(3) = 2T(\lfloor 3/2 \rfloor) + 3 = 5 \leq d3 \lg 3$
- We only need to show that $T(n) \leq dn \lg n$ for $n \geq n_0$, where n_0 need not be 1.

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.

- Find a tight upper bound for the recurrence
  $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.
  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.
  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$
  Induction step assumption :

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.
  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$
  Induction step assumption :

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) \quad , \quad T(\lceil n/2 \rceil) \leq c(\lceil n/2 \rceil)$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.
  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$
  Induction step assumption :

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) \quad , \quad T(\lceil n/2 \rceil) \leq c(\lceil n/2 \rceil)$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.
  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$
  Induction step assumption :

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) \quad , \quad T(\lceil n/2 \rceil) \leq c(\lceil n/2 \rceil)$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.
  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$
  Induction step assumption :

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) \quad , \quad T(\lceil n/2 \rceil) \leq c(\lceil n/2 \rceil)$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.
  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$
  Induction step assumption :

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) \quad , \quad T(\lceil n/2 \rceil) \leq c(\lceil n/2 \rceil)$$

$$\begin{aligned} T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ T(n) &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Find a tight upper bound for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$
- What can be an initial guess for the above question based on recursion tree?
- We guess $T(n) = O(n)$.

  To prove : $T(n) \leq cn \qquad \forall\, n \geq n_0$

  Induction step assumption :

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) \quad , \quad T(\lceil n/2 \rceil) \leq c(\lceil n/2 \rceil)$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

$$= cn + 1 \text{ (cannot proceed)}$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence
$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) - d$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence
$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) - d$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$T(\lfloor n/2 \rfloor) \leq c(\lfloor n/2 \rfloor) - d$$

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \leq c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$\begin{aligned}
T(\lfloor n/2 \rfloor) &\leq c(\lfloor n/2 \rfloor) - d \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\leq c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 \\
&= cn - 2d + 1
\end{aligned}$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$\begin{aligned} T(\lfloor n/2 \rfloor) &\leq c(\lfloor n/2 \rfloor) - d \\ T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\leq c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 \\ &= cn - 2d + 1 \\ &= cn - d - d + 1 \end{aligned}$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$
\begin{aligned}
T(\lfloor n/2 \rfloor) &\leq c(\lfloor n/2 \rfloor) - d \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\leq c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 \\
&= cn - 2d + 1 \\
&= cn - d - d + 1 \\
&\leq cn - d \text{ , for } d \geq 1
\end{aligned}
$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$
\begin{aligned}
T(\lfloor n/2 \rfloor) &\leq c(\lfloor n/2 \rfloor) - d \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\leq c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 \\
&= cn - 2d + 1 \\
&= cn - d - d + 1 \\
&\leq cn - d \text{ , for } d \geq 1 \\
T(1) = 1 \leq cn - d &
\end{aligned}
$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \le cn - d \qquad \forall\, n \ge n_0$.

$$
\begin{aligned}
T(\lfloor n/2 \rfloor) &\le c(\lfloor n/2 \rfloor) - d \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\le c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 \\
&= cn - 2d + 1 \\
&= cn - d - d + 1 \\
&\le cn - d \text{ , for } d \ge 1
\end{aligned}
$$

$$T(1) = 1 \le cn - d \text{ , for } d \ge 1 \text{ and } c \ge d + 1$$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \leq cn - d \qquad \forall\, n \geq n_0$.

$$
\begin{aligned}
T(\lfloor n/2 \rfloor) &\leq c(\lfloor n/2 \rfloor) - d \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\leq c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 \\
&= cn - 2d + 1 \\
&= cn - d - d + 1 \\
&\leq cn - d \text{ , for } d \geq 1
\end{aligned}
$$

$$T(1) = 1 \leq cn - d \text{ , for } d \geq 1 \text{ and } c \geq d + 1$$

For $c = 2,\ d = 1$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

We need to slightly modify the initial guess for the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$.
To prove: $T(n) \le cn - d \qquad \forall\, n \ge n_0$.

$$
\begin{aligned}
T(\lfloor n/2 \rfloor) &\le c(\lfloor n/2 \rfloor) - d \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 &\le c\lfloor n/2 \rfloor - d + c\lceil n/2 \rceil - d + 1 \\
&= cn - 2d + 1 \\
&= cn - d - d + 1 \\
&\le cn - d \text{ , for } d \ge 1 \\
T(1) = 1 \le cn - d &\text{ , for } d \ge 1 \text{ and } c \ge d + 1
\end{aligned}
$$

For $c = 2,\ d = 1 \quad T(n) \le 2n - 1 \quad \forall\, n \ge 1$

Substitution method : $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

- Instead of showing $T(n) = O(n)$ we have instead shown than $T(n) = O(cn - d)$, which implies $T(n) = O(n)$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess?

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Induction step:

$$T(n-1) \geq c(n-1)^2$$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Induction step:

$$\begin{aligned} T(n-1) &\geq c(n-1)^2 \\ &= c(n^2 + 1 - 2n) = cn^2 + c - 2cn \end{aligned}$$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Induction step:

$$\begin{aligned} T(n-1) &\geq c(n-1)^2 \\ &= c(n^2 + 1 - 2n) = cn^2 + c - 2cn \\ T(n) &\geq cn^2 + c - 2cn + n \end{aligned}$$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Induction step:

$$\begin{aligned} T(n-1) &\geq c(n-1)^2 \\ &= c(n^2 + 1 - 2n) = cn^2 + c - 2cn \\ T(n) &\geq cn^2 + c - 2cn + n \text{ (when c=1/2 , R.H.S. } \geq cn^2) \end{aligned}$$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Induction step:

$$\begin{aligned} T(n-1) &\geq c(n-1)^2 \\ &= c(n^2 + 1 - 2n) = cn^2 + c - 2cn \\ T(n) &\geq cn^2 + c - 2cn + n \text{ (when c=1/2 , R.H.S. } \geq cn^2) \\ &\geq cn^2 \text{ (for c=1/2 and } n \geq 1) \end{aligned}$$

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Induction step:

$$
\begin{aligned}
T(n-1) &\geq c(n-1)^2 \\
&= c(n^2 + 1 - 2n) = cn^2 + c - 2cn \\
T(n) &\geq cn^2 + c - 2cn + n \text{ (when c=1/2 , R.H.S.} \geq cn^2) \\
&\geq cn^2 \text{ (for c=1/2 and } n \geq 1)
\end{aligned}
$$

Base case:

Substitution method : $T(n) = T(n-1) + n$

- Find a lower bound for the recurrence $T(n) = T(n-1) + n$
- What would be a good guess? $T(n) = \Omega(n^2)$

To show : $T(n) \geq cn^2 \quad \forall\, n \geq n_0$

Induction step:

$$\begin{aligned} T(n-1) &\geq c(n-1)^2 \\ &= c(n^2 + 1 - 2n) = cn^2 + c - 2cn \\ T(n) &\geq cn^2 + c - 2cn + n \text{ (when c=1/2 , R.H.S. } \geq cn^2) \\ &\geq cn^2 \text{ (for c=1/2 and } n \geq 1) \end{aligned}$$

Base case:

$$T(1) \geq cn^2$$

- Recursion tree can be used to generate a good guess for the Substitution method.

Solving recurrences : Recursion tree

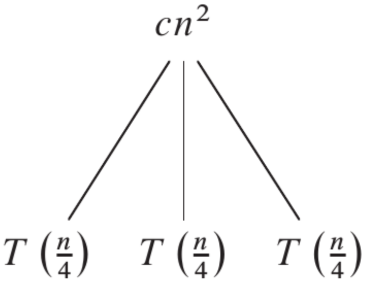
- Recursion tree can be used to generate a good guess for the Substitution method.
- Finding a good guess for $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

Solving recurrences : Recursion tree

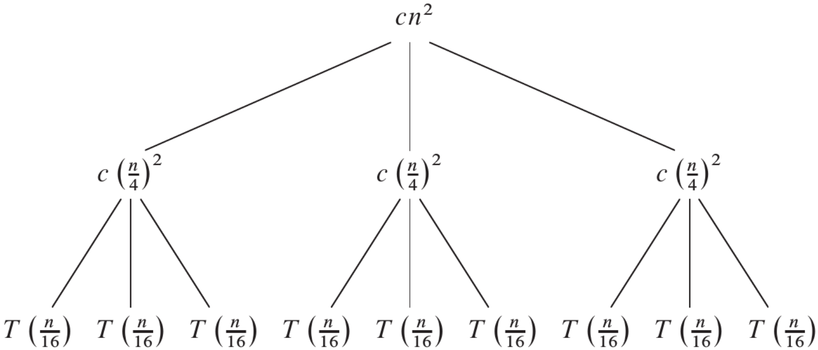
- Recursion tree can be used to generate a good guess for the Substitution method.
- Finding a good guess for $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$
- We will ignore the floors and ceilings, because usually they don't affect the asymptotic order of growth.

Recursion tree : $T(n) = 3T(n/4) + cn^2$

$T(n)$

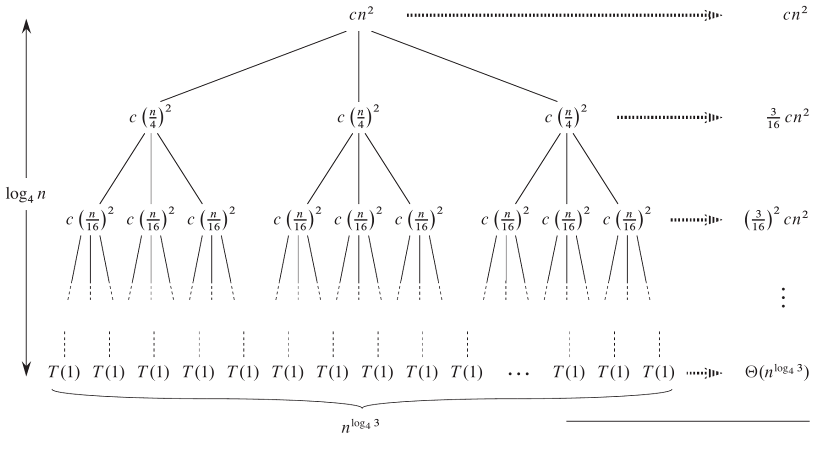


Recursion tree : $T(n) = 3T(n/4) + cn^2$



(c)

Recursion tree : $T(n) = 3T(n/4) + cn^2$



(d)

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Number of levels :

$$\frac{n}{4^{i-1}} = 1$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Number of levels :

$$\frac{n}{4^{i-1}} = 1$$

$$i = \log_4 n + 1$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Number of levels :

$$\frac{n}{4^{i-1}} = 1$$
$$i = \log_4 n + 1$$

- Running time for the non-leaf levels:

$$cn^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \dots\right) (\ \log_4 n \text{ terms})$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Number of levels :

$$\frac{n}{4^{i-1}} = 1$$
$$i = \log_4 n + 1$$

- Running time for the non-leaf levels:

$$cn^2 \left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \dots\right) (\ \log_4 n \text{ terms})$$
$$< cn^2 \left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \dots\right) (\text{ infinite series sum})$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Number of levels :

$$\frac{n}{4^{i-1}} = 1$$
$$i = \log_4 n + 1$$

- Running time for the non-leaf levels:

$$cn^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \dots\right) (\ \log_4 n \text{ terms})$$
$$< cn^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \dots\right) (\text{ infinite series sum})$$
$$= \frac{16}{13}cn^2$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Number of levels :

$$\frac{n}{4^{i-1}} = 1$$
$$i = \log_4 n + 1$$

- Running time for the non-leaf levels:

$$cn^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \dots\right) (\ \log_4 n \text{ terms})$$
$$< cn^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \dots\right) (\ \text{infinite series sum})$$
$$= \frac{16}{13}cn^2 = O(n^2)$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Running time for leaf level nodes :

$$3^{log_4 n} = n^{log_4 3}$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- Running time for leaf level nodes :

$$3^{log_4 n} = n^{log_4 3}$$

- Total running time

- Running time for leaf level nodes :

$$3^{log_4 n} = n^{log_4 3}$$

- Total running time

$$= O(n^2) + \Theta(n^{log_4 3})$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$

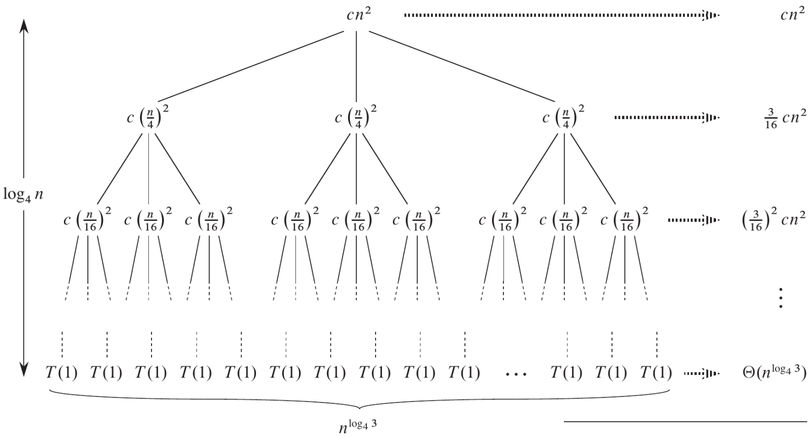
- Running time for leaf level nodes :

$$3^{log_4 n} = n^{log_4 3}$$

- Total running time

$$\begin{aligned} &= O(n^2) + \Theta(n^{log_4 3}) \\ &= O(n^2) \end{aligned}$$

Recursion tree : $T(n) = 3T(n/4) + cn^2$



(d)

- By looking at the recursion tree we can guess $T(n) = O(n^2)$

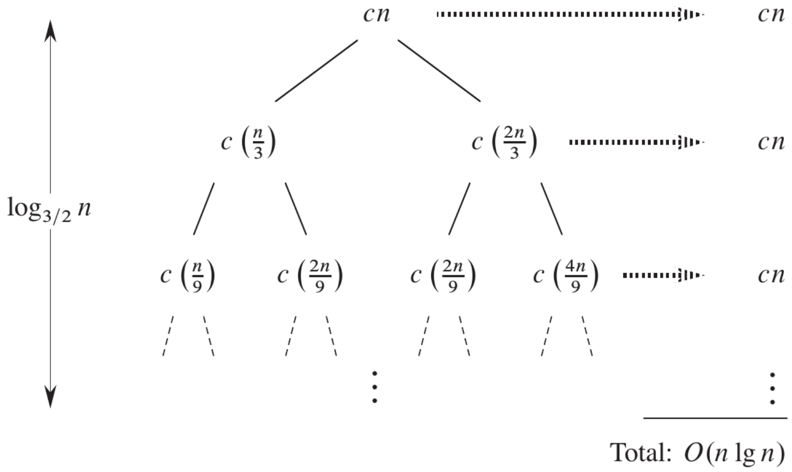
Recursion tree : $T(n) = 3T(n/4) + cn^2$

- By looking at the recursion tree we can guess $T(n) = O(n^2)$
- Then we can use the substitution method to show that our guess is correct.

Recursion tree : $T(n) = 3T(n/4) + cn^2$

- By looking at the recursion tree we can guess $T(n) = O(n^2)$
- Then we can use the substitution method to show that our guess is correct.
- To prove : $T(n) \leq dn^2 \quad \forall\, n \geq n_0$

Recursion tree : $T(n) = T(n/3) + T(2n/3) + O(n)$



Recursion tree : $T(n) = T(n/3) + T(2n/3) + O(n)$

- Number of levels :

$$\frac{n}{\left(\frac{3}{2}\right)^{i-1}} = 1$$

Recursion tree : $T(n) = T(n/3) + T(2n/3) + O(n)$

- Number of levels :

$$\frac{n}{\left(\frac{3}{2}\right)^{i-1}} = 1$$

$$i = \log_{\frac{3}{2}} n + 1$$

Recursion tree : $T(n) = T(n/3) + T(2n/3) + O(n)$

- Number of levels :

$$\frac{n}{\left(\frac{3}{2}\right)^{i-1}} = 1$$

$$i = \log_{\frac{3}{2}} n + 1$$

- Running time :

$$T(n) = O(n \log_{\frac{3}{2}} n)$$

Recursion tree : $T(n) = T(n/3) + T(2n/3) + O(n)$

- Number of levels :

$$\frac{n}{\left(\frac{3}{2}\right)^{i-1}} = 1$$

$$i = \log_{\frac{3}{2}} n + 1$$

- Running time :

$$T(n) = O(n \log_{\frac{3}{2}} n)$$

- Is $\Theta(\log_{10} n) = \Theta(\log_3 n)$?

Recursion tree : $T(n) = T(n/3) + T(2n/3) + O(n)$

- Number of levels :

$$\frac{n}{\left(\frac{3}{2}\right)^{i-1}} = 1$$

$$i = \log_{\frac{3}{2}} n + 1$$

- Running time :

$$T(n) = O(n \log_{\frac{3}{2}} n)$$

- Is $\Theta(\log_{10} n) = \Theta(\log_3 n)$?
- Running time :

$$T(n) = O(n \log_{\frac{3}{2}} n) = O(n \lg n)$$

Recursion tree : $T(n) = T(n/3) + T(2n/3) + O(n)$

- Number of levels :

$$\frac{n}{\left(\frac{3}{2}\right)^{i-1}} = 1$$

$$i = \log_{\frac{3}{2}} n + 1$$

- Running time :

$$T(n) = O(n \log_{\frac{3}{2}} n)$$

- Is $\Theta(\log_{10} n) = \Theta(\log_3 n)$?
- Running time :

$$T(n) = O(n \log_{\frac{3}{2}} n) = O(n \lg n)$$

- We can again use the substitution method to show the upper bound.

Polynomially larger/smaller

- For any positive ϵ (however small), is $n^{\epsilon} = \omega(\lg n)$?

Polynomially larger/smaller

- For any positive ϵ (however small), is $n^{\epsilon} = \omega(\lg n)$?
- Using L'Hospital's rule:

$$\lim_{n \to \infty} \frac{\lg n}{n^{\epsilon}} = 0$$

Polynomially larger/smaller

- For any positive ϵ (however small), is $n^\epsilon = \omega(\lg n)$?
- Using L'Hospital's rule:

$$\lim_{n \to \infty} \frac{\lg n}{n^\epsilon} = 0$$

- Is $n^{1+\epsilon} = \omega(n \lg n)$?

Polynomially larger/smaller

- $n = o(n^2)$

Polynomially larger/smaller

- $n = o(n^2)$ (n is asymptotically smaller)

Polynomially larger/smaller

- $n = o(n^2)$ (n is asymptotically smaller)
- $n^{1+\epsilon} = o(n^2)$

Polynomially larger/smaller

- $n = o(n^2)$ (n is asymptotically smaller)
- $n^{1+\epsilon} = o(n^2)$ (n is polynomially smaller because we can find a positive constant ϵ)

Polynomially larger/smaller

- $n = o(n^2)$ (n is asymptotically smaller)
- $n^{1+\epsilon} = o(n^2)$ (n is polynomially smaller because we can find a positive constant ϵ)
- $n = o(n \lg n)$

Polynomially larger/smaller

- $n = o(n^2)$ (n is asymptotically smaller)
- $n^{1+\epsilon} = o(n^2)$ (n is polynomially smaller because we can find a positive constant ϵ)
- $n = o(n \lg n)$ (n is asymptotically smaller)

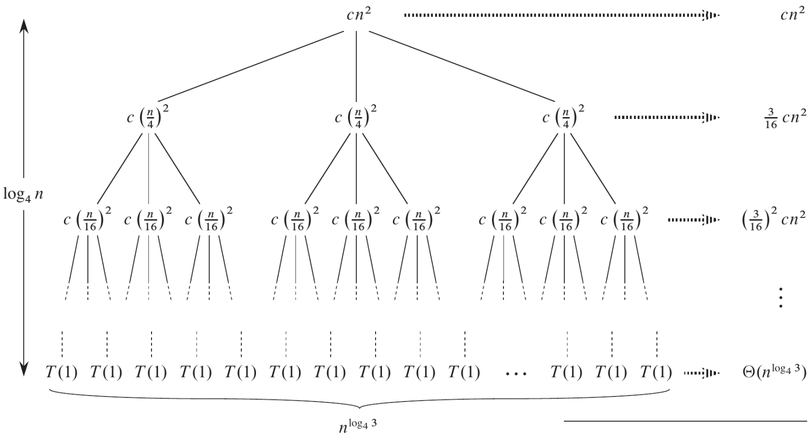
Polynomially larger/smaller

- $n = o(n^2)$ (n is asymptotically smaller)
- $n^{1+\epsilon} = o(n^2)$ (n is polynomially smaller because we can find a positive constant ϵ)
- $n = o(n \lg n)$ (n is asymptotically smaller)
- However, $n^{1+\epsilon} \neq o(n \lg n)$

Polynomially larger/smaller

- $n = o(n^2)$ (n is asymptotically smaller)
- $n^{1+\epsilon} = o(n^2)$ (n is polynomially smaller because we can find a positive constant ϵ)
- $n = o(n \lg n)$ (n is asymptotically smaller)
- However, $n^{1+\epsilon} \neq o(n \lg n)$ (n is **not** polynomially smaller)

Master method : $T(n) = 3T(n/4) + cn^2$



(d)

Solving recurrences : Master method

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 9T(n/3) + n$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 9T(n/3) + n$

$$n^{\log_b a} = n^{\log_3 9}$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 9T(n/3) + n$

$$n^{\log_b a} = n^{\log_3 9}$$

$$f(n) = n =$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 9T(n/3) + n$

$$n^{\log_b a} = n^{\log_3 9}$$

$$f(n) = n = O(n^{2-\epsilon})$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ∎

- $T(n) = 9T(n/3) + n$

$$n^{\log_b a} = n^{\log_3 9}$$
$$f(n) = n = O(n^{2-\epsilon})$$
$$T(n) = \Theta(n^2)$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = T(2n/3) + 1$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ∎

- $T(n) = T(2n/3) + 1$

$$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1}$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = T(2n/3) + 1$

$$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = T(2n/3) + 1$

$$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0$$

$$f(n) = \Theta(1)$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = T(2n/3) + 1$

$$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0$$
$$f(n) = \Theta(1) = \Theta(n^{\log_b a})$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ∎

- $T(n) = T(2n/3) + 1$

$$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0$$

$$f(n) = \Theta(1) = \Theta(n^{\log_b a})$$

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

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = T(2n/3) + 1$

$$
\begin{aligned}
n^{\log_b a} &= n^{\log_{\frac{3}{2}} 1} = n^0 \\
f(n) &= \Theta(1) = \Theta(n^{\log_b a}) \\
T(n) &= \Theta(n^{\log_b a} \lg n) \\
&= \Theta(\lg n)
\end{aligned}
$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 3T(n/4) + n \lg n$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 3T(n/4) + n \lg n$

$$n^{\log_b a} = n^{\log_4 3}$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 3T(n/4) + n \lg n$

$$n^{\log_b a} = n^{\log_4 3}$$

$$f(n) = n \lg n$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 3T(n/4) + n \lg n$

$$n^{\log_b a} = n^{\log_4 3}$$

$$f(n) = n \lg n = \Omega(n^{\log_4 3 + \epsilon}) \ ?$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ∎

- $T(n) = 3T(n/4) + n \lg n$

$$n^{\log_b a} = n^{\log_4 3}$$

$$f(n) = n \lg n = \Omega(n^{\log_4 3 + \epsilon}) \ ?$$

$$a(n/b)\lg(n/b) \leq cn \lg n$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 3T(n/4) + n \lg n$

$$\begin{aligned} n^{\log_b a} &= n^{\log_4 3} \\ f(n) &= n \lg n = \Omega(n^{\log_4 3 + \epsilon}) \text{ ?} \\ a(n/b)\lg(n/b) &\leq cn \lg n \\ 3(n/4)\lg(n/4) &\leq (3/4)n \lg n \text{ , for } c = 3/4 \end{aligned}$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 3T(n/4) + n \lg n$

$$
\begin{aligned}
n^{\log_b a} &= n^{\log_4 3} \\
f(n) &= n \lg n = \Omega(n^{\log_4 3 + \epsilon}) \ ? \\
a(n/b)\lg(n/b) &\leq cn \lg n \\
3(n/4)\lg(n/4) &\leq (3/4)n \lg n \ , \ \text{for } c = 3/4 \\
T(n) &= \Theta(n \lg n)
\end{aligned}
$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 2T(n/2) + n \lg n$

$$n^{\log_b a} = n^{\log_2 2} = n$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 2T(n/2) + n \lg n$

$$n^{\log_b a} = n^{\log_2 2} = n$$

$$f(n) = n \lg n$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 2T(n/2) + n \lg n$

$$n^{\log_b a} = n^{\log_2 2} = n$$

$$f(n) = n \lg n = \Omega(n^{1+\epsilon}) \ ?$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 2T(n/2) + n \lg n$

$$n^{\log_b a} = n^{\log_2 2} = n$$

$$f(n) = n \lg n = \Omega(n^{1+\epsilon}) \ ?$$

Master method cannot be applied.

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 7T(n/2) + \Theta(n^2)$

$$n^{\log_b a} = n^{\lg 7} \approx n^{2.81}$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 7T(n/2) + \Theta(n^2)$

$$n^{\log_b a} = n^{\lg 7} \approx n^{2.81}$$

$$f(n) = \Theta(n^2)$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ∎

- $T(n) = 7T(n/2) + \Theta(n^2)$

$$n^{\log_b a} = n^{\lg 7} \approx n^{2.81}$$

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon})$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 7T(n/2) + \Theta(n^2)$

$$n^{\log_b a} = n^{\lg 7} \approx n^{2.81}$$

$$f(n) = \Theta(n^2) = O(n^{\log_b a - \epsilon})$$

$$= O(n^{\lg 7 - \epsilon}) \ ?$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 7T(n/2) + \Theta(n^2)$

$$
\begin{aligned}
n^{\log_b a} &= n^{\lg 7} \approx n^{2.81} \\
f(n) &= \Theta(n^2) = O(n^{\log_b a - \epsilon}) \\
&= O(n^{\lg 7 - \epsilon}) \ ? \\
T(n) &= \Theta(n^{\log_b a})
\end{aligned}
$$

$T(n) = aT(n/b) + f(n)$, $a \geq 1, b > 1$

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

- $T(n) = 7T(n/2) + \Theta(n^2)$

$$
\begin{aligned}
n^{\log_b a} &= n^{\lg 7} \approx n^{2.81} \\
f(n) &= \Theta(n^2) = O(n^{\log_b a - \epsilon}) \\
&= O(n^{\lg 7 - \epsilon}) \ ? \\
T(n) &= \Theta(n^{\log_b a}) \\
&= \Theta(n^{\lg 7})
\end{aligned}
$$

Proof of the master theorem

- Section 4.6 will not be a part of our syllabus

- Record : Collection of data

Part II Sorting and Order Statistics

- Record : Collection of data
- Key : Value to be sorted

Part II Sorting and Order Statistics

- Record : Collection of data
- Key : Value to be sorted
- Satellite data

Part II Sorting and Order Statistics

- Record : Collection of data
- Key : Value to be sorted
- Satellite data
- If satellite data is large, we permute an array of pointers to the records.

Sorting algorithms

- *In place* sorting : If at any time only a constant number of elements are stored outside the array.

Sorting algorithms

- *In place* sorting : If at any time only a constant number of elements are stored outside the array.
- MERGE procedure does not operate in place.

Sorting algorithms

- *In place* sorting : If at any time only a constant number of elements are stored outside the array.
- MERGE procedure does not operate in place.
- Ch 6 : Heapsort that uses a data structure called heap.

Sorting algorithms

- *In place* sorting : If at any time only a constant number of elements are stored outside the array.
- MERGE procedure does not operate in place.
- Ch 6 : Heapsort that uses a data structure called heap.
- Heapsort sorts n elements *in place* in $O(n \lg n)$ time.

Sorting algorithms

Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)

Ch 6 : Heapsort

- Heapsort uses the Heap data structure.

Ch 6 : Heapsort

- Heapsort uses the Heap data structure.
- To understand heap, we need to understand a few terminologies.

Binary tree

- Binary tree is a tree data structure where each node can have at most two child nodes : left child node and right child node.

Binary tree

- Binary tree is a tree data structure where each node can have at most two child nodes : left child node and right child node.