Divide and Conquer Recurrences

Aritra Hazra

Department of Computer Science and Engineering,
Indian Institute of Technology Kharagpur,
Paschim Medinipur, West Bengal, India - 721302.

Email: *aritrah@cse.iitkgp.ac.in*

Autumn 2020

Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

1. *Base Case.* Unit ($n < b$) problem instances solved in constant (c) steps – b and c are known constants

Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

1. *Base Case.* Unit ($n < b$) problem instances solved in constant (c) steps – b and c are known constants
2. *Decomposition.* Problem of n instances partitioned (top-down) into m sub-problems each with n_i instances – takes $d(n)$ steps

Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

1. *Base Case.* Unit ($n < b$) problem instances solved in constant (c) steps – b and c are known constants
2. *Decomposition.* Problem of n instances partitioned (top-down) into m sub-problems each with n_i instances – takes $d(n)$ steps
3. *Recursive Calls.* All m sub-problems are recursively solved – takes $T(n_i)$ steps for each sub-problem of n_i instances

Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

1. *Base Case.* Unit ($n < b$) problem instances solved in constant (c) steps – b and c are known constants
2. *Decomposition.* Problem of n instances partitioned (top-down) into m sub-problems each with n_i instances – takes $d(n)$ steps
3. *Recursive Calls.* All m sub-problems are recursively solved – takes $T(n_i)$ steps for each sub-problem of n_i instances
4. *Recomposition.* Solutions from sub-problems composed (bottom-up) to produce solution of the problem – takes $r(n)$ steps

Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

1. *Base Case.* Unit ($n < b$) problem instances solved in constant (c) steps – b and c are known constants
2. *Decomposition.* Problem of n instances partitioned (top-down) into m sub-problems each with n_i instances – takes $d(n)$ steps
3. *Recursive Calls.* All m sub-problems are recursively solved – takes $T(n_i)$ steps for each sub-problem of n_i instances
4. *Recomposition.* Solutions from sub-problems composed (bottom-up) to produce solution of the problem – takes $r(n)$ steps

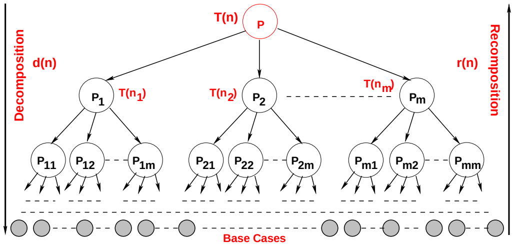
Recurrence Format: $$T(n) = \begin{cases} \left[T(n_1) + T(n_2) + \cdots + T(n_m)\right] + \left[d(n) + r(n)\right], & n > b \\ c, & n \leq b \end{cases}$$

Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

1. *Base Case.* Unit ($n < b$) problem instances solved in constant (c) steps – b and c are known constants
2. *Decomposition.* Problem of n instances partitioned (top-down) into m sub-problems each with n_i instances – takes $d(n)$ steps
3. *Recursive Calls.* All m sub-problems are recursively solved – takes $T(n_i)$ steps for each sub-problem of n_i instances
4. *Recomposition.* Solutions from sub-problems composed (bottom-up) to produce solution of the problem – takes $r(n)$ steps

Recurrence Format: $$T(n) = \begin{cases} \left[T(n_1) + T(n_2) + \cdots + T(n_m)\right] + \left[d(n) + r(n)\right], & n > b \\ c, & n \leq b \end{cases}$$

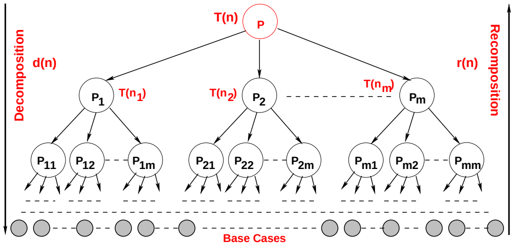


Recurrent Problem Solving: The Divide and Conquer Way

Recurrent Problem Solving: Process of solving problems involving sub-problems:

1. *Base Case.* Unit ($n < b$) problem instances solved in constant (c) steps – b and c are known constants
2. *Decomposition.* Problem of n instances partitioned (top-down) into m sub-problems each with n_i instances – takes $d(n)$ steps
3. *Recursive Calls.* All m sub-problems are recursively solved – takes $T(n_i)$ steps for each sub-problem of n_i instances
4. *Recomposition.* Solutions from sub-problems composed (bottom-up) to produce solution of the problem – takes $r(n)$ steps

Recurrence Format: $T(n) = \begin{cases} \left[T(n_1) + T(n_2) + \cdots + T(n_m)\right] + \left[d(n) + r(n)\right], & n > b \\ c, & n \leq b \end{cases}$



Formulation of Recurrence Relations and their Solutions depend on the Splitting and Composing Mechanisms!

Example-1: *Find Maximum among n Elements*

Strategy-1.1:

1. *Base Case.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Example-1: *Find Maximum among n Elements*

Strategy-1.1:

1. *Base Case.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_1(n) = \begin{cases} 2.T_1(\frac{n}{2}) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.1:

1. *Base Case.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_1(n) = \begin{cases} 2.T_1(\frac{n}{2}) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assume the existence of k, such that $n = 2^k$

$$\begin{aligned} T_1(n) &= 2.T_1\left(\frac{n}{2}\right) + 1 \quad = \quad 2^2.T_1\left(\frac{n}{2^2}\right) + 2 + 1 \\ &= 2^3.T_1\left(\frac{n}{2^3}\right) + 2^2 + 2 + 1 \quad = \quad \cdots\cdots \\ &= 2^k.T_1\left(\frac{n}{2^k}\right) + 2^{k-1} + 2^{k-2} + \cdots + 2^1 + 2^0 \\ &= 2^k.0 + 2^k - 1 \quad = \quad 2^k - 1 \quad = \quad n - 1 \end{aligned}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.2:

1. *Base Case.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Example-1: *Find Maximum among n Elements*

Strategy-1.2:

1. *Base Case.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_2(n) = \begin{cases} T_2(1) + T_2(n-1) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.2:

1. *Base Case.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_2(n) = \begin{cases} T_2(1) + T_2(n-1) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution:

$$\begin{aligned} T_2(n) &= T_2(1) + T_2(n-1) + 1 = T_2(n-1) + 1 \\ &= T_2(n-2) + 2 = T_2(n-3) + 3 = \cdots\cdots \\ &= T_2(1) + (n-1) = n - 1 \end{aligned}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.3:

1. *Base Cases.* If $n = 1$, Return that element as maximum
   If $n = 2$, Compare between these to get maximum
2. *Decomposition.* Split the set of elements into two parts having 2 elements and $(n - 2)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Example-1: *Find Maximum among n Elements*

Strategy-1.3:

1. *Base Cases.* If $n = 1$, Return that element as maximum
   If $n = 2$, Compare between these to get maximum
2. *Decomposition.* Split the set of elements into two parts having 2 elements and $(n-2)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_3(n) = \begin{cases} T_3(2) + T_3(n-2) + 1, & \text{if } n > 2 \\ 1, & \text{if } n = 2 \\ 0, & \text{if } n = 1 \end{cases}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.3:

1. *Base Cases.* If $n = 1$, Return that element as maximum
   If $n = 2$, Compare between these to get maximum
2. *Decomposition.* Split the set of elements into two parts having 2 elements and $(n-2)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_3(n) = \begin{cases} T_3(2) + T_3(n-2) + 1, & \text{if } n > 2 \\ 1, & \text{if } n = 2 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution:

$$\begin{aligned} T_3(n) &= T_3(2) + T_3(n-2) + 1 = T_3(n-2) + 2 \\ &= T_3(n-4) + 4 = T_3(n-6) + 6 = \cdots\cdots \\ &= \begin{cases} T_3(2) + (n-2) & \text{if } n \text{ is even} \\ T_3(1) + (n-1) & \text{if } n \text{ is odd} \end{cases} = n - 1 \end{aligned}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \leq c \leq n-1)$ is equally likely, the average number of comparisons, $T_4(n) = (\frac{1}{n-1}). \sum_{i=1}^{n-1} [T_4(i) + T_4(n-i) + 1]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i) + (n-1)$

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \leq c \leq n-1)$ is equally likely, the average number of comparisons, $T_4(n) = (\frac{1}{n-1}).\sum_{i=1}^{n-1}[T_4(i) + T_4(n-i) + 1]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i) + (n-1)$

Similarly, $(n-2).T_4(n-1) = 2.\sum_{i=1}^{n-2} T_4(i) + (n-2)$ [Put, $n \leftarrow n-1$]

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \leq c \leq n-1)$ is equally likely, the average number of comparisons, $T_4(n) = (\frac{1}{n-1}).\sum_{i=1}^{n-1}[T_4(i) + T_4(n-i) + 1]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i) + (n-1)$

Similarly, $(n-2).T_4(n-1) = 2.\sum_{i=1}^{n-2} T_4(i) + (n-2)$ [Put, $n \leftarrow n-1$]

Subtracting, we get, $(n-1).T_4(n) - (n-2).T_4(n-1) = 2.T_4(n-1) + 1$

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \le c \le n - 1)$ is equally likely, the average number of comparisons, $T_4(n) = (\frac{1}{n-1}). \sum_{i=1}^{n-1} [T_4(i) + T_4(n-i) + 1]$

implies, $(n-1). T_4(n) = 2. \sum_{i=1}^{n-1} T_4(i) + (n-1)$

Similarly, $(n-2). T_4(n-1) = 2. \sum_{i=1}^{n-2} T_4(i) + (n-2)$ [Put, $n \leftarrow n-1$]

Subtracting, we get, $(n-1). T_4(n) - (n-2). T_4(n-1) = 2. T_4(n-1) + 1$

$\therefore \frac{T_4(n)}{n} - \frac{T_4(n-1)}{n-1} = \frac{1}{n-1} - \frac{1}{n}$

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \le c \le n-1)$ is equally likely, the average number of comparisons, $T_4(n) = \left(\frac{1}{n-1}\right).\sum_{i=1}^{n-1}[T_4(i) + T_4(n-i) + 1]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i) + (n-1)$

Similarly, $(n-2).T_4(n-1) = 2.\sum_{i=1}^{n-2} T_4(i) + (n-2)$ [Put, $n \leftarrow n-1$]

Subtracting, we get, $(n-1).T_4(n) - (n-2).T_4(n-1) = 2.T_4(n-1) + 1$

$$\therefore \frac{T_4(n)}{n} - \frac{T_4(n-1)}{n-1} = \frac{1}{n-1} - \frac{1}{n}$$

$$\frac{T_4(n-1)}{n-1} - \frac{T_4(n-2)}{n-2} = \frac{1}{n-2} - \frac{1}{n-1}$$

$$\cdots\cdots \qquad \cdots\cdots$$

$$\frac{T_4(3)}{3} - \frac{T_4(2)}{2} = \frac{1}{2} - \frac{1}{3}$$

$$\frac{T_4(2)}{2} - \frac{T_4(1)}{1} = \frac{1}{1} - \frac{1}{2}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \leq c \leq n-1)$ is equally likely, the average number of comparisons, $T_4(n) = (\frac{1}{n-1}). \sum_{i=1}^{n-1} [T_4(i) + T_4(n-i) + 1]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i) + (n-1)$

Similarly, $(n-2).T_4(n-1) = 2.\sum_{i=1}^{n-2} T_4(i) + (n-2)$ [Put, $n \leftarrow n-1$]

Subtracting, we get, $(n-1).T_4(n) - (n-2).T_4(n-1) = 2.T_4(n-1) + 1$

$$\therefore \frac{T_4(n)}{n} - \frac{T_4(n-1)}{n-1} = \frac{1}{n-1} - \frac{1}{n}$$

$$\frac{T_4(n-1)}{n-1} - \frac{T_4(n-2)}{n-2} = \frac{1}{n-2} - \frac{1}{n-1}$$

$$\cdots\cdots \qquad \cdots\cdots$$

$$\frac{T_4(3)}{3} - \frac{T_4(2)}{2} = \frac{1}{2} - \frac{1}{3}$$

$$\frac{T_4(2)}{2} - \frac{T_4(1)}{1} = \frac{1}{1} - \frac{1}{2}$$

Adding all these equations, we get,

$$\frac{T_4(n)}{n} - \frac{T_4(1)}{1} = 1 - \frac{1}{n}$$

Example-1: *Find Maximum among n Elements*

Strategy-1.4:

1. *Base Cases.* If $n = 1$, Return that element as maximum
2. *Decomposition.* Split the set of elements into two parts having c elements and $(n - c)$ elements in respective parts
3. *Recursion.* Select maximum element from both parts
4. *Recomposition.* Compare both maximum to find largest

Recurrence: Number of comparison required to find maximum element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c) + 1, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \leq c \leq n-1)$ is equally likely, the average number of comparisons, $T_4(n) = (\frac{1}{n-1}). \sum_{i=1}^{n-1} [T_4(i) + T_4(n-i) + 1]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i) + (n-1)$

Similarly, $(n-2).T_4(n-1) = 2.\sum_{i=1}^{n-2} T_4(i) + (n-2)$ [Put, $n \leftarrow n-1$]

Subtracting, we get, $(n-1).T_4(n) - (n-2).T_4(n-1) = 2.T_4(n-1) + 1$

$$\therefore \frac{T_4(n)}{n} - \frac{T_4(n-1)}{n-1} = \frac{1}{n-1} - \frac{1}{n}$$
$$\frac{T_4(n-1)}{n-1} - \frac{T_4(n-2)}{n-2} = \frac{1}{n-2} - \frac{1}{n-1}$$
$$\cdots\cdots \qquad \cdots\cdots$$
$$\frac{T_4(3)}{3} - \frac{T_4(2)}{2} = \frac{1}{2} - \frac{1}{3}$$
$$\frac{T_4(2)}{2} - \frac{T_4(1)}{1} = \frac{1}{1} - \frac{1}{2}$$

Adding all these equations, we get,

$$\frac{T_4(n)}{n} - \frac{T_4(1)}{1} = 1 - \frac{1}{n}$$

$$\Rightarrow T_4(n) = n - 1$$

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.1:

1. *Base Case.* If $n = 1$, Return that element as max & min
   If $n = 2$, Compare between these to get max & min
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.1:

1. *Base Case.* If $n = 1$, Return that element as max & min
   If $n = 2$, Compare between these to get max & min
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Recurrence: Number of comparison required to find max & min elements,

$$T_1(n) = \begin{cases} 2.T_1(\frac{n}{2}) + 2, & \text{if } n > 2 \\ 1, & \text{if } n = 2 \end{cases}$$

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.1:

1. *Base Case.* If $n = 1$, Return that element as max & min
   If $n = 2$, Compare between these to get max & min
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Recurrence: Number of comparison required to find max & min elements,

$$T_1(n) = \begin{cases} 2.T_1(\frac{n}{2}) + 2, & \text{if } n > 2 \\ 1, & \text{if } n = 2 \end{cases}$$

Solution: Assume the existence of k, such that $n = 2^k$

$$\begin{aligned} T_1(n) &= 2.T_1\left(\frac{n}{2}\right) + 2 \quad = \quad 2^2.T_1\left(\frac{n}{2^2}\right) + 2^2 + 2 \\ &= 2^3.T_1\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2 \quad = \quad \cdots\cdots \\ &= 2^{k-1}.T_1\left(\frac{n}{2^{k-1}}\right) + 2^{k-1} + 2^{k-2} + \cdots + 2^2 + 2^1 \\ &= 2^{k-1} + 2^k - 2 \quad = \quad \frac{3}{2}.2^k - 2 \quad = \quad \frac{3}{2}.n - 2 \end{aligned}$$

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.2:

1. *Base Case.* If $n = 1$, Return that element as max & min
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.2:

1. *Base Case.* If $n = 1$, Return that element as max & min
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Recurrence: Number of comparison required to find max & min elements,

$$T_2(n) = \begin{cases} T_2(1) + T_2(n-1) + 2, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.2:

1. *Base Case.* If $n = 1$, Return that element as max & min
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Recurrence: Number of comparison required to find max & min elements,

$$T_2(n) = \begin{cases} T_2(1) + T_2(n-1) + 2, & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution:

$$\begin{aligned} T_2(n) &= T_2(1) + T_2(n-1) + 2 = T_2(n-1) + 2 \\ &= T_2(n-2) + 4 = T_2(n-3) + 6 = \cdots\cdots \\ &= T_2(1) + 2(n-1) = 2n - 2 \end{aligned}$$

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.3:

1. *Base Case.* If $n = 1$, Return that element as max & min
   If $n = 2$, Compare in between to get max & min
2. *Decomposition.* Split the set of elements into two parts having 2 elements and $(n - 2)$ elements in respective parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.3:

1. *Base Case.* If $n = 1$, Return that element as max & min
   If $n = 2$, Compare in between to get max & min
2. *Decomposition.* Split the set of elements into two parts having 2 elements and $(n-2)$ elements in respective parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Recurrence: Number of comparison required to find max & min elements,

$$T_3(n) = \begin{cases} T_3(2) + T_3(n-2) + 2, & \text{if } n > 2 \\ 1, & \text{if } n = 2 \\ 0, & \text{if } n = 1 \end{cases}$$

Example-2: *Find Max. & Min. (both) among n Elements*

Strategy-2.3:

1. *Base Case.* If $n = 1$, Return that element as max & min
   If $n = 2$, Compare in between to get max & min
2. *Decomposition.* Split the set of elements into two parts having 2 elements and $(n-2)$ elements in respective parts
3. *Recursion.* Select max & min elements from both parts
4. *Recomposition.* Compare both max to find largest
   Compare both min to find smallest

Recurrence: Number of comparison required to find max & min elements,

$$T_3(n) = \begin{cases} T_3(2) + T_3(n-2) + 2, & \text{if } n > 2 \\ 1, & \text{if } n = 2 \\ 0, & \text{if } n = 1 \end{cases}$$

Solution: Let, $2m = n - 2$ (if n is even) or $2m = n - 1$ (if n is odd)

$$\begin{aligned} T_3(n) &= T_3(2) + T_3(n-2) + 2 = T_3(n-2) + 3 \\ &= T_3(n-4) + 6 = T_3(n-6) + 9 = \cdots\cdots \\ &= \begin{cases} T_3(2) + 3m = 1 + \frac{3}{2}(n-2) = \frac{3}{2}.n - 2, & \text{if } n \text{ is even} \\ T_3(1) + 3m = 0 + \frac{3}{2}(n-1) = \frac{3}{2}.n - \frac{3}{2}, & \text{if } n \text{ is odd} \end{cases} \end{aligned}$$

Example-3: *Search an Element within n Elements*

Strategy-3.1:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Example-3: *Search an Element within n Elements*

Strategy-3.1:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_1(n) = \begin{cases} 2.T_1(\frac{n}{2}), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Example-3: *Search an Element within n Elements*

Strategy-3.1:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two equal parts
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_1(n) = \begin{cases} 2.T_1(\frac{n}{2}), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assume the existence of k, such that $n = 2^k$

$$\begin{aligned} T_1(n) &= 2.T_1\left(\frac{n}{2}\right) = 2^2.T_1\left(\frac{n}{2^2}\right) = \cdots\cdots \\ &= 2^k.T_1\left(\frac{n}{2^k}\right) = 2^k = n \end{aligned}$$

Example-3: *Search an Element within n Elements*

Strategy-3.2:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (fractional) parts (say, $\frac{1}{3}$ elements in left and $\frac{2}{3}$ elements in right)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Example-3: *Search an Element within n Elements*

Strategy-3.2:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (fractional) parts (say, $\frac{1}{3}$ elements in left and $\frac{2}{3}$ elements in right)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_3(n) = \begin{cases} T_3(\frac{n}{3}) + T_3(\frac{2n}{3}), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Example-3: *Search an Element within n Elements*

Strategy-3.2:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (fractional) parts (say, $\frac{1}{3}$ elements in left and $\frac{2}{3}$ elements in right)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_3(n) = \begin{cases} T_3(\frac{n}{3}) + T_3(\frac{2n}{3}), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Using strong mathematical induction, we can prove that (assume $T_3(k) = ak + b$ as induction hypothesis for all $k < n$), $T_3(1) = 1$ (Base Case satisfied for all $a = 1 - b$) and $T_3(n) = \frac{an+b}{3} + \frac{2(an+b)}{3} = an + b$.

Example-3: *Search an Element within n Elements*

Strategy-3.2:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (fractional) parts (say, $\frac{1}{3}$ elements in left and $\frac{2}{3}$ elements in right)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_3(n) = \begin{cases} T_3(\frac{n}{3}) + T_3(\frac{2n}{3}), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Using strong mathematical induction, we can prove that (assume $T_3(k) = ak + b$ as induction hypothesis for all $k < n$), $T_3(1) = 1$ (Base Case satisfied for all $a = 1 - b$) and $T_3(n) = \frac{an+b}{3} + \frac{2(an+b)}{3} = an + b$.
It may be noted that,

$$\begin{aligned} T_3(n) &= T_3\left(\frac{n}{3}\right) + T_3\left(\frac{2n}{3}\right) = T_3\left(\frac{n}{3^2}\right) + T_3\left(\frac{2n}{3^2}\right) + T_3\left(\frac{2n}{3^2}\right) + T_3\left(\frac{4n}{3^2}\right) \\ &= T_3\left(\frac{n}{3^2}\right) + 2T_3\left(\frac{2n}{3^2}\right) + T_3\left(\frac{4n}{3^2}\right) \\ &= \binom{3}{0}.T_3\left(\frac{n}{3^3}\right) + \binom{3}{1}.T_3\left(\frac{2n}{3^3}\right) + \binom{3}{2}.T_3\left(\frac{4n}{3^3}\right) + \binom{3}{3}.T_3\left(\frac{8n}{3^3}\right) \\ &= \cdots\cdots = \sum_{i=0}^{k} \binom{k}{i}.T\left(\frac{2^i.n}{3^k}\right) \end{aligned}$$

Example-3: *Search an Element within n Elements*

Strategy-3.3:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n - 1)$ elements in respective parts
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Example-3: *Search an Element within n Elements*

Strategy-3.3:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_2(n) = \begin{cases} T_2(1) + T_2(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Example-3: *Search an Element within n Elements*

Strategy-3.3:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two parts having 1 element and $(n-1)$ elements in respective parts
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_2(n) = \begin{cases} T_2(1) + T_2(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: [known as Linear Search]

$$\begin{aligned} T_2(n) &= T_2(1) + T_2(n-1) = T_2(n-1) + 1 \\ &= T_2(n-2) + 2 = T_2(n-3) + 3 = \cdots\cdots \\ &= T_2(1) + (n-1) = n \end{aligned}$$

Example-3: *Search an Element within n Elements*

Strategy-3.4:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (constant-depth) parts (say, c elements in left and $(n - c)$ elements in right), for an arbitrary constant (c)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Example-3: *Search an Element within n Elements*

Strategy-3.4:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (constant-depth) parts (say, c elements in left and $(n - c)$ elements in right), for an arbitrary constant (c)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n - c), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Example-3: *Search an Element within n Elements*

Strategy-3.4:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (constant-depth) parts (say, c elements in left and $(n - c)$ elements in right), for an arbitrary constant (c)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n - c), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \le c \le n - 1)$ is equally likely, the average number of probes, $T_4(n) = (\frac{1}{n-1}). \sum_{i=1}^{n-1} [T_4(i) + T_4(n - i)]$

implies, $(n - 1). T_4(n) = 2. \sum_{i=1}^{n-1} T_4(i)$

Example-3: *Search an Element within n Elements*

Strategy-3.4:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (constant-depth) parts (say, c elements in left and $(n - c)$ elements in right), for an arbitrary constant (c)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \leq c \leq n-1)$ is equally likely, the average number of probes, $T_4(n) = (\frac{1}{n-1}). \sum\limits_{i=1}^{n-1} [T_4(i) + T_4(n-i)]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i)$

Similarly, $(n-2).T_4(n-1) = 2.\sum_{i=1}^{n-2} T_4(i)$ [Putting, $n \leftarrow n-1$]

Example-3: *Search an Element within n Elements*

Strategy-3.4:

1. *Base Case.* If $n = 1$, Compare and Return found / not-found
2. *Decomposition.* Split the set of elements into two unequal (constant-depth) parts (say, c elements in left and $(n - c)$ elements in right), for an arbitrary constant (c)
3. *Recursion.* Search the element from both parts
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of comparison required to search/find an element,

$$T_4(n) = \begin{cases} T_4(c) + T_4(n-c), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assuming the choice of constant c $(1 \leq c \leq n-1)$ is equally likely, the average number of probes, $T_4(n) = \left(\frac{1}{n-1}\right).\sum_{i=1}^{n-1}[T_4(i) + T_4(n-i)]$

implies, $(n-1).T_4(n) = 2.\sum_{i=1}^{n-1} T_4(i)$

Similarly, $(n-2).T_4(n-1) = 2.\sum_{i=1}^{n-2} T_4(i)$ [Putting, $n \leftarrow n-1$]

Subtracting, we get, $(n-1).T_4(n) - (n-2).T_4(n-1) = 2.T_4(n-1)$

$\Rightarrow T_4(n) = \left(\frac{n}{n-1}\right).T_4(n-1) = \left(\frac{n}{n-1}\right).\left(\frac{n-1}{n-2}\right).T_4(n-2) = \cdots = n.T(1) = n$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.1:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at middle and Return found if matches
   Otherwise, Split the set of elements into two equal parts
3. *Recursion.* If query-element is lesser (or greater) than the middle element, Search the elements from left (or right) part
4. *Recomposition.* Return found if query-element found in any part

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.1:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at middle and Return found if matches
   Otherwise, Split the set of elements into two equal parts
3. *Recursion.* If query-element is lesser (or greater) than the middle element, Search the elements from left (or right) part
4. *Recomposition.* Return found if query-element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_1(n) = \begin{cases} T_1(\frac{n}{2}) + 1, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.1:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at middle and Return found if matches
   Otherwise, Split the set of elements into two equal parts
3. *Recursion.* If query-element is lesser (or greater) than the middle element, Search the elements from left (or right) part
4. *Recomposition.* Return found if query-element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_1(n) = \begin{cases} T_1(\frac{n}{2}) + 1, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assume the existence of k, such that $n = 2^k$

$$\begin{aligned} T_1(n) &= T_1\left(\frac{n}{2}\right) + 1 = T_1\left(\frac{n}{2^2}\right) + 2 = T_1\left(\frac{n}{2^3}\right) + 3 \\ &= \cdots\cdots = T_1\left(\frac{n}{2^k}\right) + k = 1 + k = 1 + \log_2 n \end{aligned}$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.2:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (fractional) position (say, $\frac{1}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $\frac{1}{3}$ elements in left part and $\frac{2}{3}$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the $\frac{1}{3}$rd element, Search the elements from left (or right) part
4. *Recomposition.* Return found if query-element found in any part

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.2:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (fractional) position (say, $\frac{1}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $\frac{1}{3}$ elements in left part and $\frac{2}{3}$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the $\frac{1}{3}$rd element, Search the elements from left (or right) part
4. *Recomposition.* Return found if query-element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_2(n) = \begin{cases} T_2(\frac{2n}{3}) + 1, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.2:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (fractional) position (say, $\frac{1}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $\frac{1}{3}$ elements in left part and $\frac{2}{3}$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the $\frac{1}{3}$rd element, Search the elements from left (or right) part
4. *Recomposition.* Return found if query-element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_2(n) = \begin{cases} T_2(\frac{2n}{3}) + 1, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assume the existence of k, such that $n = \left(\frac{3}{2}\right)^k$

$$T_2(n) = T_2\left(\frac{2n}{3}\right) + 1 = T_2\left(\frac{n}{(\frac{3}{2})^2}\right) + 2 = T_2\left(\frac{n}{(\frac{3}{2})^3}\right) + 3$$

$$= \cdots\cdots = T_2\left(\frac{n}{(\frac{3}{2})^k}\right) + k = 1 + k = 1 + \log_{\frac{3}{2}} n$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.2:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (fractional) position (say, $\frac{1}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $\frac{1}{3}$ elements in left part and $\frac{2}{3}$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the $\frac{1}{3}$rd element, Search the elements from left (or right) part
4. *Recomposition.* Return found if query-element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_2(n) = \begin{cases} T_2(\frac{2n}{3}) + 1, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assume the existence of k, such that $n = \left(\frac{3}{2}\right)^k$

$$\begin{aligned} T_2(n) &= T_2\left(\frac{2n}{3}\right) + 1 = T_2\left(\frac{n}{(\frac{3}{2})^2}\right) + 2 = T_2\left(\frac{n}{(\frac{3}{2})^3}\right) + 3 \\ &= \cdots\cdots = T_2\left(\frac{n}{(\frac{3}{2})^k}\right) + k = 1 + k = 1 + \log_{\frac{3}{2}} n \end{aligned}$$

Generalized Form: For αn and $(1-\alpha)n$ splits ($\frac{1}{2} < \alpha < 1$), $T_2(n) = 1 + \log_{\frac{1}{\alpha}} n$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.3:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at two arbitrary (fractional) positions (say, $\frac{1}{3}$rd and $\frac{2}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into three equal parts (i.e., $\frac{1}{3}$ elements in each of left, middle and right parts)
3. *Recursion.* If query-element is lesser than $\frac{1}{3}$rd (or greater than $\frac{2}{3}$rd) element, Search the element from left (or right) part.
   Otherwise, search the element from middle part.
4. *Recomposition.* Return found if element found in any part

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.3:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at two arbitrary (fractional) positions (say, $\frac{1}{3}$rd and $\frac{2}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into three equal parts (i.e., $\frac{1}{3}$ elements in each of left, middle and right parts)
3. *Recursion.* If query-element is lesser than $\frac{1}{3}$rd (or greater than $\frac{2}{3}$rd) element, Search the element from left (or right) part.
   Otherwise, search the element from middle part.
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_3(n) = \begin{cases} T_3(\frac{n}{3}) + 2, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.3:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at two arbitrary (fractional) positions (say, $\frac{1}{3}$rd and $\frac{2}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into three equal parts (i.e., $\frac{1}{3}$ elements in each of left, middle and right parts)
3. *Recursion.* If query-element is lesser than $\frac{1}{3}$rd (or greater than $\frac{2}{3}$rd) element, Search the element from left (or right) part.
   Otherwise, search the element from middle part.
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_3(n) = \begin{cases} T_3(\frac{n}{3}) + 2, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assume the existence of k, such that $n = 3^k$

$$\begin{aligned} T_3(n) &= T_3\left(\frac{n}{3}\right) + 2 = T_3\left(\frac{n}{3^2}\right) + 4 = T_3\left(\frac{n}{3^3}\right) + 6 \\ &= \cdots\cdots = T_3\left(\frac{n}{3^k}\right) + 2.k = 1 + 2.k = 1 + 2\log_3 n \end{aligned}$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.3:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at two arbitrary (fractional) positions (say, $\frac{1}{3}$rd and $\frac{2}{3}$rd) and Return found if matches
   Otherwise, Split the set of elements into three equal parts (i.e., $\frac{1}{3}$ elements in each of left, middle and right parts)
3. *Recursion.* If query-element is lesser than $\frac{1}{3}$rd (or greater than $\frac{2}{3}$rd) element, Search the element from left (or right) part.
   Otherwise, search the element from middle part.
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element,

$$T_3(n) = \begin{cases} T_3(\frac{n}{3}) + 2, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

Solution: Assume the existence of k, such that $n = 3^k$

$$\begin{aligned} T_3(n) &= T_3\left(\frac{n}{3}\right) + 2 = T_3\left(\frac{n}{3^2}\right) + 4 = T_3\left(\frac{n}{3^3}\right) + 6 \\ &= \cdots\cdots = T_3\left(\frac{n}{3^k}\right) + 2.k = 1 + 2.k = 1 + 2\log_3 n \end{aligned}$$

Generalized Form: For β equal-sized splits $(2 \leq \beta \leq n)$, $T_2(n) = 1 + (\beta - 1)\log_\beta n$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.4:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (constant-depth) positions (say, a constant c^{th} element) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $(c-1)$ elements in left part and $(n-c)$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the c^{th} element, Search the element from left (or right) part
4. *Recomposition.* Return found if element found in any part

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.4:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (constant-depth) positions (say, a constant c^{th} element) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $(c-1)$ elements in left part and $(n-c)$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the c^{th} element, Search the element from left (or right) part
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element (let $c < \frac{n}{2}$),

$$T_4(n) = \begin{cases} T_4(n-c) + 1, & \text{if } n > c \\ n, & \text{if } 1 \leq n \leq c \end{cases}$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.4:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (constant-depth) positions (say, a constant c^{th} element) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $(c-1)$ elements in left part and $(n-c)$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the c^{th} element, Search the element from left (or right) part
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element (let $c < \frac{n}{2}$),

$$T_4(n) = \begin{cases} T_4(n-c)+1, & \text{if } n > c \\ n, & \text{if } 1 \leq n \leq c \end{cases}$$

Solution:

$$T_4(n) = T_4(n-c)+1 = T_4(n-2c)+2 = \cdots \leq T_4(c)+\frac{n-c}{c} = \left(\frac{1}{c}\right).n + (c-1)$$

$$T_4(n) = T_4(n-c)+1 = T_4(n-2c)+2 = \cdots \geq T_4(1)+\frac{n-1}{c} = \left(\frac{1}{c}\right).n + \frac{c-1}{c}$$

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.4:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (constant-depth) positions (say, a constant c^{th} element) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $(c-1)$ elements in left part and $(n-c)$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the c^{th} element, Search the element from left (or right) part
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element (let $c < \frac{n}{2}$),

$$T_4(n) = \begin{cases} T_4(n-c) + 1, & \text{if } n > c \\ n, & \text{if } 1 \leq n \leq c \end{cases}$$

Solution: $T_4(n) = T_4(n-c)+1 = T_4(n-2c)+2 = \cdots \leq T_4(c)+\frac{n-c}{c} = \left(\frac{1}{c}\right).n + (c-1)$

$T_4(n) = T_4(n-c)+1 = T_4(n-2c)+2 = \cdots \geq T_4(1)+\frac{n-1}{c} = \left(\frac{1}{c}\right).n + \frac{c-1}{c}$

[Caution] It can be as bad as linear search (if $c = 1$ is chosen)

Example-4: *Binary Search from n (Sorted) Elements*

Strategy-4.4:

1. *Base Case.* If $n = 1$, Probe and Return found / not-found
2. *Decomposition.* Probe at arbitrary (constant-depth) positions (say, a constant c^{th} element) and Return found if matches
   Otherwise, Split the set of elements into two unequal parts (i.e., $(c-1)$ elements in left part and $(n-c)$ elements in right part)
3. *Recursion.* If query-element is lesser (or greater) than the c^{th} element, Search the element from left (or right) part
4. *Recomposition.* Return found if element found in any part

Recurrence: Number of probes (assume each probe can decide whether $<, =, >$) required to search/find an element (let $c < \frac{n}{2}$),

$$T_4(n) = \begin{cases} T_4(n-c)+1, & \text{if } n > c \\ n, & \text{if } 1 \leq n \leq c \end{cases}$$

Solution: $T_4(n) = T_4(n-c)+1 = T_4(n-2c)+2 = \cdots \leq T_4(c) + \frac{n-c}{c} = \left(\frac{1}{c}\right).n + (c-1)$

$T_4(n) = T_4(n-c)+1 = T_4(n-2c)+2 = \cdots \geq T_4(1) + \frac{n-1}{c} = \left(\frac{1}{c}\right).n + \frac{c-1}{c}$

[Caution] It can be as bad as linear search (if $c = 1$ is chosen)

Insights from Recurrence Relations: *Why Binary Search needs to Split at Middle?*

Since, $\log_2 n \leq \log_{\frac{3}{2}} n$ [*i.e.* $\log_{\frac{1}{\alpha}} n$] and $\log_2 n \leq 2.\log_3 n$ [*i.e.* $(\beta - 1)\log_\beta n$],

Therefore, $T_1(n) \leq T_2(n)$ and $T_1(n) \leq T_3(n)$. Also, $T_1(n) \leq T_4(n)$

(implying lowest number of probes when splitting at middle position)

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.1A:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Find `max` element and $\mathcal{S}' \leftarrow \mathcal{S} - \{\texttt{max}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-1)$ elements
4. *Recomposition.* Return `max` followed by sorted elements of $\mathcal{S}'$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.1A:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Find max element and $\mathcal{S}' \leftarrow \mathcal{S} - \{\texttt{max}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-1)$ elements
4. *Recomposition.* Return max followed by sorted elements of $\mathcal{S}'$

Recurrence: Number of element comparisons done for sorting, [Selection Sort]

$$T(n) = \begin{cases} T(n-1) + (n-1), & \text{if } n > 1 \\ 0, & n = 1 \end{cases}$$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.1A:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Find max element and $\mathcal{S}' \leftarrow \mathcal{S} - \{\texttt{max}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-1)$ elements
4. *Recomposition.* Return max followed by sorted elements of $\mathcal{S}'$

Recurrence: Number of element comparisons done for sorting, [Selection Sort]

$$T(n) = \begin{cases} T(n-1) + (n-1), & \text{if } n > 1 \\ 0, & n = 1 \end{cases}$$

Solution:

$$T(n) = T(n-1) + (n-1) = T(n-2) + (n-2) + (n-1)$$
$$= \cdots = T(1) + 1 + 2 + \cdots + (n-1) = \frac{1}{2}.n^2 - \frac{1}{2}.n$$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.1A:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Find $\texttt{max}$ element and $\mathcal{S}' \leftarrow \mathcal{S} - \{\texttt{max}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-1)$ elements
4. *Recomposition.* Return $\texttt{max}$ followed by sorted elements of $\mathcal{S}'$

Recurrence: Number of element comparisons done for sorting, [Selection Sort]

$$T(n) = \begin{cases} T(n-1) + (n-1), & \text{if } n > 1 \\ 0, & n = 1 \end{cases}$$

Solution:

$$T(n) = T(n-1) + (n-1) = T(n-2) + (n-2) + (n-1)$$
$$= \cdots = T(1) + 1 + 2 + \cdots + (n-1) = \frac{1}{2}.n^2 - \frac{1}{2}.n$$

Strategy-5.1B:

1. *Base Case.* If $n = 2$ Return $\texttt{max}$ followed by $\texttt{min}$ elements
2. *Decomposition.* Find $\langle \texttt{max}, \texttt{min} \rangle$ elements and $\mathcal{S}' \leftarrow \mathcal{S} - \{\texttt{max}, \texttt{min}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-2)$ elements
4. *Recomposition.* Return $\langle \texttt{max}$, sorted elements of $\mathcal{S}'$, $\texttt{min} \rangle$ in order

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.1A:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Find max element and $\mathcal{S}' \leftarrow \mathcal{S} - \{\mathtt{max}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-1)$ elements
4. *Recomposition.* Return max followed by sorted elements of $\mathcal{S}'$

Recurrence: Number of element comparisons done for sorting, [Selection Sort]

$$T(n) = \begin{cases} T(n-1) + (n-1), & \text{if } n > 1 \\ 0, & n = 1 \end{cases}$$

Solution: $T(n) = T(n-1) + (n-1) = T(n-2) + (n-2) + (n-1)$
$= \cdots = T(1) + 1 + 2 + \cdots + (n-1) = \frac{1}{2}.n^2 - \frac{1}{2}.n$

Strategy-5.1B:

1. *Base Case.* If $n = 2$ Return max followed by min elements
2. *Decomposition.* Find $\langle \mathtt{max}, \mathtt{min} \rangle$ elements and $\mathcal{S}' \leftarrow \mathcal{S} - \{\mathtt{max}, \mathtt{min}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-2)$ elements
4. *Recomposition.* Return $\langle$max, sorted elements of $\mathcal{S}'$, min$\rangle$ in order

Recurrence: Number of element comparisons done for sorting (assuming n as even),

$$T(n) = \begin{cases} T(n-2) + (\frac{3}{2}.n - 1), & \text{if } n > 2 \\ 1, & n = 2 \end{cases}$$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.1A:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Find max element and $\mathcal{S}' \leftarrow \mathcal{S} - \{\texttt{max}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-1)$ elements
4. *Recomposition.* Return max followed by sorted elements of $\mathcal{S}'$

Recurrence: Number of element comparisons done for sorting, [Selection Sort]

$$T(n) = \begin{cases} T(n-1) + (n-1), & \text{if } n > 1 \\ 0, & n = 1 \end{cases}$$

Solution: $T(n) = T(n-1) + (n-1) = T(n-2) + (n-2) + (n-1)$
$= \cdots = T(1) + 1 + 2 + \cdots + (n-1) = \frac{1}{2}.n^2 - \frac{1}{2}.n$

Strategy-5.1B:

1. *Base Case.* If $n = 2$ Return max followed by min elements
2. *Decomposition.* Find $\langle \texttt{max}, \texttt{min} \rangle$ elements and $\mathcal{S}' \leftarrow \mathcal{S} - \{\texttt{max}, \texttt{min}\}$
3. *Recursion.* Sort $\mathcal{S}'$ with $(n-2)$ elements
4. *Recomposition.* Return $\langle$max, sorted elements of $\mathcal{S}'$, min$\rangle$ in order

Recurrence: Number of element comparisons done for sorting (assuming n as even),

$$T(n) = \begin{cases} T(n-2) + (\frac{3}{2}.n - 1), & \text{if } n > 2 \\ 1, & n = 2 \end{cases}$$

Solution: $T(n) = T(n-2) + (\frac{3}{2}.n - 1) = T(n-4) + \frac{3}{2}.[(n-2) + n] - 2 = \cdots$
$= T(2) + \frac{3}{2}.[4 + 6 + \cdots + (n-1)] - \frac{n-2}{2} = \frac{3}{8}.n^2 - \frac{1}{2}.n - \frac{11}{8}$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.2:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Split $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1$ and $\mathcal{S}_2$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* **Combine** sorted elements of $\mathcal{S}_1$ with $\mathcal{S}_2$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.2:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Split $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1$ and $\mathcal{S}_2$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* **Combine** sorted elements of $\mathcal{S}_1$ with $\mathcal{S}_2$

Combine-Step:

1. If $\mathcal{S}_1$ (or $\mathcal{S}_2$) is empty, Return elements of $\mathcal{S}_2$ (or $\mathcal{S}_1$)
2. Compare first elements, $a_1 \in \mathcal{S}_1$ with $b_1 \in \mathcal{S}_2$
3. If $a_1 \geq b_1$, Return a_1 followed by *combined sorted elements of* $\mathcal{S}_1 - \{a_1\}$ *with* $\mathcal{S}_2$. Otherwise, Return b_1 followed by *combined sorted elements of* $\mathcal{S}_1$ *with* $\mathcal{S}_2 - \{b_1\}$.

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.2:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Split $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1$ and $\mathcal{S}_2$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* **Combine** sorted elements of $\mathcal{S}_1$ with $\mathcal{S}_2$

Combine-Step:

1. If $\mathcal{S}_1$ (or $\mathcal{S}_2$) is empty, Return elements of $\mathcal{S}_2$ (or $\mathcal{S}_1$)
2. Compare first elements, $a_1 \in \mathcal{S}_1$ with $b_1 \in \mathcal{S}_2$
3. If $a_1 \geq b_1$, Return a_1 followed by *combined sorted elements of* $\mathcal{S}_1 - \{a_1\}$ *with* $\mathcal{S}_2$. Otherwise, Return b_1 followed by *combined sorted elements of* $\mathcal{S}_1$ *with* $\mathcal{S}_2 - \{b_1\}$.

Recurrence: Number of comparisons done for combining, [Merge]

$$T_C(j, n-j) = \begin{cases} \texttt{MAX}[T_C(j-1, n-j), T_C(j, n-j-1)] + 1, & \text{if } 1 \leq j < n \\ 0, & \text{otherwise} \end{cases}$$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.2:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Split $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1$ and $\mathcal{S}_2$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* **Combine** sorted elements of $\mathcal{S}_1$ with $\mathcal{S}_2$

Combine-Step:

1. If $\mathcal{S}_1$ (or $\mathcal{S}_2$) is empty, Return elements of $\mathcal{S}_2$ (or $\mathcal{S}_1$)
2. Compare first elements, $a_1 \in \mathcal{S}_1$ with $b_1 \in \mathcal{S}_2$
3. If $a_1 \geq b_1$, Return a_1 followed by *combined sorted elements of* $\mathcal{S}_1 - \{a_1\}$ *with* $\mathcal{S}_2$. Otherwise, Return b_1 followed by *combined sorted elements of* $\mathcal{S}_1$ *with* $\mathcal{S}_2 - \{b_1\}$.

Recurrence: Number of comparisons done for combining, [Merge]

$$T_C(j, n-j) = \begin{cases} \texttt{MAX}[T_C(j-1, n-j), T_C(j, n-j-1)] + 1, & \text{if } 1 \leq j < n \\ 0, & \text{otherwise} \end{cases}$$

Number of comparisons done for overall sorting, [Merge-Sort]

[Arbitrary Split] $$T(n) = \begin{cases} T(i) + T(n-i) + T_C(i, n-i), & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

[Middle Split] $$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + T_C\left(\frac{n}{2}, \frac{n}{2}\right), & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.3:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Choose a pivot element $p \in \mathcal{S}$. **Partition** $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1 = \{a \mid a \geq p\}$ and $\mathcal{S}_2 = \{a \mid a < p\}$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* Return sorted elements of $\mathcal{S}_1$ followed by $\mathcal{S}_2$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.3:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Choose a pivot element $p \in \mathcal{S}$. **Partition** $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1 = \{a \mid a \geq p\}$ and $\mathcal{S}_2 = \{a \mid a < p\}$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* Return sorted elements of $\mathcal{S}_1$ followed by $\mathcal{S}_2$

Partition-Step: Linear scan elements of $\mathcal{S}$ and put into $\mathcal{S}_1$ and $\mathcal{S}_2$ sets.

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.3:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Choose a pivot element $p \in \mathcal{S}$. **Partition** $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1 = \{a \mid a \geq p\}$ and $\mathcal{S}_2 = \{a \mid a < p\}$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* Return sorted elements of $\mathcal{S}_1$ followed by $\mathcal{S}_2$

Partition-Step: Linear scan elements of $\mathcal{S}$ and put into $\mathcal{S}_1$ and $\mathcal{S}_2$ sets.

Recurrence: Number of comparisons done for partitioning, [Partition]

$$T_P(n) = \begin{cases} T_P(1) + T_P(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases} \quad \Rightarrow T_P(n) = n$$

Example-5: *Sort n-element Set $\mathcal{S}$ (in Descending Order)*

Strategy-5.3:

1. *Base Case.* If $n = 1$, Return element
2. *Decomposition.* Choose a pivot element $p \in \mathcal{S}$. **Partition** $\mathcal{S}$ into two non-empty sets, $\mathcal{S}_1 = \{a \mid a \geq p\}$ and $\mathcal{S}_2 = \{a \mid a < p\}$
3. *Recursion.* Sort $\mathcal{S}_1$ and $\mathcal{S}_2$ set elements
4. *Recomposition.* Return sorted elements of $\mathcal{S}_1$ followed by $\mathcal{S}_2$

Partition-Step: Linear scan elements of $\mathcal{S}$ and put into $\mathcal{S}_1$ and $\mathcal{S}_2$ sets.

Recurrence: Number of comparisons done for partitioning, [Partition]

$$T_P(n) = \begin{cases} T_P(1) + T_P(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases} \quad \Rightarrow T_P(n) = n$$

Number of comparisons done for overall sorting, [Quick-Sort]

[Arbitrary Split]
$$T(n) = \begin{cases} T(i) + T(n-i) + T_P(n), & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

[Middle Split]
$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + T_P(n), & \text{if } n > 1 \\ 0, & \text{if } n = 1 \end{cases}$$

General Form of (Equal) Divide and Conquer Recerrence

Recurrence Relation: Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a function,

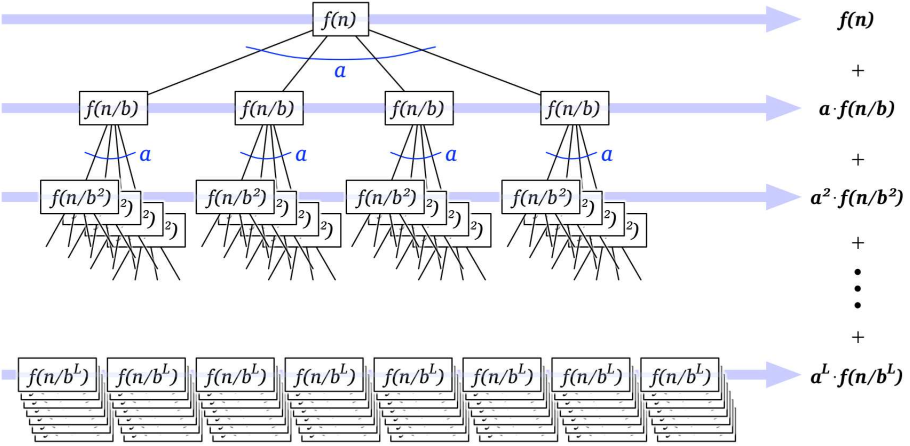
$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases}$$

General Form of (Equal) Divide and Conquer Recerrence

Recurrence Relation: Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a function,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases}$$

Recursion Tree: Step-wise unfolded form of computations from $T(n) = a.T\left(\frac{n}{b}\right) + f(n)$



A recursion tree for the recurrence $T(n) = a\,T(n/b) + f(n)$

Solution: Unfolding the computation steps as shown in the recursion tree, we get,

$$T(n) = a.T(\frac{n}{b}) + f(n) = a^2.T(\frac{n}{b^2}) + a.f(\frac{n}{b}) + f(n) = \cdots\cdots$$

$$= a^i.T(\frac{n}{b^i}) + \sum_{j=0}^{i-1} a^j.f(\frac{n}{b^j}) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f(\frac{n}{b^j}) \quad [as\ n = b^i]$$

General Form of (Equal) Divide and Conquer Recerrence

Solution: Unfolding the computation steps as shown in the recursion tree, we get,

$$T(n) = a.T(\frac{n}{b}) + f(n) = a^2.T(\frac{n}{b^2}) + a.f(\frac{n}{b}) + f(n) = \cdots\cdots$$

$$= a^i.T(\frac{n}{b^i}) + \sum_{j=0}^{i-1} a^j.f(\frac{n}{b^j}) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f(\frac{n}{b^j}) \quad [as\ n = b^i]$$

Case-1: If $f(n) \leq d.n^{\log_b a-\epsilon}$ for some constant $d, \epsilon > 0$, then

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j.f(\frac{n}{b^j}) \leq d. \sum_{j=0}^{\log_b n-1} a^j.(\frac{n}{b^j})^{\log_b a-\epsilon}$$

$$= d.n^{\log_b a-\epsilon}. \sum_{j=0}^{\log_b n-1} \left(\frac{a.b^\epsilon}{b^{\log_b a}}\right)^j = d.n^{\log_b a-\epsilon}. \sum_{j=0}^{\log_b n-1} (b^\epsilon)^j$$

$$= d.n^{\log_b a-\epsilon}.\left(\frac{b^{\epsilon.\log_b n} - 1}{b^\epsilon - 1}\right) = d.n^{\log_b a-\epsilon}.\left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right)$$

$$\leq D.n^{\log_b a} \quad [for\ some\ constant\ D > 0]$$

General Form of (Equal) Divide and Conquer Recerrence

Solution: Unfolding the computation steps as shown in the recursion tree, we get,

$$T(n) = a.T\left(\frac{n}{b}\right) + f(n) = a^2.T\left(\frac{n}{b^2}\right) + a.f\left(\frac{n}{b}\right) + f(n) = \cdots\cdots$$

$$= a^i.T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n - 1} a^j.f\left(\frac{n}{b^j}\right) \quad [as\ n = b^i]$$

Case-1: If $f(n) \leq d.n^{\log_b a - \epsilon}$ for some constant $d, \epsilon > 0$, then

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j.f\left(\frac{n}{b^j}\right) \leq d.\sum_{j=0}^{\log_b n - 1} a^j.\left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}$$

$$= d.n^{\log_b a - \epsilon}.\sum_{j=0}^{\log_b n - 1} \left(\frac{a.b^\epsilon}{b^{\log_b a}}\right)^j = d.n^{\log_b a - \epsilon}.\sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j$$

$$= d.n^{\log_b a - \epsilon}.\left(\frac{b^{\epsilon.\log_b n} - 1}{b^\epsilon - 1}\right) = d.n^{\log_b a - \epsilon}.\left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right)$$

$$\leq D.n^{\log_b a} \quad [for\ some\ constant\ D > 0]$$

So, $T(n) \leq c.n^{\log_b a} + D.n^{\log_b a} \leq C.n^{\log_b a}$ [for some constant $C > 0$]

General Form of (Equal) Divide and Conquer Recerrence

Case-2: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2 > 0$, then

General Form of (Equal) Divide and Conquer Recerrence

Case-2: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2 > 0$, then

$$\begin{aligned} g(n) &= \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) \leq d_2.\sum_{j=0}^{\log_b n-1} a^j.\left(\frac{n}{b^j}\right)^{\log_b a} \\ &= d_2.n^{\log_b a}.\sum_{j=0}^{\log_b n-1}\left(\frac{a}{b^{\log_b a}}\right)^j = d_2.n^{\log_b a}.\sum_{j=0}^{\log_b n-1} 1 \\ &= d_2.n^{\log_b a}.\log_b n \leq D_2.n^{\log_b a}.\log_2 n \ [\textit{for some constant } D_2 > 0] \end{aligned}$$

General Form of (Equal) Divide and Conquer Recerrence

Case-2: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2 > 0$, then

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) \leq d_2.\sum_{j=0}^{\log_b n-1} a^j.\left(\frac{n}{b^j}\right)^{\log_b a}$$

$$= d_2.n^{\log_b a}.\sum_{j=0}^{\log_b n-1}\left(\frac{a}{b^{\log_b a}}\right)^j = d_2.n^{\log_b a}.\sum_{j=0}^{\log_b n-1} 1$$

$$= d_2.n^{\log_b a}.\log_b n \leq D_2.n^{\log_b a}.\log_2 n \text{ [for some constant } D_2 > 0]$$

Similarly, $g(n) \geq D_1.n^{\log_b a}.\log_2 n$ [for some constant $D_1 > 0$]

General Form of (Equal) Divide and Conquer Recerrence

Case-2: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2 > 0$, then

$$
\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) \leq d_2.\sum_{j=0}^{\log_b n-1} a^j.\left(\frac{n}{b^j}\right)^{\log_b a} \\
&= d_2.n^{\log_b a}.\sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j = d_2.n^{\log_b a}.\sum_{j=0}^{\log_b n-1} 1 \\
&= d_2.n^{\log_b a}.\log_b n \leq D_2.n^{\log_b a}.\log_2 n \text{ [for some constant } D_2 > 0]
\end{aligned}
$$

Similarly, $g(n) \geq D_1.n^{\log_b a}.\log_2 n$ [for some constant $D_1 > 0$]

Therefore,

$$
\begin{aligned}
c.n^{\log_b a} + D_1.n^{\log_b a}.\log_2 n \leq \; & T(n) \leq c.n^{\log_b a} + D_2.n^{\log_b a}.\log_2 n \\
\Rightarrow \quad C_1.n^{\log_b a}.\log_2 n \leq \; & T(n) \leq C_2.n^{\log_b a}.\log_2 n
\end{aligned}
$$

[for some constants $C_1, C_2 > 0$]

General Form of (Equal) Divide and Conquer Recerrence

Case-3: We had, $T(n) \;=\; c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) \;=\; c.n^{\log_b a} + g(n)$

If $f(n) \geq d.n^{\log_b a+\epsilon}$ for some constant $d, \epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then

General Form of (Equal) Divide and Conquer Recerrence

Case-3: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $f(n) \geq d.n^{\log_b a+\epsilon}$ for some constant $d, \epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then

$$a.f\left(\frac{n}{b}\right) \leq k.f(n) \Rightarrow f\left(\frac{n}{b}\right) \leq \frac{k}{a}.f(n) \Rightarrow f\left(\frac{n}{b^2}\right) \leq \frac{k}{a}.f\left(\frac{n}{b}\right) \leq \left(\frac{k}{a}\right)^2.f(n)$$

Iterating in this manner, we get, $f(\frac{n}{b^j}) \leq (\frac{k}{a})^j.f(n)$. Hence,

General Form of (Equal) Divide and Conquer Recerrence

Case-3: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $f(n) \geq d.n^{\log_b a+\epsilon}$ for some constant $d, \epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then

$$a.f\left(\frac{n}{b}\right) \leq k.f(n) \Rightarrow f\left(\frac{n}{b}\right) \leq \frac{k}{a}.f(n) \Rightarrow f\left(\frac{n}{b^2}\right) \leq \frac{k}{a}.f\left(\frac{n}{b}\right) \leq \left(\frac{k}{a}\right)^2.f(n)$$

Iterating in this manner, we get, $f(\frac{n}{b^j}) \leq (\frac{k}{a})^j.f(n)$. Hence,

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) \leq \sum_{j=0}^{\log_b n-1} a^j.\left(\frac{k}{a}\right)^j.f(n) = \sum_{j=0}^{\log_b n-1} k^j.f(n)$$

$$\leq f(n).\sum_{j=0}^{\infty} k^j = \left(\frac{1}{1-k}\right).f(n)$$

General Form of (Equal) Divide and Conquer Recerrence

Case-3: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $f(n) \geq d.n^{\log_b a+\epsilon}$ for some constant $d, \epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then

$$a.f\left(\frac{n}{b}\right) \leq k.f(n) \Rightarrow f\left(\frac{n}{b}\right) \leq \frac{k}{a}.f(n) \Rightarrow f\left(\frac{n}{b^2}\right) \leq \frac{k}{a}.f\left(\frac{n}{b}\right) \leq \left(\frac{k}{a}\right)^2.f(n)$$

Iterating in this manner, we get, $f(\frac{n}{b^j}) \leq (\frac{k}{a})^j.f(n)$. Hence,

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) \leq \sum_{j=0}^{\log_b n-1} a^j.\left(\frac{k}{a}\right)^j.f(n) = \sum_{j=0}^{\log_b n-1} k^j.f(n)$$

$$\leq f(n).\sum_{j=0}^{\infty} k^j = \left(\frac{1}{1-k}\right).f(n)$$

Since $k < 1$ is a constant, for exact powers of b we can conclude that,

$$D_1.f(n) \leq g(n) \leq D_2.f(n) \qquad [\text{for some constants } D_1, D_2 > 0]$$

General Form of (Equal) Divide and Conquer Recerrence

Case-3: We had, $T(n) = c.n^{\log_b a} + \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) = c.n^{\log_b a} + g(n)$

If $f(n) \geq d.n^{\log_b a+\epsilon}$ for some constant $d, \epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then

$$a.f\left(\frac{n}{b}\right) \leq k.f(n) \Rightarrow f\left(\frac{n}{b}\right) \leq \frac{k}{a}.f(n) \Rightarrow f\left(\frac{n}{b^2}\right) \leq \frac{k}{a}.f\left(\frac{n}{b}\right) \leq \left(\frac{k}{a}\right)^2.f(n)$$

Iterating in this manner, we get, $f(\frac{n}{b^j}) \leq (\frac{k}{a})^j.f(n)$. Hence,

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j.f\left(\frac{n}{b^j}\right) \leq \sum_{j=0}^{\log_b n-1} a^j.\left(\frac{k}{a}\right)^j.f(n) = \sum_{j=0}^{\log_b n-1} k^j.f(n)$$

$$\leq f(n).\sum_{j=0}^{\infty} k^j = \left(\frac{1}{1-k}\right).f(n)$$

Since $k < 1$ is a constant, for exact powers of b we can conclude that,

$$D_1.f(n) \leq g(n) \leq D_2.f(n) \quad [\textit{for some constants } D_1, D_2 > 0]$$

Therefore, [for some constants $C_1, C_2 > 0$]

$$c.n^{\log_b a} + D_1.f(n) \leq T(n) \leq c.n^{\log_b a} + D_2.f(n)$$

$$\Rightarrow C_1.f(n) \leq T(n) \leq C_2.f(n) \quad [\textit{with } f(n) \geq d.n^{\log_b a+\epsilon}]$$

Master Theorem

Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a non-negative function defined on exact powers of b. We define $T(n)$ on exact powers of b by the following recurrence,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases} \qquad [\text{ where } i \in \mathbb{Z}^+ \ ]$$

Then, $T(n)$ follows the following inequalities:

Master Theorem

Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a non-negative function defined on exact powers of b. We define $T(n)$ on exact powers of b by the following recurrence,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases} \qquad [\text{ where } i \in \mathbb{Z}^+]$$

Then, $T(n)$ follows the following inequalities:

1. If $f(n) \leq d.n^{\log_b a - \epsilon}$ for some constant $d, \epsilon > 0$, then $T(n) \leq C.n^{\log_b a}$, for some constant $C > 0$.

Master Theorem

Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a non-negative function defined on exact powers of b. We define $T(n)$ on exact powers of b by the following recurrence,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases} \qquad [\text{ where } i \in \mathbb{Z}^+ \ ]$$

Then, $T(n)$ follows the following inequalities:

1. If $f(n) \leq d.n^{\log_b a - \epsilon}$ for some constant $d, \epsilon > 0$, then $T(n) \leq C.n^{\log_b a}$, for some constant $C > 0$.

   If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = O(n^{\log_b a})$

Master Theorem

Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a non-negative function defined on exact powers of b. We define $T(n)$ on exact powers of b by the following recurrence,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases} \qquad [\text{ where } i \in \mathbb{Z}^+ \text{ }]$$

Then, $T(n)$ follows the following inequalities:

1. If $f(n) \leq d.n^{\log_b a - \epsilon}$ for some constant $d, \epsilon > 0$, then $T(n) \leq C.n^{\log_b a}$, for some constant $C > 0$.

   If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = O(n^{\log_b a})$

2. If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2, \epsilon > 0$, then $C_1.n^{\log_b a}.\log_2 n \leq T(n) \leq C_2.n^{\log_b a}.\log_2 n$, for some constant $C_1, C_2 > 0$.

Master Theorem

Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a non-negative function defined on exact powers of b. We define $T(n)$ on exact powers of b by the following recurrence,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases} \qquad [\text{ where } i \in \mathbb{Z}^+ \ ]$$

Then, $T(n)$ follows the following inequalities:

1. If $f(n) \leq d.n^{\log_b a - \epsilon}$ for some constant $d, \epsilon > 0$, then $T(n) \leq C.n^{\log_b a}$, for some constant $C > 0$.
   If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = O(n^{\log_b a})$
2. If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2, \epsilon > 0$, then $C_1.n^{\log_b a}.\log_2 n \leq T(n) \leq C_2.n^{\log_b a}.\log_2 n$, for some constant $C_1, C_2 > 0$.
   If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a}.\log_2 n)$

Master Theorem

Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a non-negative function defined on exact powers of b. We define $T(n)$ on exact powers of b by the following recurrence,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases} \qquad [\text{ where } i \in \mathbb{Z}^+ \text{]}$$

Then, $T(n)$ follows the following inequalities:

1. If $f(n) \leq d.n^{\log_b a - \epsilon}$ for some constant $d, \epsilon > 0$, then $T(n) \leq C.n^{\log_b a}$, for some constant $C > 0$.
   If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = O(n^{\log_b a})$
2. If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2, \epsilon > 0$, then $C_1.n^{\log_b a}.\log_2 n \leq T(n) \leq C_2.n^{\log_b a}.\log_2 n$, for some constant $C_1, C_2 > 0$.
   If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a}.\log_2 n)$
3. If $f(n) \geq d.n^{\log_b a + \epsilon}$ for some constant $d, \epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then $C_1.f(n) \leq T(n) \leq C_2.f(n)$, for some constant $C_1, C_2 > 0$.

Master Theorem

Let $a \geq 1$, $b > 1$ and c be constants, and $f(n)$ be a non-negative function defined on exact powers of b. We define $T(n)$ on exact powers of b by the following recurrence,

$$T(n) = \begin{cases} a.T\left(\frac{n}{b}\right) + f(n) & n = b^i > 1 \\ c, & n = 1 \end{cases} \qquad [\text{ where } i \in \mathbb{Z}^+ \,]$$

Then, $T(n)$ follows the following inequalities:

1. If $f(n) \leq d.n^{\log_b a - \epsilon}$ for some constant $d, \epsilon > 0$, then $T(n) \leq C.n^{\log_b a}$, for some constant $C > 0$.
   If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = O(n^{\log_b a})$
2. If $d_1.n^{\log_b a} \leq f(n) \leq d_2.n^{\log_b a}$ for some constant $d_1, d_2, \epsilon > 0$, then $C_1.n^{\log_b a}.\log_2 n \leq T(n) \leq C_2.n^{\log_b a}.\log_2 n$, for some constant $C_1, C_2 > 0$.
   If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a}.\log_2 n)$
3. If $f(n) \geq d.n^{\log_b a + \epsilon}$ for some constant $d, \epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then $C_1.f(n) \leq T(n) \leq C_2.f(n)$, for some constant $C_1, C_2 > 0$.
   If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and $a.f(\frac{n}{b}) \leq k.f(n)$ for some constant $k < 1$ and for all sufficiently large $n \geq b$, then $T(n) = \Theta(f(n))$

Example Applications of Master Theorem

1. In the recurrence relation, $T(n) = \begin{cases} 9T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

Example Applications of Master Theorem

1. In the recurrence relation, $T(n) = \begin{cases} 9T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 9, b = 2, f(n) = 2n^3$. Now, $f(n) = 2n^3 \le d.n^{\log_2 9 - \epsilon}$ for some $d = 3, \epsilon > 0$. [Case-1]

   Hence, $T(n) \le C.n^{\log_2 9} \quad \Rightarrow T(n) = O(n^{\log_2 9})$

Example Applications of Master Theorem

1. In the recurrence relation, $T(n) = \begin{cases} 9T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 9, b = 2, f(n) = 2n^3$. Now, $f(n) = 2n^3 \leq d.n^{\log_2 9-\epsilon}$ for some $d = 3, \epsilon > 0$. [Case-1]

   Hence, $T(n) \leq C.n^{\log_2 9} \quad \Rightarrow T(n) = O(n^{\log_2 9})$

2. In the recurrence relation, $T(n) = \begin{cases} 8T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

Example Applications of Master Theorem

1. In the recurrence relation, $T(n) = \begin{cases} 9T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 9, b = 2, f(n) = 2n^3$. Now, $f(n) = 2n^3 \leq d.n^{\log_2 9 - \epsilon}$ for some $d = 3, \epsilon > 0$. [Case-1]

   Hence, $T(n) \leq C.n^{\log_2 9} \quad \Rightarrow T(n) = O(n^{\log_2 9})$

2. In the recurrence relation, $T(n) = \begin{cases} 8T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 8, b = 2, f(n) = 2n^3$. Now, $d_1.n^{\log_2 8} \leq 2.n^3 = f(n)$ and $f(n) = 2n^3 \leq d_2.n^{\log_2 8}$ for some $d_1 = 1, d_2 = 3, \epsilon > 0$. [Case-2]

   Hence, $C_1.n^3.\log_2 n \leq T(n) \leq C_2.n^3.\log_2 n \quad \Rightarrow T(n) = \Theta(n^3.\log_2 n)$

Example Applications of Master Theorem

1. In the recurrence relation, $T(n) = \begin{cases} 9T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 9, b = 2, f(n) = 2n^3$. Now, $f(n) = 2n^3 \leq d.n^{\log_2 9 - \epsilon}$ for some $d = 3, \epsilon > 0$. [Case-1]

   Hence, $T(n) \leq C.n^{\log_2 9} \quad \Rightarrow T(n) = O(n^{\log_2 9})$

2. In the recurrence relation, $T(n) = \begin{cases} 8T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 8, b = 2, f(n) = 2n^3$. Now, $d_1.n^{\log_2 8} \leq 2.n^3 = f(n)$ and $f(n) = 2n^3 \leq d_2.n^{\log_2 8}$ for some $d_1 = 1, d_2 = 3, \epsilon > 0$. [Case-2]

   Hence, $C_1.n^3.\log_2 n \leq T(n) \leq C_2.n^3.\log_2 n \quad \Rightarrow T(n) = \Theta(n^3.\log_2 n)$

3. In the recurrence relation, $T(n) = \begin{cases} 7T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

Example Applications of Master Theorem

1. In the recurrence relation, $T(n) = \begin{cases} 9T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 9, b = 2, f(n) = 2n^3$. Now, $f(n) = 2n^3 \leq d.n^{\log_2 9 - \epsilon}$ for some $d = 3, \epsilon > 0$. [Case-1]

   Hence, $T(n) \leq C.n^{\log_2 9} \quad \Rightarrow T(n) = O(n^{\log_2 9})$

2. In the recurrence relation, $T(n) = \begin{cases} 8T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 8, b = 2, f(n) = 2n^3$. Now, $d_1.n^{\log_2 8} \leq 2.n^3 = f(n)$ and $f(n) = 2n^3 \leq d_2.n^{\log_2 8}$ for some $d_1 = 1, d_2 = 3, \epsilon > 0$. [Case-2]

   Hence, $C_1.n^3.\log_2 n \leq T(n) \leq C_2.n^3.\log_2 n \quad \Rightarrow T(n) = \Theta(n^3.\log_2 n)$

3. In the recurrence relation, $T(n) = \begin{cases} 7T(\frac{n}{2}) + 2n^3, & n > 1 \\ 1, & n = 1 \end{cases}$,

   we find that $a = 7, b = 2, f(n) = 2n^3$. Now, $f(n) = 2.n^3 \geq d.n^{\log_2 7 + \epsilon}$ for any $d, \epsilon > 0$, and $7.f(\frac{n}{2}) = \frac{7}{4}.n^3 \leq k.2n^3$ for $k < 1$. [Case-3]

   Hence, $C_1.2n^3 \leq T(n) \leq C_2.2n^3 \quad \Rightarrow T(n) = \Theta(n^3)$

General Form of (Unequal) Divide and Conquer Recurrence

Recurrence Relation: For all i ($i \in \mathbb{Z}^+$), let a_i, α_i, k, c be constants where $a_i, k \in \mathbb{Z}^+$ and $0 < \alpha_i < 1$; and $f(n)$ be a function.

We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} a_1.T(\alpha_1.n) + a_2.T(\alpha_2.n) + \cdots + a_k.T(\alpha_k.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

General Form of (Unequal) Divide and Conquer Recurrence

Recurrence Relation: For all i ($i \in \mathbb{Z}^+$), let a_i, α_i, k, c be constants where $a_i, k \in \mathbb{Z}^+$ and $0 < \alpha_i < 1$; and $f(n)$ be a function.

We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} a_1.T(\alpha_1.n) + a_2.T(\alpha_2.n) + \cdots + a_k.T(\alpha_k.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Let us solve for a simpler variant of this recurrence defined as,

$$T(n) = \begin{cases} a.T(\alpha.n) + b.T(\beta.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$ [a, b, c are constants]

General Form of (Unequal) Divide and Conquer Recurrence

Recurrence Relation: For all i ($i \in \mathbb{Z}^+$), let a_i, α_i, k, c be constants where $a_i, k \in \mathbb{Z}^+$ and $0 < \alpha_i < 1$; and $f(n)$ be a function.
We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} a_1.T(\alpha_1.n) + a_2.T(\alpha_2.n) + \cdots + a_k.T(\alpha_k.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Let us solve for a simpler variant of this recurrence defined as,

$$T(n) = \begin{cases} a.T(\alpha.n) + b.T(\beta.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$ [a, b, c are constants]

Solution: By expansion we get,

$$\begin{aligned} T(n) &= a.T(\alpha.n) + b.T(\beta.n) + f(n) \\ &= a^2.T(\alpha^2.n) + 2.a.b.T(\alpha.\beta.n) + b^2.T(\beta^2.n) \quad + \quad f(n) + \Big[a.f(\alpha.n) + b.f(\beta.n)\Big] \end{aligned}$$

General Form of (Unequal) Divide and Conquer Recurrence

Recurrence Relation: For all i ($i \in \mathbb{Z}^+$), let a_i, α_i, k, c be constants where $a_i, k \in \mathbb{Z}^+$ and $0 < \alpha_i < 1$; and $f(n)$ be a function.
We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} a_1.T(\alpha_1.n) + a_2.T(\alpha_2.n) + \cdots + a_k.T(\alpha_k.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Let us solve for a simpler variant of this recurrence defined as,

$$T(n) = \begin{cases} a.T(\alpha.n) + b.T(\beta.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases} \quad [\ a, b, c \text{ are constants }]$$

Solution: By expansion we get,

$$\begin{aligned} T(n) &= a.T(\alpha.n) + b.T(\beta.n) + f(n) \\ &= a^2.T(\alpha^2.n) + 2.a.b.T(\alpha.\beta.n) + b^2.T(\beta^2.n) \quad + \quad f(n) + \Big[a.f(\alpha.n) + b.f(\beta.n)\Big] \\ &= \binom{3}{0}.a^3.T(\alpha^3.n) + \binom{3}{1}.a^2.b.T(\alpha^2.\beta.n) + \binom{3}{2}.a.b^2 T(\alpha.\beta^2.n) + \binom{3}{3}.b^3.T(\beta^3.n) + \Big[\binom{0}{0}.f(n)\Big] \\ &+ \quad \Big[\binom{1}{0}.a.f(\alpha.n) + \binom{1}{1}.b.f(\beta.n)\Big] + \Big[\binom{2}{0}.a^2.f(\alpha^2.n) + \binom{2}{1}.a.b.f(\alpha.\beta.n) + \binom{2}{2}.a.b^2 f(\beta^2.n)\Big] \end{aligned}$$

General Form of (Unequal) Divide and Conquer Recurrence

Recurrence Relation: For all i ($i \in \mathbb{Z}^+$), let a_i, α_i, k, c be constants where $a_i, k \in \mathbb{Z}^+$ and $0 < \alpha_i < 1$; and $f(n)$ be a function.
We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} a_1.T(\alpha_1.n) + a_2.T(\alpha_2.n) + \cdots + a_k.T(\alpha_k.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Let us solve for a simpler variant of this recurrence defined as,

$$T(n) = \begin{cases} a.T(\alpha.n) + b.T(\beta.n) + f(n) & n > 1 \\ c, & n = 1 \end{cases} \quad [\ a, b, c \text{ are constants}\ ]$$

Solution: By expansion we get,

$$\begin{aligned} T(n) &= a.T(\alpha.n) + b.T(\beta.n) + f(n) \\ &= a^2.T(\alpha^2.n) + 2.a.b.T(\alpha.\beta.n) + b^2.T(\beta^2.n) \quad + \quad f(n) + \Big[a.f(\alpha.n) + b.f(\beta.n)\Big] \\ &= \binom{3}{0}.a^3.T(\alpha^3.n) + \binom{3}{1}.a^2.b.T(\alpha^2.\beta.n) + \binom{3}{2}.a.b^2 T(\alpha.\beta^2.n) + \binom{3}{3}.b^3.T(\beta^3.n) + \Big[\binom{0}{0}.f(n)\Big] \\ &+ \Big[\binom{1}{0}.a.f(\alpha.n) + \binom{1}{1}.b.f(\beta.n)\Big] + \Big[\binom{2}{0}.a^2.f(\alpha^2.n) + \binom{2}{1}.a.b.f(\alpha.\beta.n) + \binom{2}{2}.a.b^2 f(\beta^2.n)\Big] \\ &= \quad \cdots\cdots \quad = \sum_{i=0}^{L-1} \Big[\binom{L+1}{i}.a^{L+1-i}.b^i T(\alpha^{L+1-i}.\beta^i.n) \quad + \quad \sum_{j=0}^{i} \binom{i}{j}.a^{i-j}.b^j f(\alpha^{i-j}.\beta^j.n) \Big] \end{aligned}$$

General Form of (Unequal) Divide and Conquer Recurrence

Solution (cont.): So, $T(n) = \sum_{i=0}^{L-1} \Big[\binom{L+1}{i} . a^{L+1-i} . b^{i}\, T(\alpha^{L+1-i} . \beta^{i} . n) \quad + \quad \sum_{j=0}^{i} \binom{i}{j} . a^{i-j} . b^{j} f(\alpha^{i-j} . \beta^{j} . n) \Big]$

General Form of (Unequal) Divide and Conquer Recurrence

Solution (cont.): So, $T(n) = \sum_{i=0}^{L-1} \Big[\binom{L+1}{i} . a^{L+1-i} . b^i T(\alpha^{L+1-i} . \beta^i . n) \quad + \quad \sum_{j=0}^{i} \binom{i}{j} . a^{i-j} . b^j f(\alpha^{i-j} . \beta^j . n) \Big]$

Without loss of generality, let us assume that, $0 < \beta \leq \alpha < 1$ and $\alpha^{m_1}.n = 1$, $\beta^{m_2}.n = 1$ (Obviously, $m_1 \geq m_2$). Note that,

General Form of (Unequal) Divide and Conquer Recurrence

Solution (cont.): So, $T(n) = \sum_{i=0}^{L-1} \left[\binom{L+1}{i} . a^{L+1-i} . b^i \, T(\alpha^{L+1-i} . \beta^i . n) \quad + \quad \sum_{j=0}^{i} \binom{i}{j} . a^{i-j} . b^j f(\alpha^{i-j} . \beta^j . n) \right]$

Without loss of generality, let us assume that, $0 < \beta \leq \alpha < 1$ and $\alpha^{m_1} . n = 1$, $\beta^{m_2} . n = 1$ (Obviously, $m_1 \geq m_2$). Note that,

$$T(n) \leq T(\alpha^{m_1} . n) . \sum_{i=0}^{m_1} \left[\binom{m_1}{i} . a^{m_1 - i} . b^i \right] + \sum_{i=0}^{m_1 - 1} \left[\sum_{j=0}^{i} \binom{i}{j} . a^{i-j} . b^j . f(\alpha^{i-j} . \beta^j . n) \right]$$

$$= c.(a+b)^{\log_{\frac{1}{\alpha}} n} + \sum_{i=0}^{\left(\log_{\frac{1}{\alpha}} n\right) - 1} \sum_{j=0}^{i} \left[\binom{i}{j} . a^{i-j} . b^j . f(\alpha^{i-j} . \beta^j . n) \right] \qquad [as \; m_1 = \log_{\frac{1}{\alpha}} n]$$

General Form of (Unequal) Divide and Conquer Recurrence

Solution (cont.): So, $T(n) = \sum_{i=0}^{L-1} \left[\binom{L+1}{i}.a^{L+1-i}.b^i\,T(\alpha^{L+1-i}.\beta^i.n) \quad + \quad \sum_{j=0}^{i} \binom{i}{j}.a^{i-j}.b^j f(\alpha^{i-j}.\beta^j.n) \right]$

Without loss of generality, let us assume that, $0 < \beta \leq \alpha < 1$ and $\alpha^{m_1}.n = 1$, $\beta^{m_2}.n = 1$ (Obviously, $m_1 \geq m_2$). Note that,

$$T(n) \leq T(\alpha^{m_1}.n).\sum_{i=0}^{m_1} \left[\binom{m_1}{i}.a^{m_1-i}.b^i \right] + \sum_{i=0}^{m_1-1} \left[\sum_{j=0}^{i} \binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n) \right]$$

$$= c.(a+b)^{\log_{\frac{1}{\alpha}} n} + \sum_{i=0}^{\left(\log_{\frac{1}{\alpha}} n\right)-1} \sum_{j=0}^{i} \left[\binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n) \right] \qquad [as\ m_1 = \log_{\frac{1}{\alpha}} n]$$

$$T(n) \geq T(\beta^{m_2}.n).\sum_{i=0}^{m_2} \left[\binom{m_2}{i}.a^{m_2-i}.b^i \right] + \sum_{i=0}^{m_2-1} \left[\sum_{j=0}^{i} \binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n) \right]$$

$$= c.(a+b)^{\log_{\frac{1}{\beta}} n} + \sum_{i=0}^{\left(\log_{\frac{1}{\beta}} n\right)-1} \sum_{j=0}^{i} \left[\binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n) \right] \qquad [as\ m_2 = \log_{\frac{1}{\beta}} n]$$

General Form of (Unequal) Divide and Conquer Recurrence

Solution (cont.): So, $T(n) = \sum_{i=0}^{L-1} \Big[\binom{L+1}{i}.a^{L+1-i}.b^i\,T(\alpha^{L+1-i}.\beta^i.n) \quad + \quad \sum_{j=0}^{i} \binom{i}{j}.a^{i-j}.b^j f(\alpha^{i-j}.\beta^j.n) \Big]$

Without loss of generality, let us assume that, $0 < \beta \leq \alpha < 1$ and $\alpha^{m_1}.n = 1,\ \beta^{m_2}.n = 1$ (Obviously, $m_1 \geq m_2$). Note that,

$$T(n) \leq T(\alpha^{m_1}.n).\sum_{i=0}^{m_1}\left[\binom{m_1}{i}.a^{m_1-i}.b^i\right] + \sum_{i=0}^{m_1-1}\left[\sum_{j=0}^{i}\binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n)\right]$$

$$= c.(a+b)^{\log_{\frac{1}{\alpha}} n} + \sum_{i=0}^{\left(\log_{\frac{1}{\alpha}} n\right)-1}\sum_{j=0}^{i}\left[\binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n)\right] \qquad [as\ m_1 = \log_{\frac{1}{\alpha}} n]$$

$$T(n) \geq T(\beta^{m_2}.n).\sum_{i=0}^{m_2}\left[\binom{m_2}{i}.a^{m_2-i}.b^i\right] + \sum_{i=0}^{m_2-1}\left[\sum_{j=0}^{i}\binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n)\right]$$

$$= c.(a+b)^{\log_{\frac{1}{\beta}} n} + \sum_{i=0}^{\left(\log_{\frac{1}{\beta}} n\right)-1}\sum_{j=0}^{i}\left[\binom{i}{j}.a^{i-j}.b^j.f(\alpha^{i-j}.\beta^j.n)\right] \qquad [as\ m_2 = \log_{\frac{1}{\beta}} n]$$

Finding Closed-form Expressions under different Cases (like Master Theorem):

Left for You to Explore!

Example Application of (Unequal) Divide & Conquer Recurrence

Revisit the recurrence capturing number of comparisons for *Fractional Split* in Divide and Conquer Search Strategy (in Linear-Search):

$$T(n) = \begin{cases} T(\frac{n}{3}) + T(\frac{2n}{3}), & n > 1 \\ 1, & n = 1 \end{cases}$$

Example Application of (Unequal) Divide & Conquer Recurrence

Revisit the recurrence capturing number of comparisons for *Fractional Split* in Divide and Conquer Search Strategy (in Linear-Search):

$$T(n) = \begin{cases} T(\frac{n}{3}) + T(\frac{2n}{3}), & n > 1 \\ 1, & n = 1 \end{cases}$$

Here, $f(n) = 0$ and $a = b = 1$, $\alpha = \frac{2}{3}$, $\beta = \frac{1}{3}$, so unfolding the recurrence (or draw the recursion tree) reveals the following equation:

$$T(n) = \sum_{i=0}^{k} \binom{k}{i} . T\left(\frac{2^i . n}{3^k}\right)$$

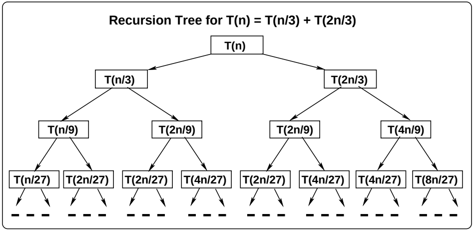
Example Application of (Unequal) Divide & Conquer Recurrence

Revisit the recurrence capturing number of comparisons for *Fractional Split* in Divide and Conquer Search Strategy (in Linear-Search):

$$T(n) = \begin{cases} T(\frac{n}{3}) + T(\frac{2n}{3}), & n > 1 \\ 1, & n = 1 \end{cases}$$

Here, $f(n) = 0$ and $a = b = 1$, $\alpha = \frac{2}{3}$, $\beta = \frac{1}{3}$, so unfolding the recurrence (or draw the recursion tree) reveals the following equation:

$$T(n) = \sum_{i=0}^{k} \binom{k}{i} . T\left(\frac{2^i . n}{3^k}\right)$$



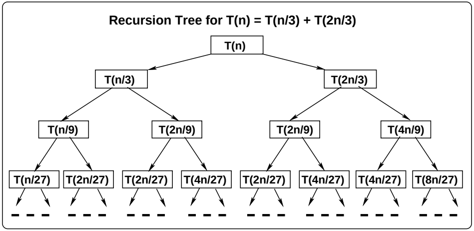
Example Application of (Unequal) Divide & Conquer Recurrence

Revisit the recurrence capturing number of comparisons for *Fractional Split* in Divide and Conquer Search Strategy (in Linear-Search):

$$T(n) = \begin{cases} T(\frac{n}{3}) + T(\frac{2n}{3}), & n > 1 \\ 1, & n = 1 \end{cases}$$

Here, $f(n) = 0$ and $a = b = 1$, $\alpha = \frac{2}{3}$, $\beta = \frac{1}{3}$, so unfolding the recurrence (or draw the recursion tree) reveals the following equation:

$$T(n) = \sum_{i=0}^{k} \binom{k}{i} . T\left(\frac{2^i.n}{3^k}\right)$$



Since in this case $m_1 = \log_{\frac{3}{2}} n \geq \log_3 n = m_2$, hence we can find the inequalities (in similar way as derived in the earlier slides),

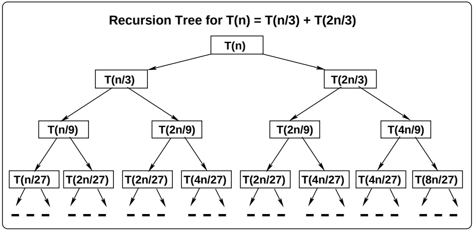
$$T(n) \leq 2^{\log_{\frac{3}{2}} n} = n^{\log_{\frac{3}{2}} 2} \quad \textit{and} \quad T(n) \geq 2^{\log_3 n} = n^{\log_3 2} \qquad \Rightarrow n^{\log_3 2} \leq T(n) \leq n^{\log_{\frac{3}{2}} 2}$$

Revisit the recurrence capturing number of comparisons for *Fractional Split* in Divide and Conquer Search Strategy (in Linear-Search):

$$T(n) = \begin{cases} T(\frac{n}{3}) + T(\frac{2n}{3}), & n > 1 \\ 1, & n = 1 \end{cases}$$

Here, $f(n) = 0$ and $a = b = 1$, $\alpha = \frac{2}{3}$, $\beta = \frac{1}{3}$, so unfolding the recurrence (or draw the recursion tree) reveals the following equation:

$$T(n) = \sum_{i=0}^{k} \binom{k}{i} . T\left(\frac{2^i.n}{3^k}\right)$$



Since in this case $m_1 = \log_{\frac{3}{2}} n \geq \log_3 n = m_2$, hence we can find the inequalities (in similar way as derived in the earlier slides),

$$T(n) \leq 2^{\log_{\frac{3}{2}} n} = n^{\log_{\frac{3}{2}} 2} \quad \textit{and} \quad T(n) \geq 2^{\log_3 n} = n^{\log_3 2} \qquad \Rightarrow n^{\log_3 2} \leq T(n) \leq n^{\log_{\frac{3}{2}} 2}$$

Exercise: $$T(n) = \begin{cases} T(\frac{n}{3}) + T(\frac{2n}{3}) + \log_2 n, & n > 1 \\ 1, & n = 1 \end{cases}$$

General Form of (Constant) Divide & Conquer Recurrence

Recurrence Relation: Let a $(0 < a < n)$ and c be constants, and $f(n)$ be a function. We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} T(a) + T(n-a) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

General Form of (Constant) Divide & Conquer Recurrence

Recurrence Relation: Let a $(0 < a < n)$ and c be constants, and $f(n)$ be a function. We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} T(a) + T(n-a) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Solution: Since the choice of constant a is equally likely (within $[1, n-1]$), therefore,

$$T(n) = \left(\frac{1}{n-1}\right). \sum_{i=1}^{n-1} [T(i) + T(n-i) + f(n)] = \left(\frac{2}{n-1}\right). \sum_{i=1}^{n-1} T(i) + f(n)$$

$$\Rightarrow \quad (n-1).T(n) = 2. \sum_{i=1}^{n-1} T(i) + (n-1)f(n)$$

General Form of (Constant) Divide & Conquer Recurrence

Recurrence Relation: Let a $(0 < a < n)$ and c be constants, and $f(n)$ be a function. We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} T(a) + T(n-a) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Solution: Since the choice of constant a is equally likely (within $[1, n-1]$), therefore,

$$T(n) = \left(\frac{1}{n-1}\right). \sum_{i=1}^{n-1} [T(i) + T(n-i) + f(n)] = \left(\frac{2}{n-1}\right). \sum_{i=1}^{n-1} T(i) + f(n)$$

$$\Rightarrow \quad (n-1).T(n) = 2. \sum_{i=1}^{n-1} T(i) + (n-1)f(n)$$

$$Similarly, \quad (n-2).T(n-1) = 2. \sum_{i=1}^{n-2} T(i) + (n-2).f(n-1)$$

General Form of (Constant) Divide & Conquer Recurrence

Recurrence Relation: Let a $(0 < a < n)$ and c be constants, and $f(n)$ be a function. We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} T(a) + T(n-a) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Solution: Since the choice of constant a is equally likely (within $[1, n-1]$), therefore,

$$T(n) = \left(\frac{1}{n-1}\right).\sum_{i=1}^{n-1}[T(i) + T(n-i) + f(n)] = \left(\frac{2}{n-1}\right).\sum_{i=1}^{n-1} T(i) + f(n)$$

$$\Rightarrow \quad (n-1).T(n) = 2.\sum_{i=1}^{n-1} T(i) + (n-1)f(n)$$

$$\textit{Similarly}, \quad (n-2).T(n-1) = 2.\sum_{i=1}^{n-2} T(i) + (n-2).f(n-1)$$

$$\textit{Subtracting}, \quad (n-1).T(n) - n.T(n-1) = (n-1).f(n) - (n-2).f(n-1)$$

$$\Rightarrow \quad \frac{T(n)}{n} - \frac{T(n-1)}{n-1} = \left(\frac{1}{n}\right).f(n) - \left(\frac{n-2}{n.(n-1)}\right).f(n-1)$$

General Form of (Constant) Divide & Conquer Recurrence

Recurrence Relation: Let a $(0 < a < n)$ and c be constants, and $f(n)$ be a function. We define $T(n)$ by the following recurrence,

$$T(n) = \begin{cases} T(a) + T(n-a) + f(n) & n > 1 \\ c, & n = 1 \end{cases}$$

Solution: Since the choice of constant a is equally likely (within $[1, n-1]$), therefore,

$$T(n) = \left(\frac{1}{n-1}\right). \sum_{i=1}^{n-1} [T(i) + T(n-i) + f(n)] = \left(\frac{2}{n-1}\right). \sum_{i=1}^{n-1} T(i) + f(n)$$

$$\Rightarrow \quad (n-1).T(n) = 2. \sum_{i=1}^{n-1} T(i) + (n-1)f(n)$$

$$Similarly, \quad (n-2).T(n-1) = 2. \sum_{i=1}^{n-2} T(i) + (n-2).f(n-1)$$

$$Subtracting, \quad (n-1).T(n) - n.T(n-1) = (n-1).f(n) - (n-2).f(n-1)$$

$$\Rightarrow \quad \frac{T(n)}{n} - \frac{T(n-1)}{n-1} = \left(\frac{1}{n}\right).f(n) - \left(\frac{n-2}{n.(n-1)}\right).f(n-1)$$

$$\Rightarrow \quad \frac{T(n)}{n} - \frac{T(n-1)}{n-1} = \left(\frac{1}{n}\right).f(n) + \left(\frac{1}{n-1} - \frac{2}{n}\right).f(n-1)$$

Solution (cont.):

$$
\begin{aligned}
\frac{T(n)}{n} - \frac{T(n-1)}{n-1} &= \left(\frac{1}{n}\right).f(n) + \left(\frac{1}{n-1} - \frac{2}{n}\right).f(n-1) \\
\frac{T(n-1)}{n-1} - \frac{T(n-2)}{n-2} &= \left(\frac{1}{n-1}\right).f(n-1) + \left(\frac{1}{n-2} - \frac{2}{n-1}\right).f(n-2) \\
\frac{T(n-2)}{n-2} - \frac{T(n-3)}{n-3} &= \left(\frac{1}{n-2}\right).f(n-2) + \left(\frac{1}{n-3} - \frac{2}{n-2}\right).f(n-3) \\
\cdots\cdots & \quad \cdots\cdots \\
\frac{T(3)}{3} - \frac{T(2)}{2} &= \left(\frac{1}{3}\right).f(3) - \left(\frac{1}{2} - \frac{2}{3}\right).f(2) \\
\frac{T(2)}{2} - \frac{T(1)}{1} &= \left(\frac{1}{2}\right).f(2) - \left(\frac{1}{1} - \frac{2}{2}\right).f(1)
\end{aligned}
$$

General Form of (Constant) Divide & Conquer Recurrence

Solution (cont.):

$$
\begin{aligned}
\frac{T(n)}{n} - \frac{T(n-1)}{n-1} &= \left(\frac{1}{n}\right).f(n) + \left(\frac{1}{n-1} - \frac{2}{n}\right).f(n-1) \\
\frac{T(n-1)}{n-1} - \frac{T(n-2)}{n-2} &= \left(\frac{1}{n-1}\right).f(n-1) + \left(\frac{1}{n-2} - \frac{2}{n-1}\right).f(n-2) \\
\frac{T(n-2)}{n-2} - \frac{T(n-3)}{n-3} &= \left(\frac{1}{n-2}\right).f(n-2) + \left(\frac{1}{n-3} - \frac{2}{n-2}\right).f(n-3) \\
\cdots\cdots & \quad \cdots\cdots \\
\frac{T(3)}{3} - \frac{T(2)}{2} &= \left(\frac{1}{3}\right).f(3) - \left(\frac{1}{2} - \frac{2}{3}\right).f(2) \\
\frac{T(2)}{2} - \frac{T(1)}{1} &= \left(\frac{1}{2}\right).f(2) - \left(\frac{1}{1} - \frac{2}{2}\right).f(1)
\end{aligned}
$$

Adding all the above equations, we get,

$$
\frac{T(n)}{n} - \frac{T(1)}{1} = \left(\frac{1}{n}\right).f(n) + 2.\sum_{i=2}^{n-1} \left[\left\{\frac{1}{i.(i+1)}\right\}.f(i)\right]
$$

General Form of (Constant) Divide & Conquer Recurrence

Solution (cont.):

$$
\begin{aligned}
\frac{T(n)}{n} - \frac{T(n-1)}{n-1} &= \left(\frac{1}{n}\right).f(n) + \left(\frac{1}{n-1} - \frac{2}{n}\right).f(n-1) \\
\frac{T(n-1)}{n-1} - \frac{T(n-2)}{n-2} &= \left(\frac{1}{n-1}\right).f(n-1) + \left(\frac{1}{n-2} - \frac{2}{n-1}\right).f(n-2) \\
\frac{T(n-2)}{n-2} - \frac{T(n-3)}{n-3} &= \left(\frac{1}{n-2}\right).f(n-2) + \left(\frac{1}{n-3} - \frac{2}{n-2}\right).f(n-3) \\
\cdots\cdots & \quad \cdots\cdots \\
\frac{T(3)}{3} - \frac{T(2)}{2} &= \left(\frac{1}{3}\right).f(3) - \left(\frac{1}{2} - \frac{2}{3}\right).f(2) \\
\frac{T(2)}{2} - \frac{T(1)}{1} &= \left(\frac{1}{2}\right).f(2) - \left(\frac{1}{1} - \frac{2}{2}\right).f(1)
\end{aligned}
$$

Adding all the above equations, we get,

$$
\begin{aligned}
\frac{T(n)}{n} - \frac{T(1)}{1} &= \left(\frac{1}{n}\right).f(n) + 2.\sum_{i=2}^{n-1}\left[\left\{\frac{1}{i.(i+1)}\right\}.f(i)\right] \\
\Rightarrow T(n) &= c + f(n) + 2n.\sum_{i=2}^{n-1}\left[\left\{\frac{1}{i.(i+1)}\right\}.f(i)\right]
\end{aligned}
$$

Example Application of (Constant) Divide & Conquer Recurrence

Revisit the recurrence capturing number of comparisons for *Arbitrary Split* in Divide and Conquer Sorting Strategy (in Quick-Sort):

$$T(n) = \begin{cases} T(a) + T(n-a) + n, & n > 1 \\ 0, & n = 1 \end{cases}$$

Example Application of (Constant) Divide & Conquer Recurrence

Revisit the recurrence capturing number of comparisons for *Arbitrary Split* in Divide and Conquer Sorting Strategy (in Quick-Sort):

$$T(n) = \begin{cases} T(a) + T(n-a) + n, & n > 1 \\ 0, & n = 1 \end{cases}$$

If we follow the derivation procedure in earlier slides, we get,

$$\begin{aligned} T(n) &= 0 + n + 2.n.\sum_{i=2}^{n-1}\Big[\Big\{\frac{1}{i.(i+1)}\Big\}.i\Big] \\ &= n + 2.n\Big[\frac{1}{3} + \frac{1}{4} + \cdots \frac{1}{n}\Big] = 2.n\Big[\Big(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\Big) - 1\Big] \\ &= 2.n.\Big(\ln n + \gamma + \frac{1}{2n} - 1\Big) \approx C.n\log_2 n \end{aligned}$$

[$\gamma = 0.5772156649..$ is the Euler-Mascheroni Constant and $C > 0$ is some constant]

Example Application of (Constant) Divide & Conquer Recurrence

Revisit the recurrence capturing number of comparisons for *Arbitrary Split* in Divide and Conquer Sorting Strategy (in Quick-Sort):

$$T(n) = \begin{cases} T(a) + T(n-a) + n, & n > 1 \\ 0, & n = 1 \end{cases}$$

If we follow the derivation procedure in earlier slides, we get,

$$\begin{aligned} T(n) &= 0 + n + 2.n. \sum_{i=2}^{n-1} \Big[\Big\{ \frac{1}{i.(i+1)} \Big\} .i \Big] \\ &= n + 2.n \Big[\frac{1}{3} + \frac{1}{4} + \cdots \frac{1}{n} \Big] = 2.n \Big[\Big(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \Big) - 1 \Big] \\ &= 2.n. \Big(\ln n + \gamma + \frac{1}{2n} - 1 \Big) \approx C.n \log_2 n \end{aligned}$$

[$\gamma = 0.5772156649..$ is the Euler-Mascheroni Constant and $C > 0$ is some constant]

Exercise: $$T(n) = \begin{cases} T(a) + T(n-a) + k.n. \log_2 n, & n > 1 \\ 1, & n = 1 \end{cases}$$

Some Variants of Divide and Conquer Recurrence: *Changing Variables*

Recurrence Relation: $T(n) = \begin{cases} 2.T(\sqrt{n}) + \log_2 n, & n > 2 \\ 1, & n = 2 \end{cases}$

Some Variants of Divide and Conquer Recurrence: *Changing Variables*

Recurrence Relation: $T(n) = \begin{cases} 2.T(\sqrt{n}) + \log_2 n, & n > 2 \\ 1, & n = 2 \end{cases}$

Solution: Let $n = 2^{2^m}$, implies $\log_2 n = 2^m$. So, we have

$$T(2^{2^m}) = 2.T(2^{2^{m-1}}) + 2^m$$

Recurrence Relation: $T(n) = \begin{cases} 2.T(\sqrt{n}) + \log_2 n, & n > 2 \\ 1, & n = 2 \end{cases}$

Solution: Let $n = 2^{2^m}$, implies $\log_2 n = 2^m$. So, we have

$$\begin{aligned} T(2^{2^m}) &= 2.T(2^{2^{m-1}}) + 2^m \\ \Rightarrow \quad S(m) &= 2.S(m-1) + 2^m \quad \textit{and} \quad S(0) = 1 \end{aligned}$$

Recurrence Relation: $T(n) = \begin{cases} 2.T(\sqrt{n}) + \log_2 n, & n > 2 \\ 1, & n = 2 \end{cases}$

Solution: Let $n = 2^{2^m}$, implies $\log_2 n = 2^m$. So, we have

$$\begin{aligned} T(2^{2^m}) &= 2.T(2^{2^{m-1}}) + 2^m \\ \Rightarrow \quad S(m) &= 2.S(m-1) + 2^m \quad \text{and} \quad S(0) = 1 \\ &= 2S(m-2) + 2.2^{m-1} + 2^m = 2S(m-2) + 2.2^m \\ &= 2S(m-3) + 3.2^m \quad = \quad \cdots\cdots \\ &= S(0) + m.2^m \quad = \quad 1 + m.2^m \end{aligned}$$

Recurrence Relation: $T(n) = \begin{cases} 2.T(\sqrt{n}) + \log_2 n, & n > 2 \\ 1, & n = 2 \end{cases}$

Solution: Let $n = 2^{2^m}$, implies $\log_2 n = 2^m$. So, we have

$$\begin{aligned} T(2^{2^m}) &= 2.T(2^{2^{m-1}}) + 2^m \\ \Rightarrow \quad S(m) &= 2.S(m-1) + 2^m \quad \textit{and} \quad S(0) = 1 \\ &= 2S(m-2) + 2.2^{m-1} + 2^m = 2S(m-2) + 2.2^m \\ &= 2S(m-3) + 3.2^m \quad = \quad \cdots\cdots \\ &= S(0) + m.2^m \quad = \quad 1 + m.2^m \end{aligned}$$

Therefore,

$$\begin{aligned} T(n) &= T(2^{2^m}) = S(m) = 1 + m.2^m \\ &= 1 + \log_2 n.(\log_2 \log_2 n) \end{aligned}$$

Recurrence Relation: $T(n) = \begin{cases} 2.T(\sqrt{n}) + \log_2 n, & n > 2 \\ 1, & n = 2 \end{cases}$

Solution: Let $n = 2^{2^m}$, implies $\log_2 n = 2^m$. So, we have

$$\begin{aligned} T(2^{2^m}) &= 2.T(2^{2^{m-1}}) + 2^m \\ \Rightarrow \quad S(m) &= 2.S(m-1) + 2^m \quad \textit{and} \quad S(0) = 1 \\ &= 2S(m-2) + 2.2^{m-1} + 2^m = 2S(m-2) + 2.2^m \\ &= 2S(m-3) + 3.2^m \quad = \quad \cdots\cdots \\ &= S(0) + m.2^m \quad = \quad 1 + m.2^m \end{aligned}$$

Therefore,

$$\begin{aligned} T(n) &= T(2^{2^m}) = S(m) = 1 + m.2^m \\ &= 1 + \log_2 n.(\log_2 \log_2 n) \end{aligned}$$

Exercise: $T(n) = \begin{cases} \sqrt{n}.T(\sqrt{n}) + n & n > 2 \\ 1, & n = 2 \end{cases}$

Thank You!