Using recursion we do the implementation of Binary Search

Recurrence Relation (Binary Search)

Recursion → Sorted array

BinarySearch(arr, i, j, x)

mid = i + (j-i)//2

Left side: i → starting index, j → ending index

arr[mid] == x ↳ return mid → O(1) ↳ c

arr(mid) < x ↳ Recursion ↳ BinarySearch(arr, mid+1, j, x) → $T(n/2)$ (Right side of mid)

arr(mid) > x ↳ Recursion BinarySearch(arr, i, mid-1, j) → $T(n/2)$

Binary Search

Recurrence Relation

$$T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

Left side, Right side

$a = 1,\ b = 2$

$$T(n) = T(n/2) + c$$

$f(n) = \Theta(n^k \log^p c)$

$k = 0,\ p = 0$

Master's Theorem :-

Calculating time complexity using Master Theorem for Binary Search algorithm

$\log_b a = \log_2 1 = 0$

$\log_b a = k$ → case 2

$\Rightarrow \Theta(c \cdot \log n)$

$\Rightarrow \Theta(\log n)$

Substitution Method

Calculating time complexity using Substitution method for Binary Search algorithm

$$T(n) = T\left(\frac{n}{2}\right) + c \quad \text{—— 1st}$$

$$T(n) = T\left(\frac{n}{2^2}\right) + c + c \quad \text{—— 2nd}$$

$$= T\left(\frac{n}{2^3}\right) + c + c + c \quad \text{— 3rd}$$

$T(1) = 1$

$$k \text{ times} \left\{ \frac{n}{2^k} = 1 \Rightarrow n = 2^k \right.$$

$$k = \log_2 n$$

$$\Rightarrow T\left(\frac{n}{2^k}\right) + c \cdot k$$

$$\Rightarrow T\left(\frac{n}{2^{\log_2 n}}\right) + c \cdot \log_2 n$$

$$\Rightarrow T\left(\frac{\cancel{n}\ 1}{\cancel{n}^{\log_2 \cancel{2}\,1}}\right) + c \cdot \log_2 n$$

$$\Rightarrow 1 + c \cdot \log_2 n$$

$$\Rightarrow O(\log_2 n)$$