→ Best case, Worst case & Expected case of an Algorithm:

→ Sometimes we get lucky in life. Exams cancelled when you were not prepared, Surprise test when you were prepared etc. ⇒ BEST CASE

→ Sometimes we get unlucky. Questions you never prepared asked in exams, Rain during sports period etc. ⇒ <u>Worst case</u>

→ But overall the life remains balanced with the mixture of lucky & unlucky events ⇒ <u>EXPECTED</u> case

→ Analysis of a search algorithm :
  → Consider an array which is sorted in ↑sing order.

1	7	18	28	50	180

→ We have to search a given number in this array & report whether its present in the array or not.

→ <u>Algo 1</u> → Start from first element until an element ↑ter than or equal to the number to be searched is found.

→ <u>Algo 2</u> → Check whether the first or the last element is equal to the number. If not find the number b/w these two elements (center of the array). If the center element is ↑ter than the number to be searched, repeat the process for first half else repeat for second half until

Date

the number is found.

→ Analyzing Algo 1:

→ If we really get lucky, the first element of the array might turn out to be the element we are searching for. Hence, we made just one comparison.

$\boxed{\text{Best Case Complexity} = O(1)}$ $T_n = K$

→ If we are really unlucky, the element we are searching for might be the last one.

$\boxed{\text{Worst Case Complexity} = O(n)}$ $T_n = nK$

→ For calculating Average case time, we sum the list of all the possible cases runtime & ~~decide~~ divide it with the total number of cases:

$$\text{Avg Case} = O\left(\frac{\Sigma \text{ All possible runtimes}}{\text{Total no. of possibility}}\right)$$

$$\text{Avg Case} = O\,\frac{[K + 2K + 3K \cdots nK] + [nK]}{n+1}$$

$$\Rightarrow O\,\frac{K[(1+2+3 \cdots n) + n]}{n+1}$$

$$\Rightarrow O\ K\left[\frac{n^2+n+2n}{2(n+1)}\right]$$ Ignoring 2n, Non-dominant.

$$O\ K\ \frac{n\cancel{(n+1)}}{2\cancel{(n+1)}} \Rightarrow O\left(\frac{K}{2}\right)n$$ It is constant

$\boxed{\therefore \text{Avg case} \Rightarrow O(n)}$

→ Analyzing Algo 2:

→ If we get really lucky, the first element will be the only one which gets compared.

$\boxed{\text{Best case Complexity} = O(1)}$

→ If we get unlucky, we will have to keep dividing the array into halves until we get a single element (the array gets finished).

$\boxed{\text{Worst case Complexity} = O(\log n)}$

→ What log (n)? What is that!

log (n) → Number of time we need to half the array of size n before it gets exhausted.

$$\log 8 = 3 \Rightarrow \frac{8}{2} \rightarrow \frac{4}{2} \rightarrow \frac{2}{2} \rightarrow$$ can't break anymore

$$1 + 1 + 1$$

Page No.

Date

$\log 4 = 2 \Rightarrow \frac{4}{2} \longrightarrow \frac{2}{2}$ → can't break anymore.

$1 + 1$

→ log(n) simply means how many times, I need to divide n units such that we cannot divide them (into halves) anymore.

→ Space Complexity: Time is not the only thing we worry about while analyzing algorithms. Space is equally important.

→ Creating an array of size n → O(n) space

SIZE OF INPUT

→ If a function calls itself recursively n times its space complexity is O(n).

→ Why can't we calculate complexity in seconds?

→ Not everyone's computer is equally powerful.

→ Asymptotic analysis is the measure of how time (runtime) grows with input.