log n (logarithm of n):

- which is commonly denoted as $\log_2(n)$.
- $\log_2(n) = x$ means $2^x = n$.
- Logarithms grow relatively slowly, which means that as 'n' gets larger, the increase in $\log_2(n)$ is much smaller compared to 'n' itself. For example, $\log_2(2) = 1$, $\log_2(4) = 2$, $\log_2(8) = 3$, and so on.

'n' (Linear Growth):

- As you increase the input size 'n,' the number of operations or the resource usage (such as time or space) also increases linearly.
- For example, if 'n' doubles, the resource usage also roughly doubles.

'n' (Linear Time): If you have 10 things to sort, it might take 10 seconds. If you have 20 things, it might take 20 seconds. The time it takes grows at the same rate as the number of things you have. It's like counting one by one.

'n^2' (Quadratic Time): If you have 10 things, it might take 100 seconds to sort them. If you have 20 things, it might take 400 seconds. The time it takes grows much faster than the number of things you have. It's like counting for each thing, and then counting again for each thing, which is slow.

'log n' (Logarithmic Time): Imagine you have a phone book with 100 pages, and you want to find a name. You start by opening the book roughly in the middle. If the name you're looking for is in the first half, you can eliminate the second half. Then, you repeat this process with the remaining half until you find the name. Even if the phone book had 1000 pages, you wouldn't need many more steps because you keep eliminating half of the pages each time. This is like 'log n' because it grows slowly.

'n log n' (Superlinear Time): Suppose you have a stack of 10 envelopes, and you want to address them and then put them in order based on the recipient's name. First, you address each envelope (which takes 'n' time), and then you arrange them alphabetically (which takes 'log n' time for each envelope). So, you do some work for each envelope ('n') and a bit more work to arrange them ('log n')

'log n^2' (Logarithmic Time of Squares): Imagine you have a puzzle with 100 pieces, and you want to assemble it. You start by finding two matching pieces (which takes 'log n' time). Then, you do this again with other pairs until you've put all 100 pieces together. Even if you had 1000 pieces, the process doesn't take much longer because you're still just matching pairs, not examining each piece individually.

Arrays:

- Best Case: O(1) - In the best case, accessing an element in an array is very fast. It's like knowing exactly where your toy is on a shelf. You just grab it instantly because you remember its exact location.

- Average Case: O(1) - On average, finding an item in an array is still quick. It's like having a bookshelf where you know the general area where your book is. You find it fairly fast by looking in the right section.

- Worst Case: O(n) - In the worst case, finding an element in an unsorted array can be slow. Imagine you have a pile of toys, and you have to check each one until you find the right one. It can take a long time if the toy is at the bottom.

Linked Lists:

- Best Case: O(1) - In the best case, adding or removing an element at the beginning of a linked list is very fast. It's like adding or removing a toy from the top of a stack. You can do it quickly.

- Average Case: O(n) - On average, finding an item in a linked list can be slower because you might have to go through each element one by one to find what you're looking for. It's like looking for a specific toy in a long line of toys.

- Worst Case: O(n) - In the worst case, you may need to traverse the entire linked list to find or remove an element at the end. It's like having a chain of toys, and you need to follow the chain all the way to the last toy.

Skip Lists:

- Best Case: O(1) - Skip lists are clever because they have multiple layers, like having shortcuts in a maze. In the best case, you can use these shortcuts to find an element very quickly. It's like having a map of the maze that shows you the shortest path.

- Average Case: O(log n) - On average, skip lists are still quite efficient. You might need to use a few shortcuts to find what you're looking for, but it's not too bad. It's like using a map with a few directions to navigate through a maze.

- Worst Case: O(n) - In the worst case, you might not have many shortcuts, and you'd need to go through almost all the elements. It's like having a maze with very few shortcuts, and you have to explore most of it to find your way.