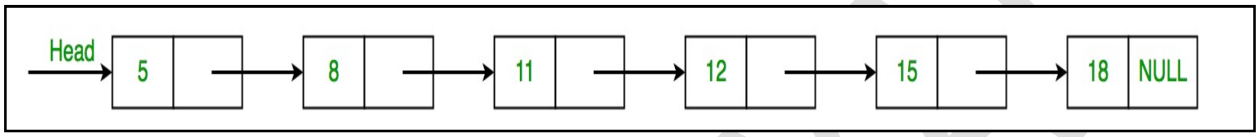
SKIP LIST

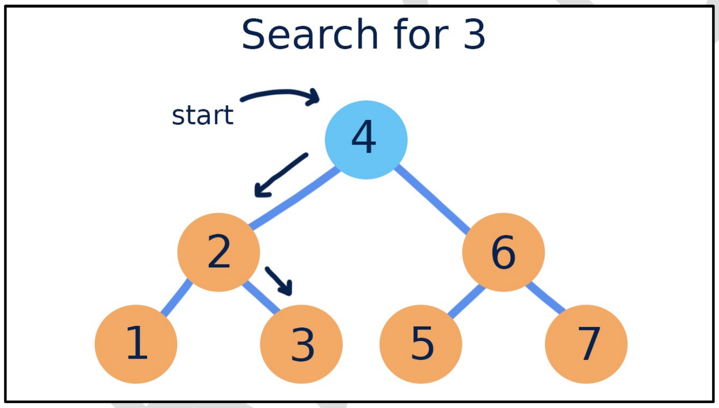
Need of Skip List

In a linked list whenever we want to search an element we have to traverse each node even in a sorted linked list till the time we find the node or reached the end. So, it takes so much of time with time complexity O(n).



In the above LL, if we want to search 12 then we have to travel through each node till 12.

In a non- linear data structure say Binary Search Tree, we almost neglect half of the nodes after every iteration which makes him take less time by skipping the nodes but no such thing is possible with simple linked list

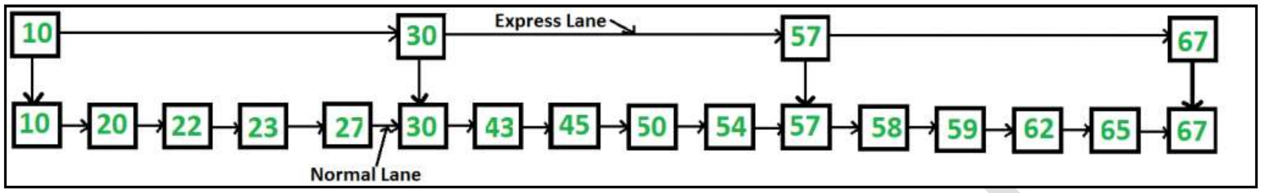


In the above Binary Search tree, if we want to search 3 we first check it with 4 if it is less we go on the left node neglecting all right node and proceed further which make it much efficient in use.

Now, we want the similar kind of thing with Linked list which makes our searching more efficient because search is the main thing we do to perform any type of operation.

Skip List

For the improve version of Linked List we use Skip List. In this we have elements stored at multi-levels.



Like in the above Skip list we have 2 lanes now if we want to search 54 in this, we first traverse through the express lane or say level 1 and search for 54 or its next first greater element than the required now when we reach 30 it's next is 57 which is greater than 54, now we go in the normal lane or say level 0 and move till 54.In this way we have searched for 54 in a better and efficient way by skipping many nodes before it. By trading our space for time we have improvised our search time.

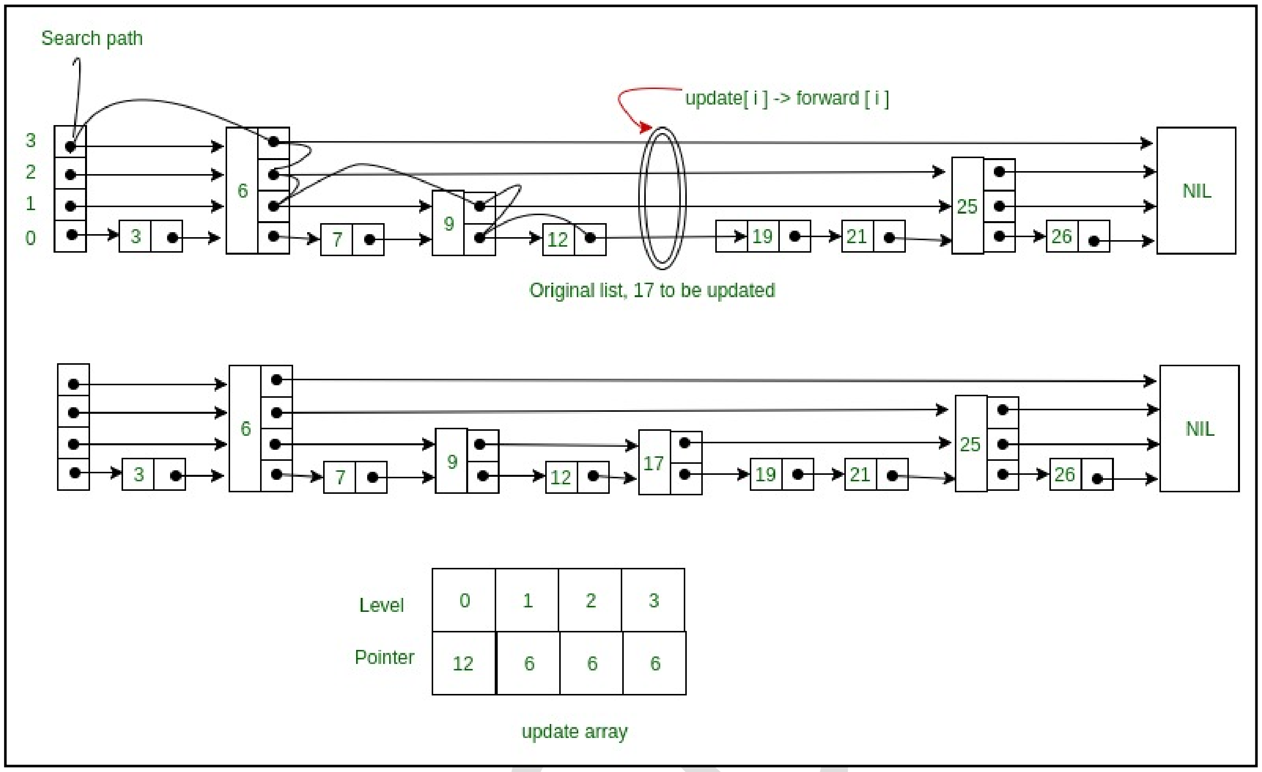
Also we can have as many lanes we want but it is prefer to have log(N/2) where n is total no of nodes in the given list.

Create Skip List & Performing Operations

Insertion:-

- Firstly, we decide the level of nodes it needs to be.
- Finding an appropriate position for the addition of node.
- By starting from the highest level, we compare each node with the key till the key is lesser than the key to be inserted, we go to a down level.
- Also keep an extra array in which we keep all the level nodes from where we come to down level.
- When we reach a level 0 we will insert a key at that place.
- Then update each level of that node with the help of the update array.

Example: - 17 to be insert

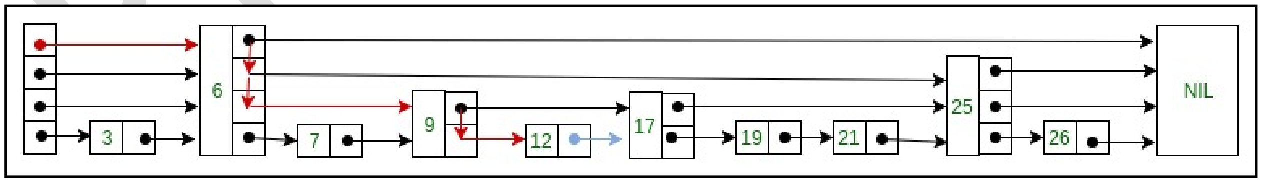


Level	0	1	2	3
Pointer	12	6	6	6

Searching:-

- By starting from the highest level, we compare each node with the key till the key is less than the key to be searched, we go to a down level.
- If the next key is equal to the key to be searched than key found else key not found.

Example: - 12 to be searched.

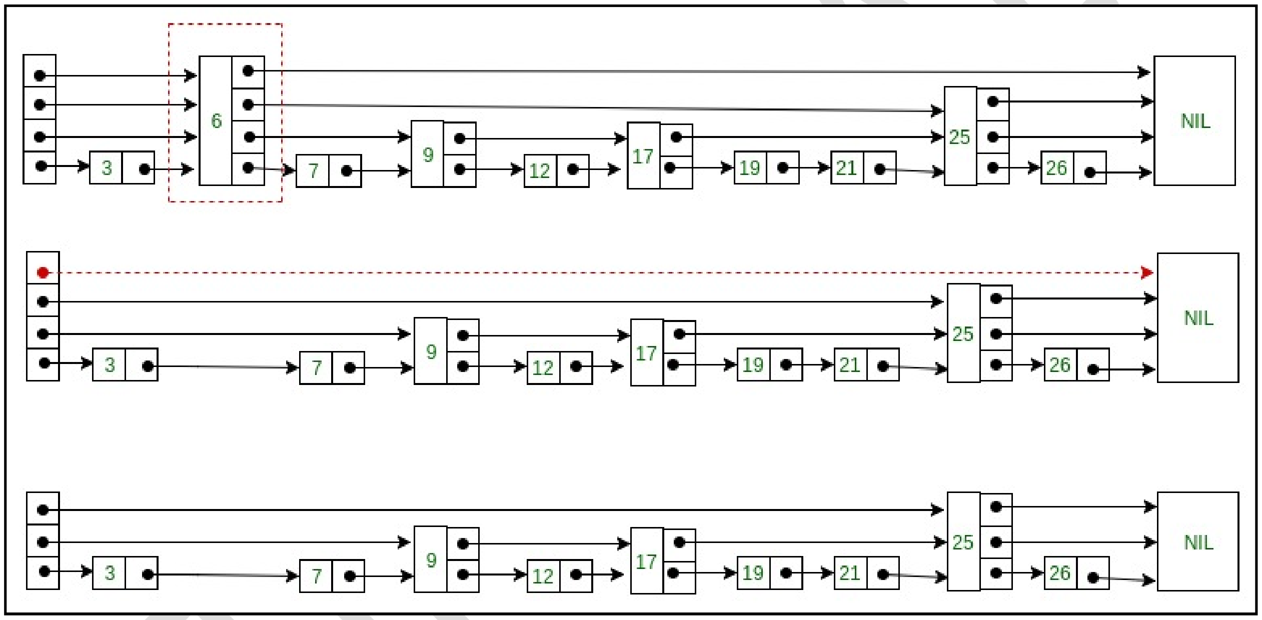


Deletion:-

- Firstly we will search for the element to be deleted.

- By starting from the highest level, we compare each node with the key till the key is lesser than the key to be deleted, we go to a down level.
- Also keep an extra array in which we keep all the level nodes from where we come to down level.
- When we reach a level 0 and found the key which is to be deleted we will just update all the pointers of the extra array and delete the key.
- After that we delete all the levels which have no key.

Example: - 6 to be deleted here



This algorithm or data structure has O(log n) average time complexity for each operation while O(n) as the worst case time complexity.