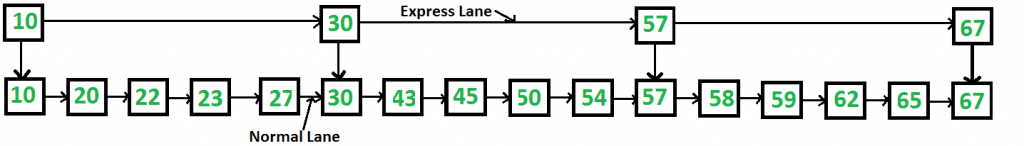
**Can we search in a sorted linked list in better than O(n) time?**  
The worst case search time for a sorted linked list is O(n) as we can only linearly traverse the list and cannot skip nodes while searching. For a Balanced Binary Search Tree, we skip almost half of the nodes after one comparison with root. For a sorted array, we have random access and we can apply Binary Search on arrays.

Can we augment sorted linked lists to make the search faster? The answer is [Skip List](http://en.wikipedia.org/wiki/Skip_list). The idea is simple, we create multiple layers so that we can skip some nodes. See the following example list with 16 nodes and two layers. The upper layer works as an “express lane” which connects only main outer stations, and the lower layer works as a “normal lane” which connects every station. Suppose we want to search for 50, we start from first node of “express lane” and keep moving on “express lane” till we find a node whose next is greater than 50. Once we find such a node (30 is the node in following example) on “express lane”, we move to “normal lane” using pointer from this node, and linearly search for 50 on “normal lane”. In following example, we start from 30 on “normal lane” and with linear search, we find 50.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/SkipList1.png)

**What is the time complexity with two layers?** The worst case time complexity is number of nodes on “express lane” plus number of nodes in a segment (A segment is number of “normal lane” nodes between two “express lane” nodes) of “normal lane”. So if we have n nodes on “normal lane”, √n (square root of n) nodes on “express lane” and we equally divide the “normal lane”, then there will be √n nodes in every segment of “normal lane” . √n is actually optimal division with two layers. With this arrangement, the number of nodes traversed for a search will be O(√n). Therefore, with O(√n) extra space, we are able to reduce the time complexity to O(√n).

We have already d iscussed the idea of Skip list and how they work in [Skip List | Set 1 (Introduction)](https://www.geeksforgeeks.org/skip-list/). In this article, we will be discussing how to insert an element in Skip list.

**Deciding nodes level**

Each element in the list is represented by a node, the level of the node is chosen randomly while insertion in the list. **Level does not depend on the number of elements in the node.** The level for node is decided by the following algorithm –

**randomLevel()**

lvl := 1

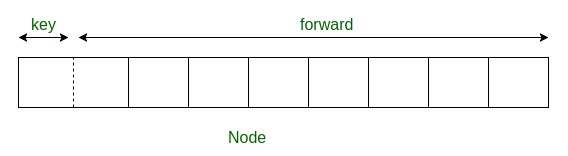
//random() that returns a random value in [0...1)

while random() < p and lvl < MaxLevel do

lvl := lvl + 1

return lvl

**MaxLevel** is the upper bound on number of levels in the skip list. It can be determined as – . Above algorithm assure that random level will never be greater than MaxLevel. Here **p** is the fraction of the nodes with level **i** pointers also having level **i+1** pointers and N is the number of nodes in the list.

Each node carries a key and a **forward** array carrying pointers to nodes of a different level. A level i node carries i forward pointers indexed through 0 to i.  


**Insertion in Skip List**

We will start from highest level in the list and compare key of next node of the current node with the key to be inserted. Basic idea is If –

1. Key of next node is less than key to be inserted then we keep on moving forward on the same level
2. Key of next node is greater than the key to be inserted then we store the pointer to current node **i** at **update[i]** and move one level down and continue our search.