Page No.
Date

Homework 11.1

following is the inorder traversal for multi way search tree

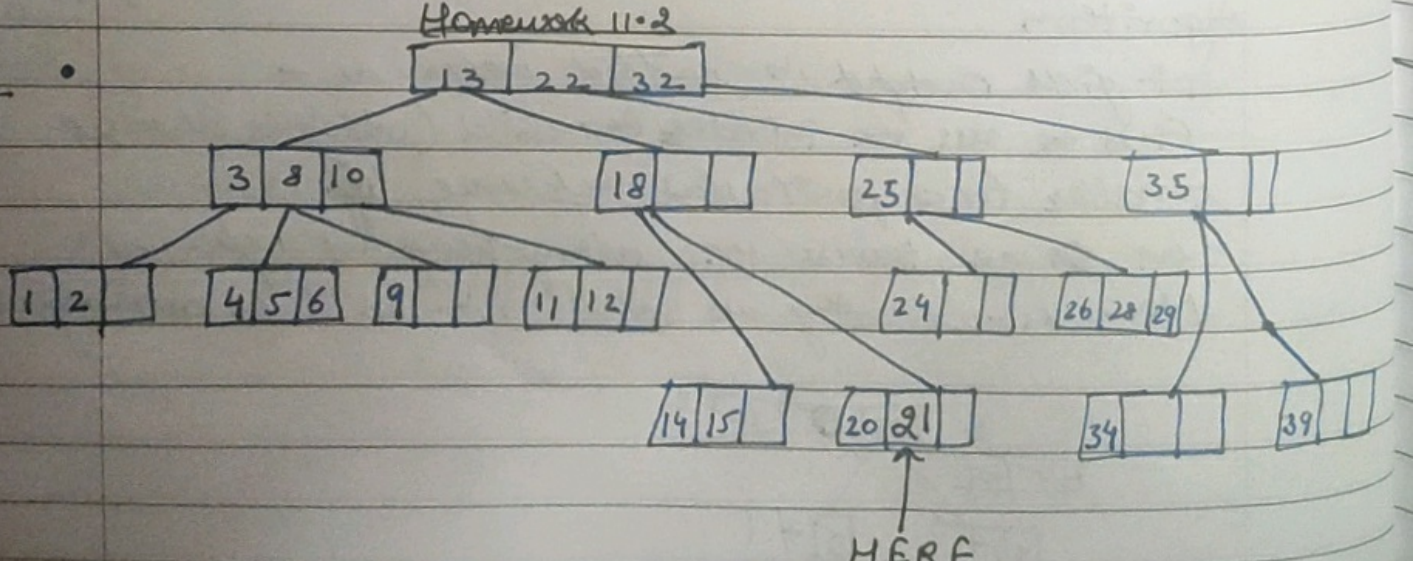
```
Algorithm inOrder ( node )
    if ( node = null )
        return
    else  inOrder ( node.link[0] ).
        for ( int i ← 0 ; i < d ; i++ ) // d is number of keys and
            visit key[i]                // consistent with Dr. Naveen lecture
            inOrder (node.link[i+1]).
```

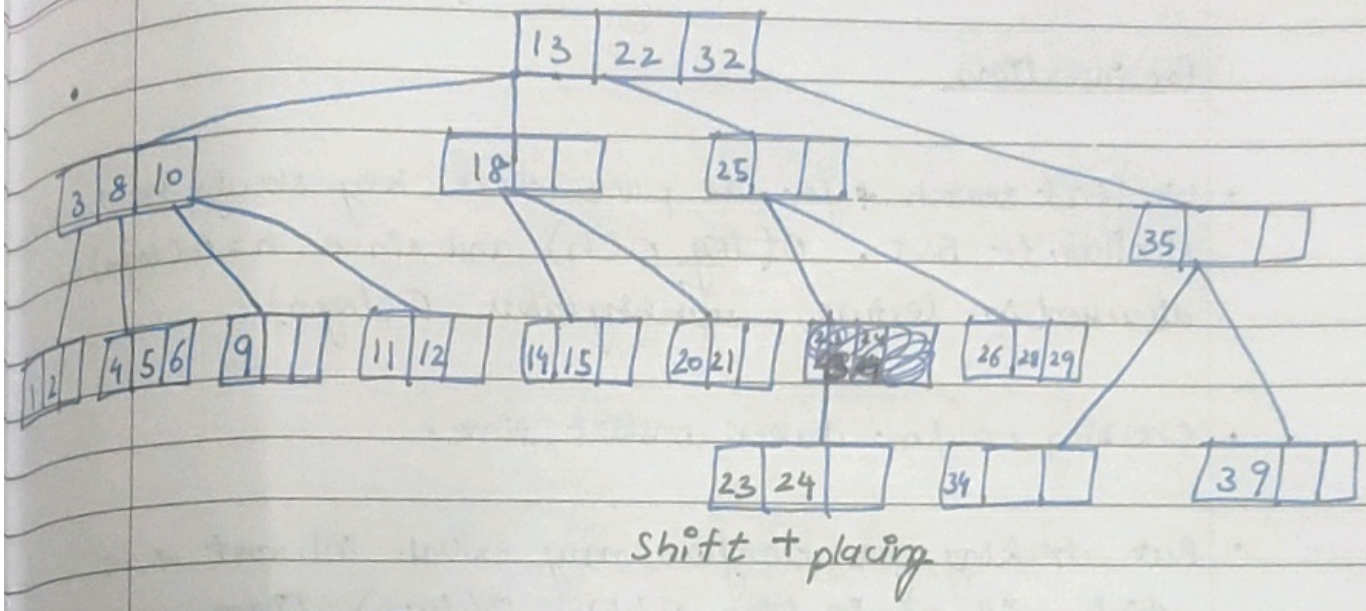
Note:
- Array link contains d+1 entries of respective links.
- Array key contains the keys in a node in sorted order.
- There are d+1 links, for d keys.

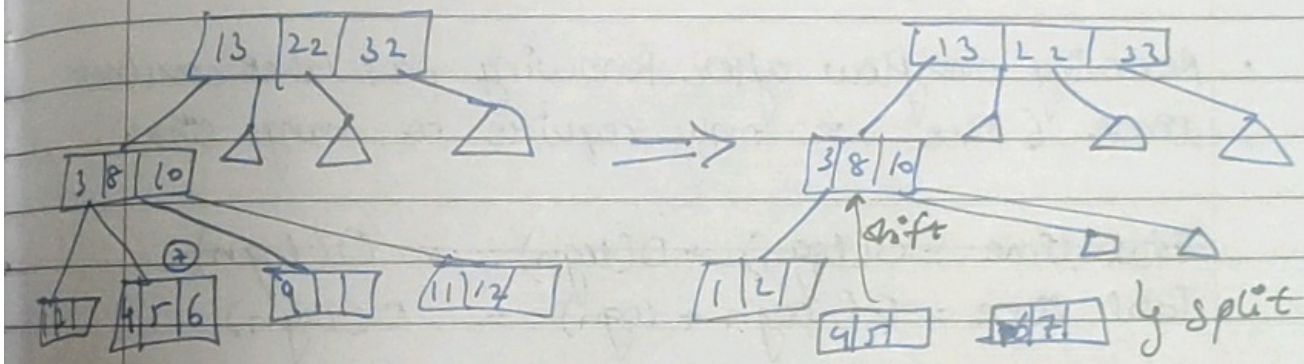
- first we call the inorder for child which is leftmost and thus the shortest.
- Now since keys are in sorted and all the links are in accordance to links, traversing the array of keys from left to right AND calling functions for corresponding links would yield the sorted output.

Homework 11.2

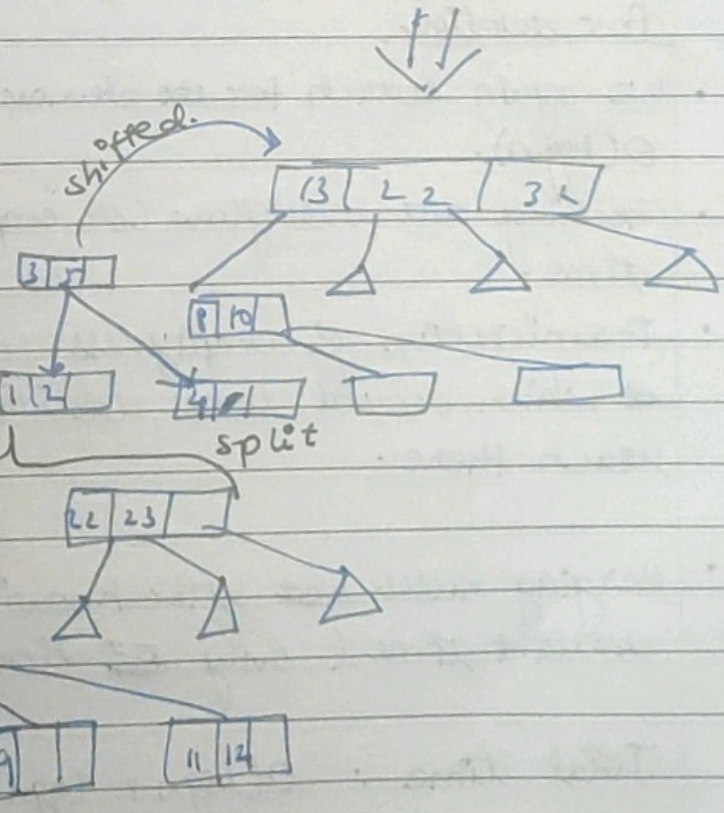
Ans:-







Note :-
I think accidently, maam put the same question as the one explained during lecture



Page No.
Date

Homework 11.3

for insertion

- We first search for the place where key should be inserted, similar to BST. $O(h)$ and since $h = O(\log n)$ as discussed in lecture, searching takes $O(\log n)$
- Checking overflow takes constant time.
- But checking for overflow may extend till root node which will again take $O(h) = O(\log n)$ time.
- Removing overflow after knowing place takes constant time (since we only require to change links).

~~Total time = $O(\log n) + O(\log n) = O(\log n)$~~

Total time = $O(\log n + \log n) = O(\log n)$

---

for deletion

- We again search for the element which should be removed. $O(\log n)$.
- checking underflow (or "emptyness") takes constant time.
- Termination of emptyness may occur all up to root which would take additional $O(\log n)$ time to reach there.
- Merging nodes after knowing the place will take constant time (only link changing is required).

Total time : $O(\log n + \log n) = O(\log n)$