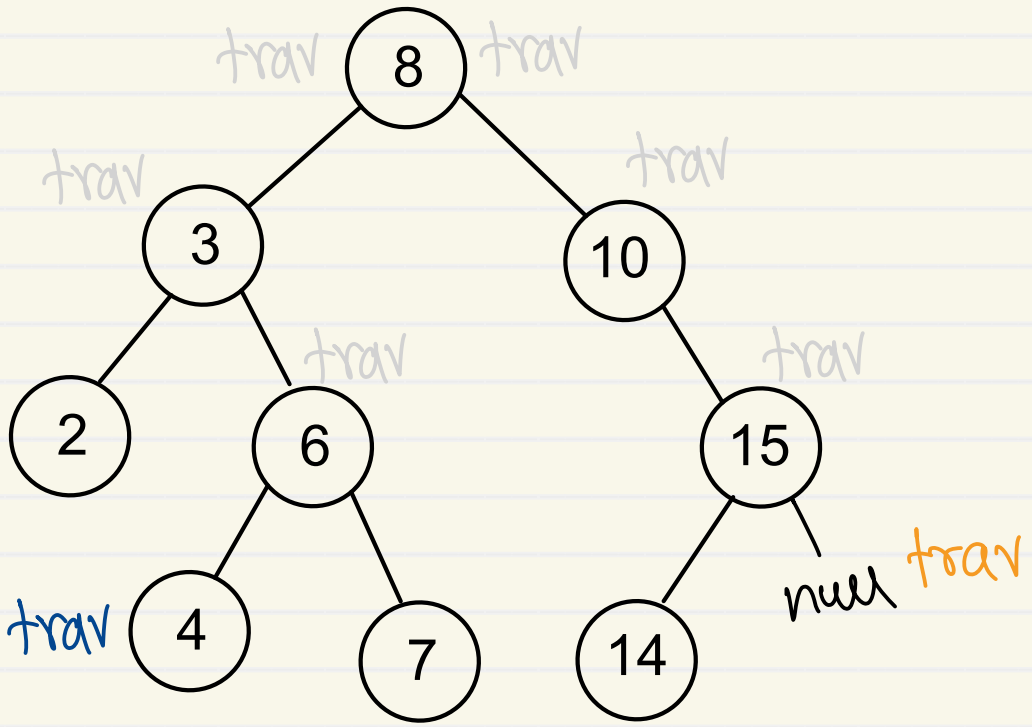
Sunbeam Institute of Information Technology
Pune and Karad

Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

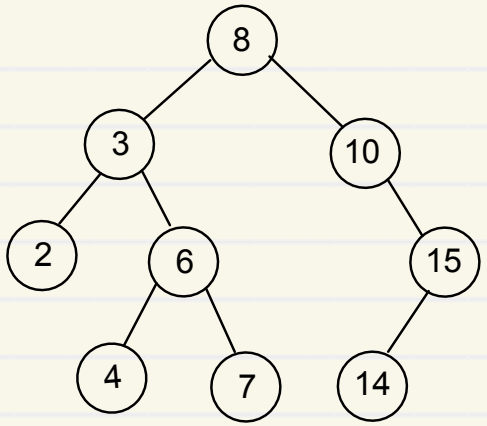
Binary Search Tree - Binary Search



1. Start from root
2. If key is equal to current node data return current node
3. If key is less than current node data search key into left sub tree of current node
4. If key is greater than current node data search key into right sub tree of current node
5. Repeat step 2 to 4 till leaf node

key = 4 ← key is found

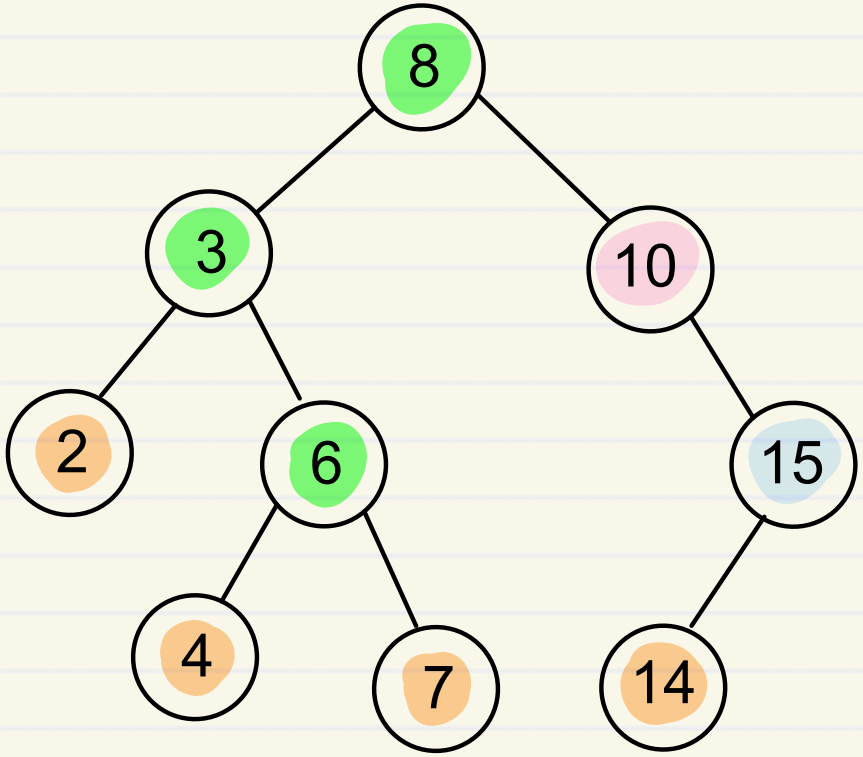
key = 20 ← key is not found

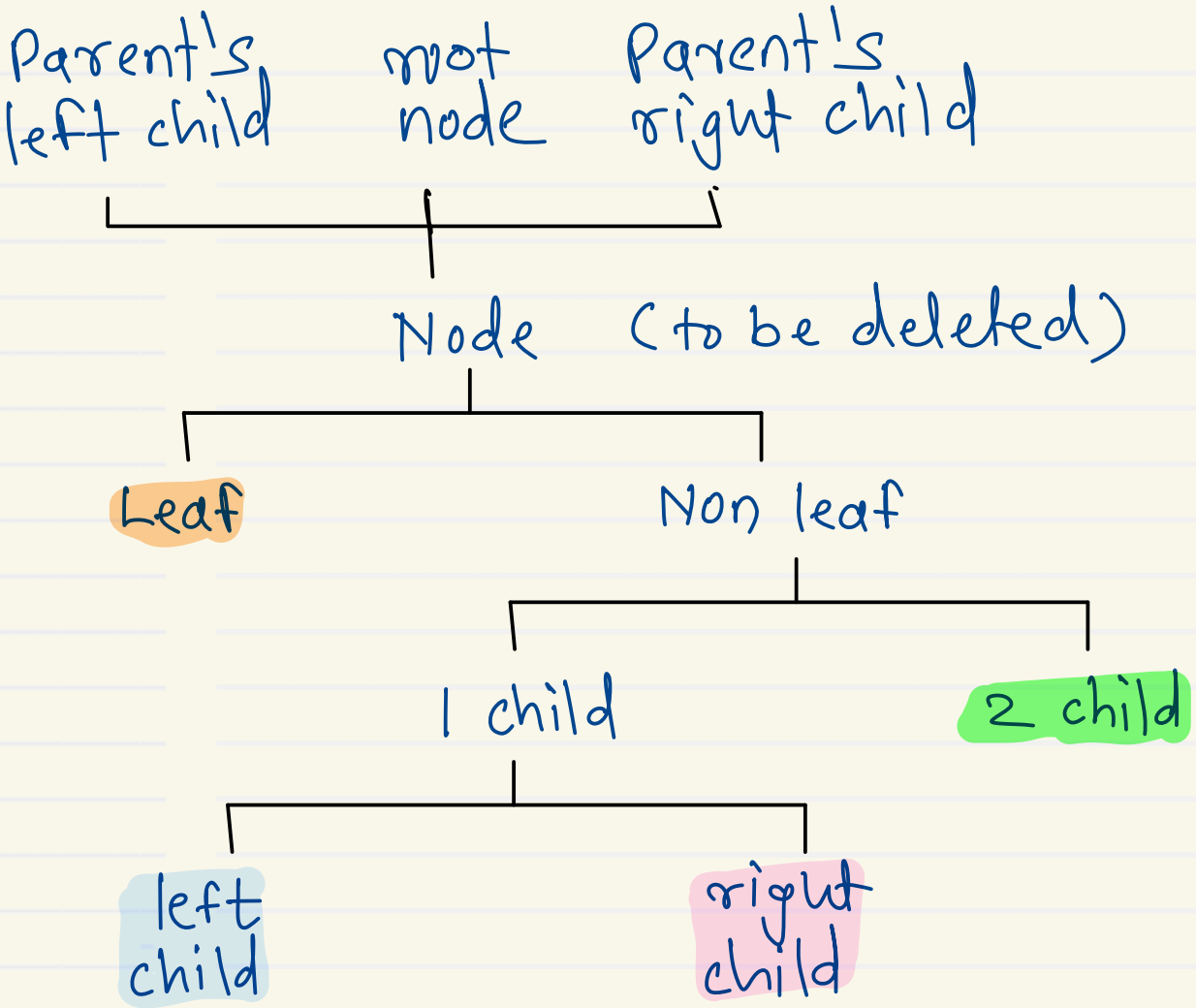


binarySearchRec(&8, 4)
binarySearchRec(&3, 4)
binarySearchRec(&6, 4)
binarySearchRec(&4, 4)

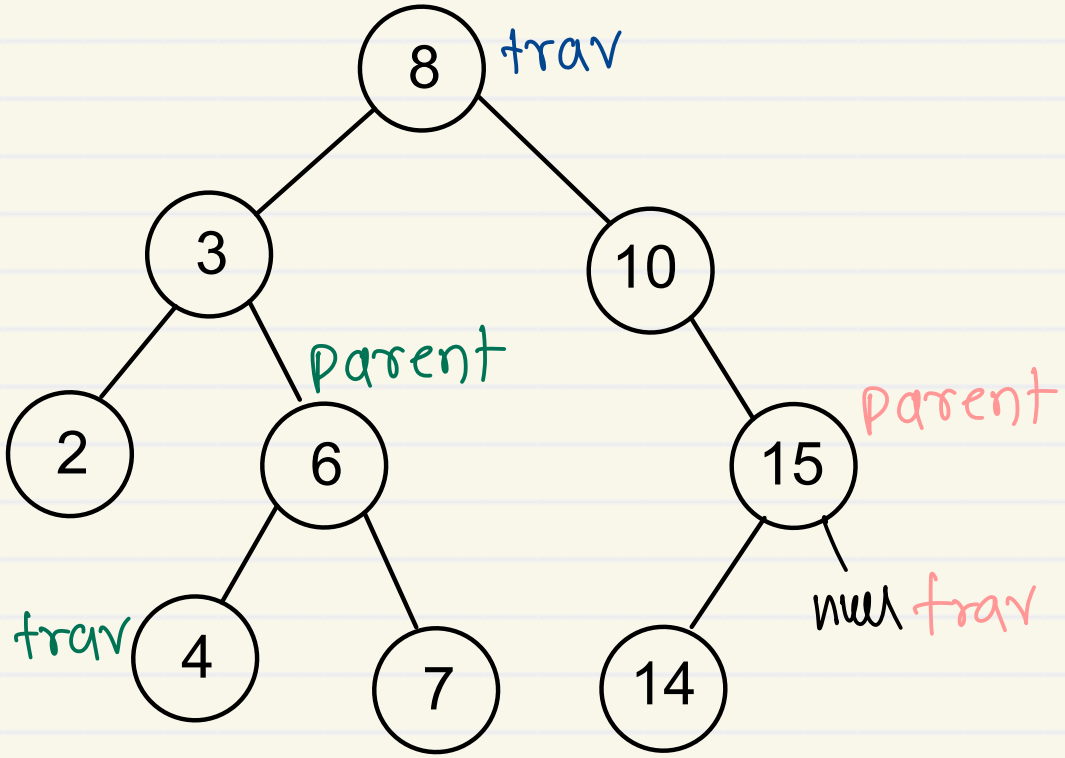
```
Node binarySearchRec(Node trav, int key){
 if(trav == null)
     return null;
 if(key == trav.data)
     return trav;

 if(key < trav.data)
    return binarySearchRec(trav.left, key);
else
    return binarySearchRec(trav.right, key);

}
```

Binary Search Tree - Delete Node





Binary Search Tree - Binary Search with Parent



key = 4

trav	parent
&8	null
&3	&8
&6	&3
&4	&6

key = 8

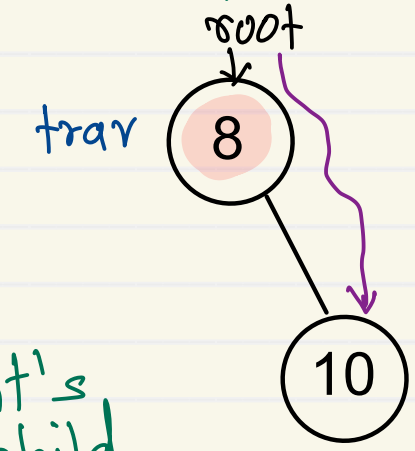
trav	parent
&8	null

key = 20

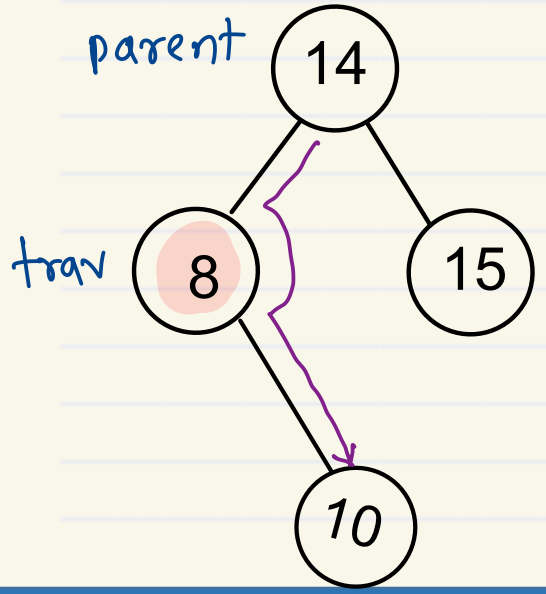
trav	parent
&8	null
&10	&8
&15	&10
null	&15

BST - Delete Single child node (Right child)

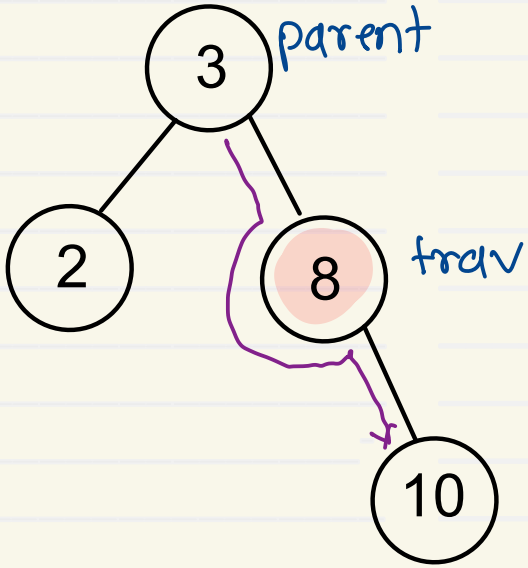
a. root node



b. Parent's left child

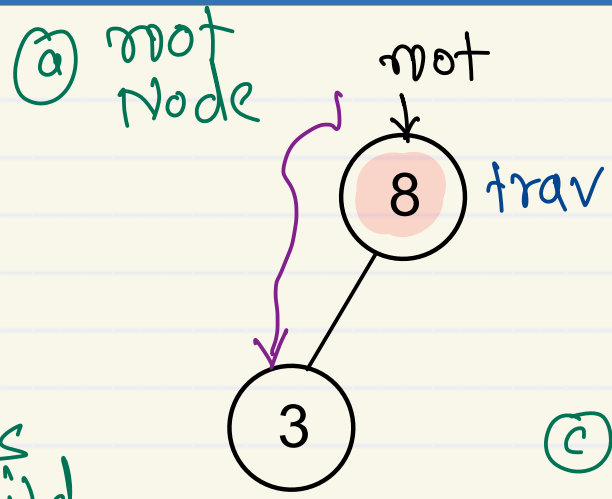


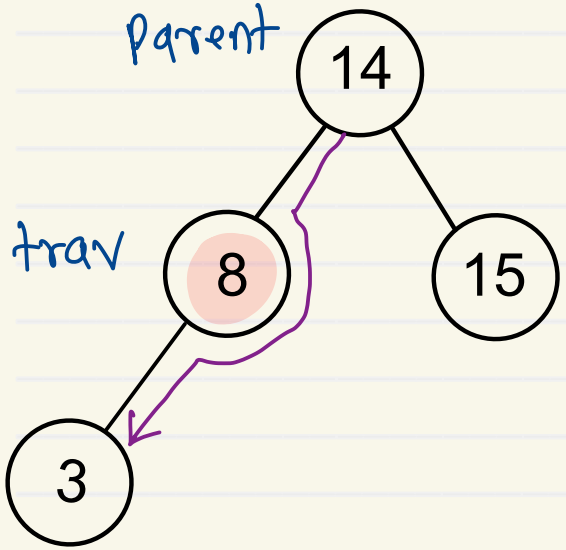
c. Parent's right child

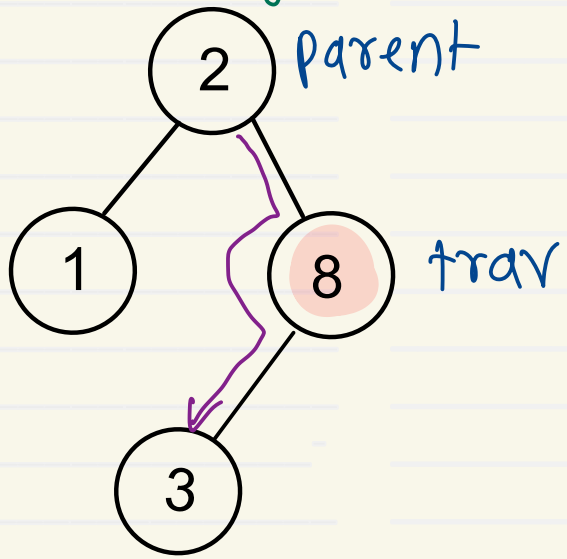



```
if( trav. left == null ) {
    if( trav == root )
(a)     root = trav. right;
    else if( trav == parent. left )
(b)     parent. left = trav. right;
    else if( trav == parent. right )
(c)     parent . right = trav. right;
}
```

BST- Delete Single child node (Left child)

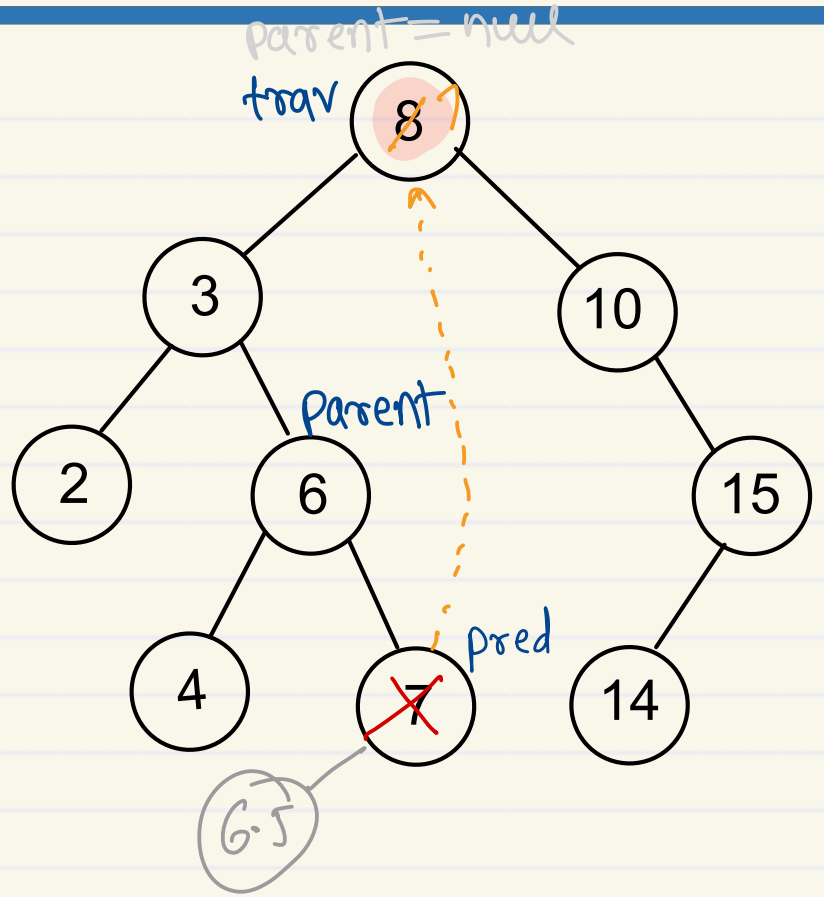







```
if ( trav.right == null ) {
    if( trav == root)
(a)     root = trav.left ;
    else if (trav == parent.left)
(b)     parent.left = trav.left ;
    else if ( trav == parent.right)
(c)     parent.right = trav.left ;
}
```

BST - Delete Two childs node



- trav.left & trav.right both are not null

1. find inorder predecessor of a node (trav)
2. replace node data (trav.data) by predecessor data (pred.data)
3. delete inorder predecessor

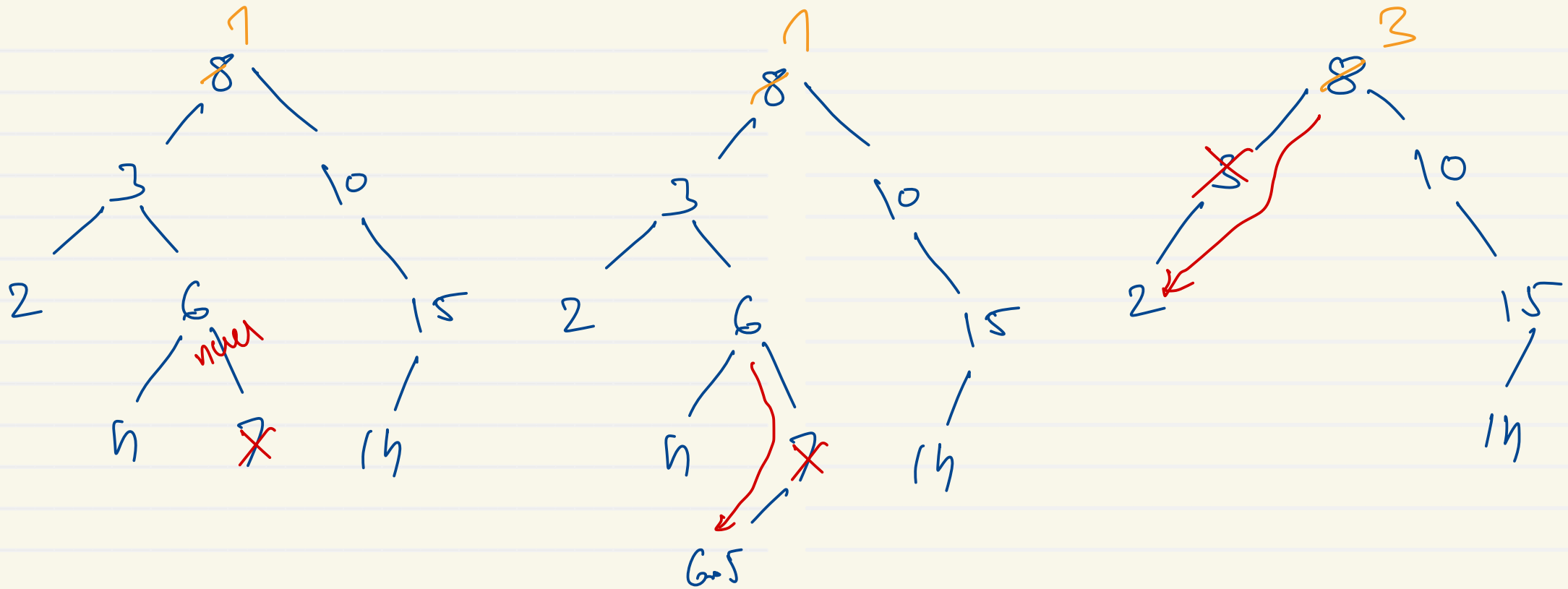
pred	parent
&3	&8
&6	&3
&7	&6

```
pred = trav.left;
parent = trav;
while(pred.right != null){
    parent = pred;
    pred = pred.right;
}
```

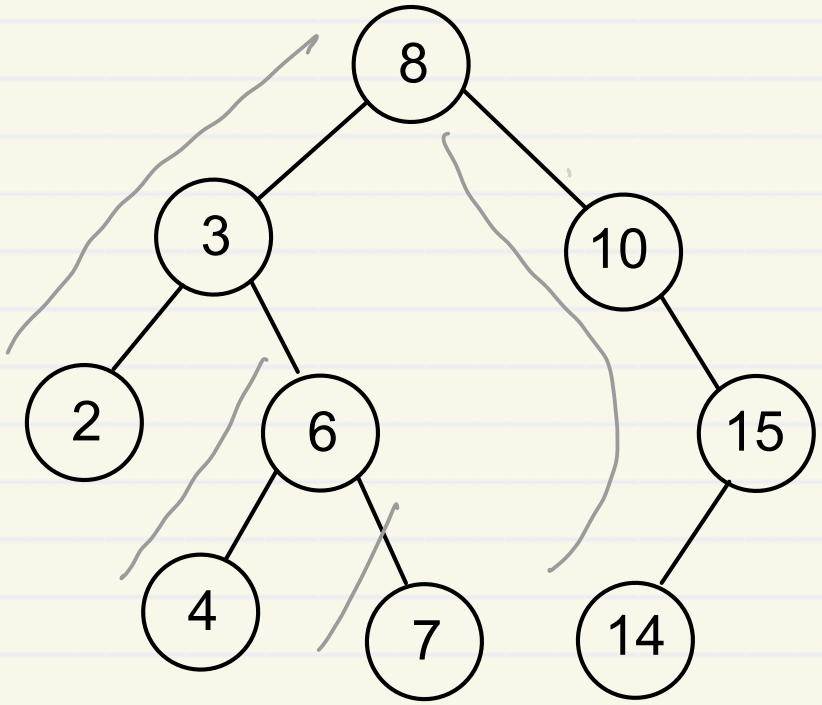
Inorder : 2 3 4 6 7 8 10 14 15

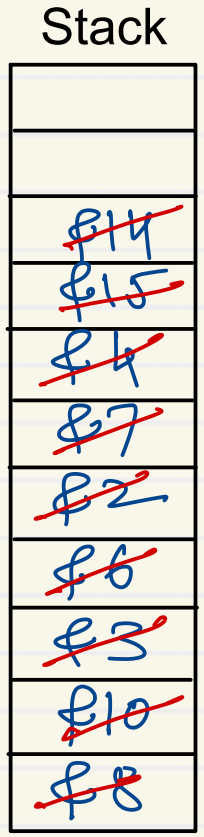
left sub tree extreme right → inorder predecessor

inorder successor ← right sub tree extreme left


1
8
3
10
2
6
15
new
5
7
14
1
8
3
10
2
6
15
5
7
14
6.5
3
8
5
10
2
15
14

Binary Search Tree - DFS Traversal (Depth First Search)

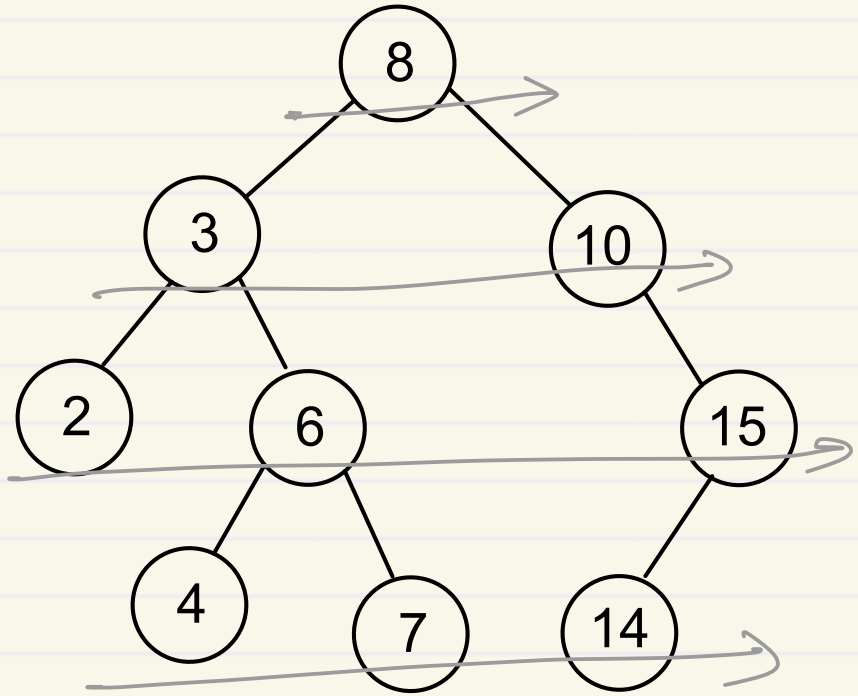


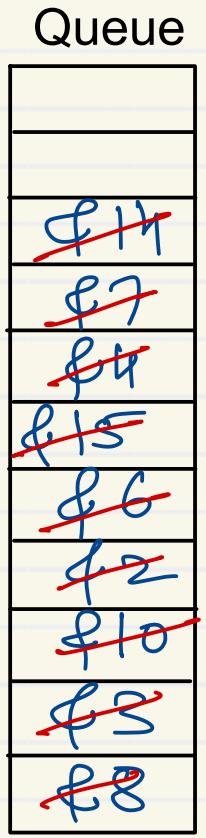


1. Push root node on stack
2. Pop one node from stack
3. Visit (print) popped node
4. If right exists, push it on stack
5. If left exists, push it on stack
6. While stack is not empty, repeat step 2 to 5

DFS Traversal : 8, 3, 2, 6, 4, 7, 10, 15, 14

Binary Search Tree - BFS Traversal (Breadth First Search)





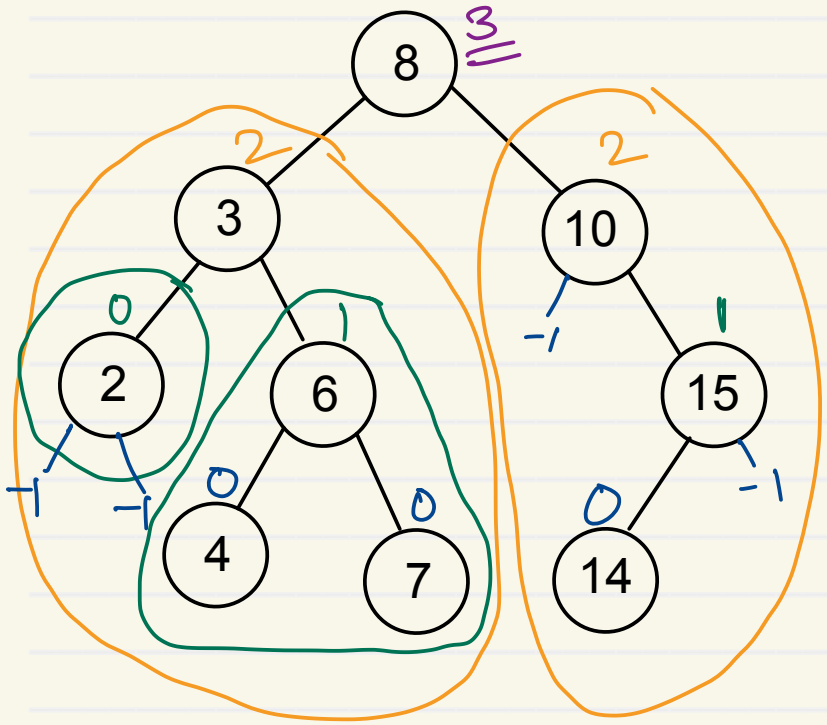
1. Push root node on queue
2. Pop one node from queue
3. Visit (print) popped node
4. If left exists, push it on queue
5. If right exists, push it on queue
6. While queue is not empty, repeat step 2 to 5

(level order traversal)

BFS Traversal : 8, 3, 10, 2, 6, 15, 4, 7, 14

Binary Search Tree - Height

Height of root = MAX (height (left sub tree), height (right sub tree)) + 1



1. If left or right sub tree is absent then return -1
2. Find height of left sub tree
3. Find height of right sub tree
4. Find max height
5. Add one to max height and return

```
int height( Node trav) {
    if(trav == null)
        return -1;
    int hl = height(trav.left);
    int hr = height(trav.right);
    int max = hl > hr ? hl : hr;
    return max+1;
}
```