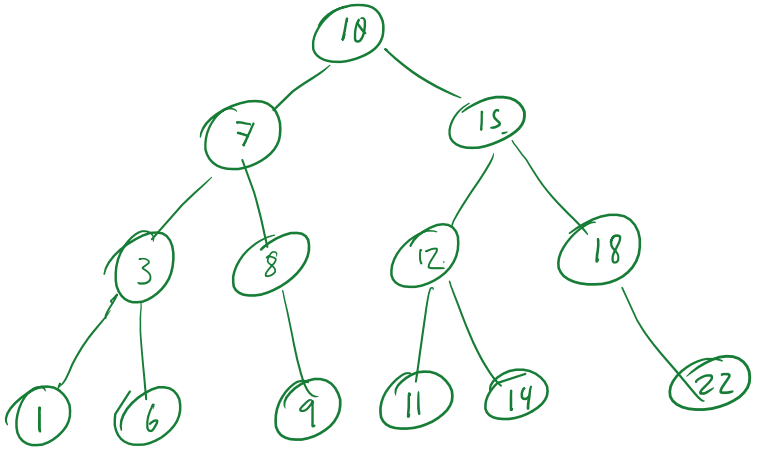
The ADT Binary Search Tree and its implementations.

Monday, November 4, 2019 5:17 PM

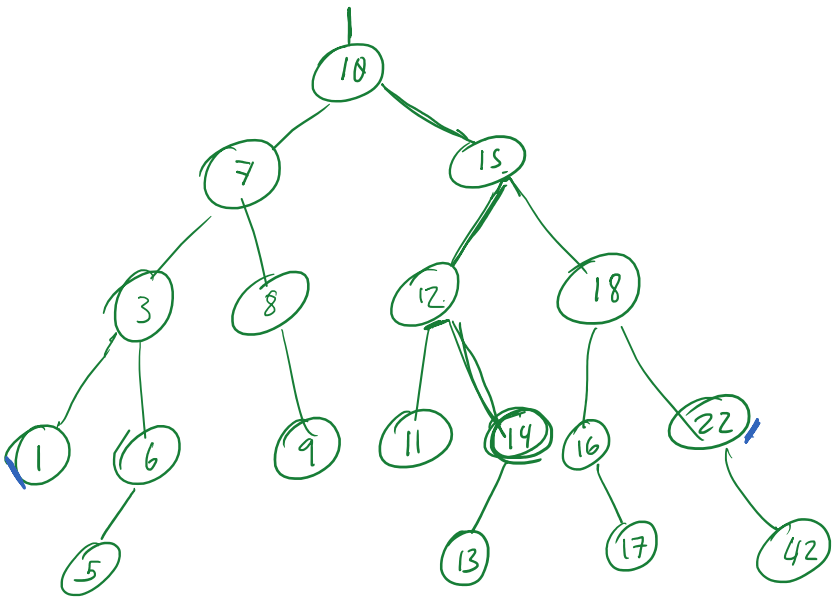
- Binary:- The tree is of degree 2.
- Distinguish Left from Right subtree
- The elements in the tree are comparable.
- Search property:-
  for every node n in the tree
  - n is greater than any node in n's left subtree
  - n is lesser than any node in n's right subtree

  (duplicates not allowed.)



Operations:

- ★ find (x, T)
- getMin (T)
- getMax (T)
- insert (x, T)
- remove (x, T)



find(x,t) : req as many comparisons as height of tree.

Average height is $\log_2(n)$: n number of elements in tree

	n	$\log_2(n)$
5000.	5000	13
5,000,000	5,000,000	23

remove (x, T)

case 1: X has no children

Example 13

Case 2: X has one child.

Example 16

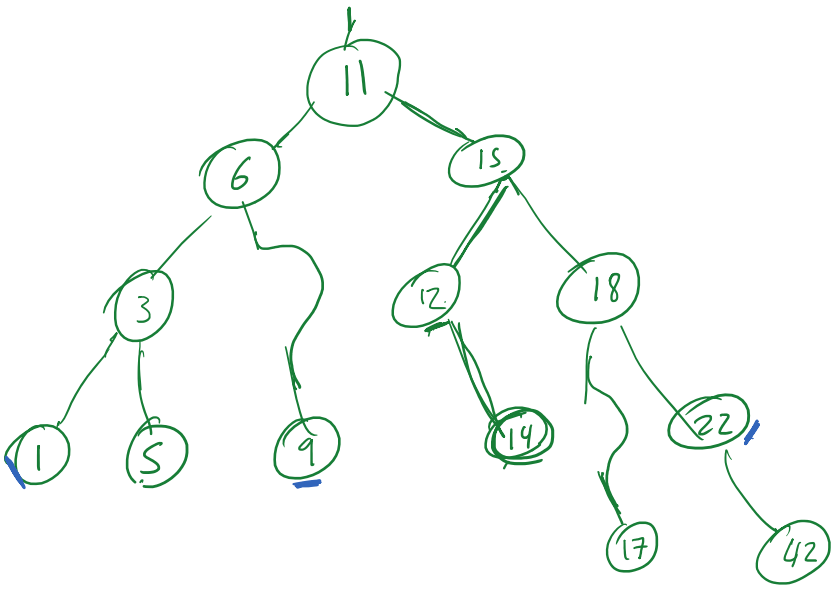
Example 8

put child under parent of X

Case 3: X has 2 children.

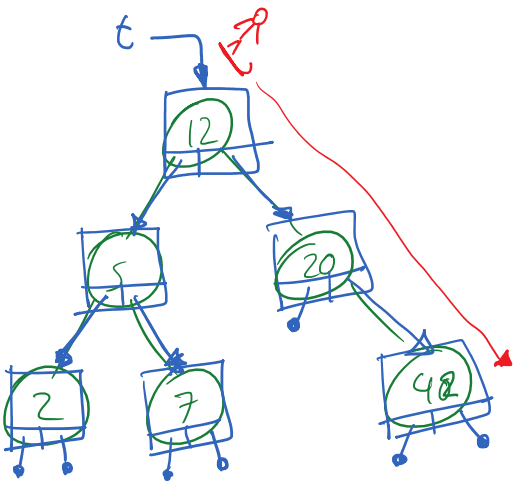
Example 10.

Example 7.



The Data Structure TreeNode

```
template <typename T>
class TreeNode{
public:
   T m_data;
   TreeNode<T>* m_left;
   TreeNode<T>* m_right;
};
```



• find

```
bool find( TreeNode* t, T& x)
{
   if (t==NULL) return false;
   if (t->m_data == x ) return true;
   if (x < t->m_data )
      return find( t->m_left, x );
   else // ( x > t->m_data )
      return find( t->m_right, x );
}
```

• getMin [Recursive]

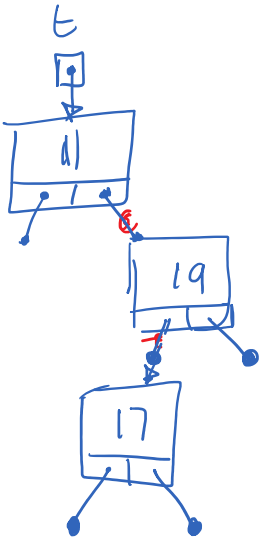
```
TreeNode* getMin( TreeNode* t )
{
   if (t==NULL) return NULL;
   if (t->m_left == NULL)
       return t;
   else
       return getMin( t->m_left )
}
```

• getMax [Iterative]

```
TreeNode* getMax( TreeNode* t )
{
   if (t==NULL) return NULL;
   TreeNode* tmp = t;
   while (tmp->m_right != NULL)
      tmp = tmp->m_right;
   return tmp;
}
```

• Insert

```
void insert( TreeNode* &t, T& x )
{
   if ( t == NULL )
      t = new TreeNode(x, NULL, NULL);
   else
      if ( x < t->m_data )
         insert( t->m_left , x)
      else if ( x > t->m_data )
         insert( t->m_right, x)
      else // x == t->m_data
         return;
}
```



Remove

```
void remove( TreeNode* &t, T& x )
{
   if( t==NULL ) return;
   if( x < t->m_data)
```

My t

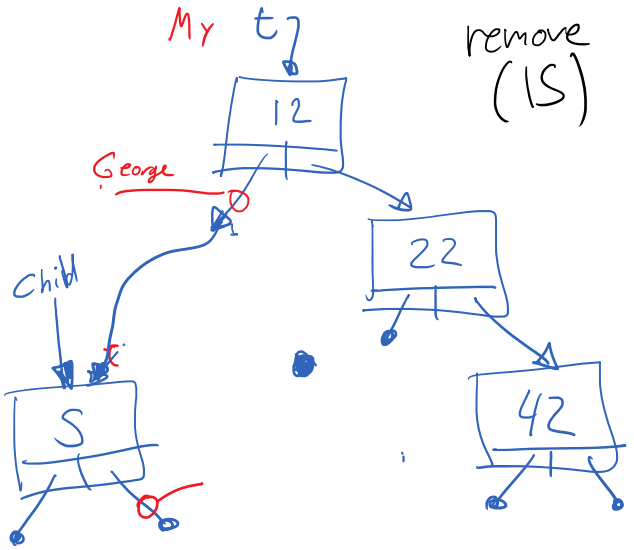
remove (15)

```
void remove( TreeNode* &t, T& x )
{
   if( t==NULL ) return;
   if( x < t->m_data)
      remove( t->m_left, x)
   else if ( x > t->m_data)
      remove( t->m_right, x)
   else // x == t->m_data
   {

         // no children
      if (t->mleft == NULL &&
          t->m_right == NULL){
          delete t;
          t = NULL;
      }
            // one-child
      else if (t->mleft == NULL ||
               t->m_right == NULL){

          TreeNode* child = t->m_left;
          if ( child == NULL )
             child = t->m_right;

          delete t;
          t = child;
      }
      else // two children
      {
          t->m_data = getMax( t->m_left );
          remove( t->m_left, t->m_data );
      }
   }
```

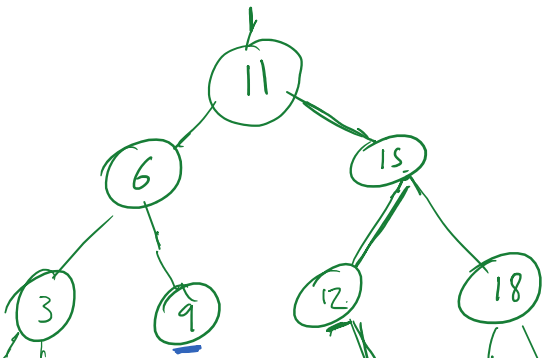


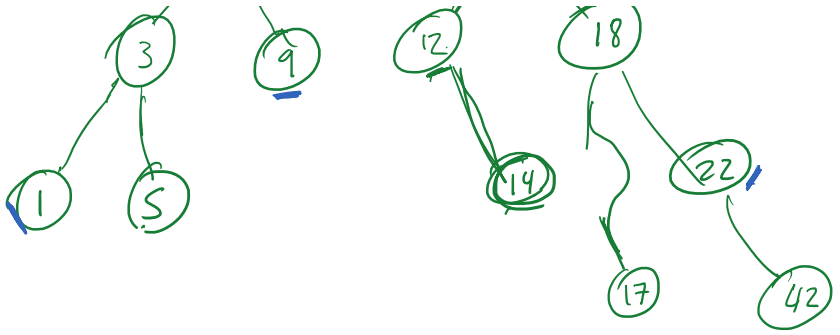
Motivation

find	$\log_2 n$
Insert	$\log_2 n$
remove	$\log_2 n$

height $\approx \log_2 n$

The Data Structure ArrayTree





0	1	2	3	4	5	6	7	8	9	10	11	12	13	14							
11	6	15	3	9	12	18	1	5	-	-	-	14	17	22	-	-	-	-	-	...	42

- Cons: space may be wasted
- pro: $left(i) = 2 \cdot i + 1$

  $right(i) = 2 \cdot i + 2$

  $parent(i) = \left\lfloor \frac{i-1}{2} \right\rfloor$