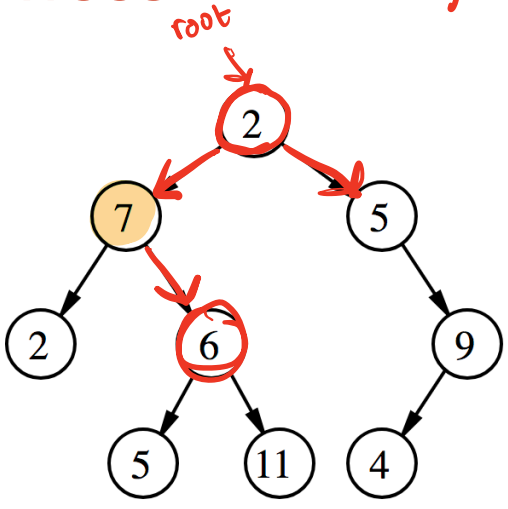
BINARY SEARCH TREES

Problem Solving with Computers-II

Trees

Hierarchy



A tree has following general properties:

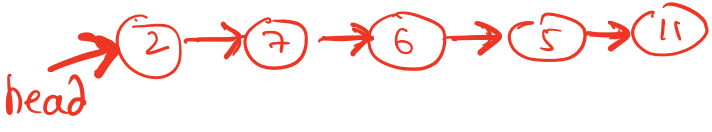
- One node is distinguished as a **root**;
- Every node (exclude a root) is connected by a directed edge *from* exactly one other node;
  A direction is: *parent -> children*
- *Leaf node: Node that has no children*

2's children are 7 and 5

Parent of 7 is 2

" " 5 " 2

Linked List (Linear)



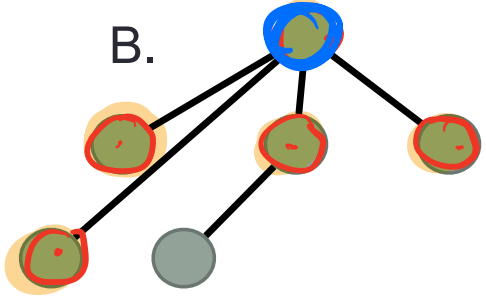
Which of the following is/are a tree?

Empty root

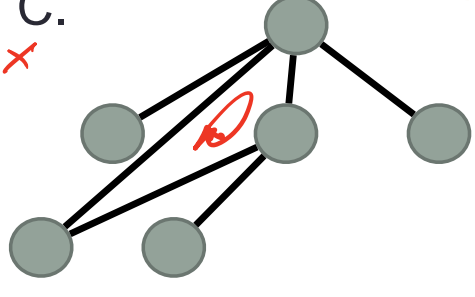
A.

Single node tree.

B.



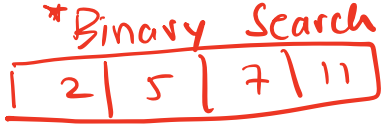
C.



D. A & B

E. All of A-C

Binary Search Trees (BST)



1. What are the operations supported?

All the operations supported by sorted arrays + fast insert and delete

2. What are the running times of these operations?

efficient

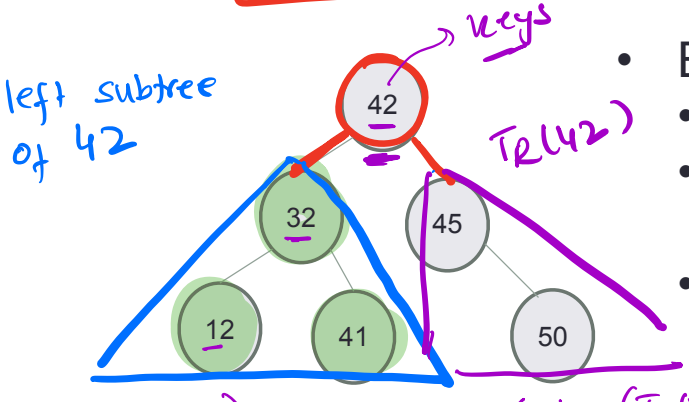
3. How do you implement the BST i.e. operations supported by it?

Sorted arrays vs Binary Search Trees (BST)

Operations
Min
Max
Successor
Predecessor
Search
Insert
Delete
Print elements in order

Binary Search Tree – What is it?

Binary Trees



- Each node:
  - stores a key (k)
  - has a pointer to left child, right child and parent (optional)
  - Satisfies the Search Tree Property

For any node,
Keys in node's left subtree $\leq$ Node's key
Node's key < Keys in node's right subtree

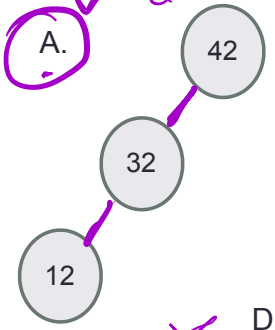
$key\ T_L(x) < key(x) < key\ T_R(x)$

Do the keys have to be integers?

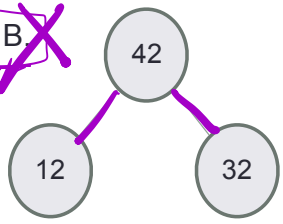
Which of the following is/are a binary search tree?

BST that looks like a linked list

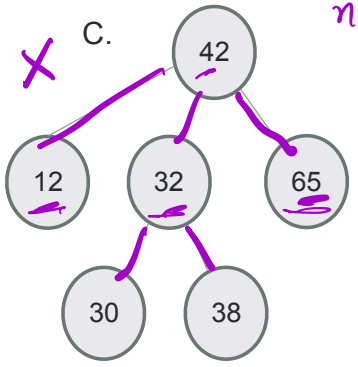
A.



B.

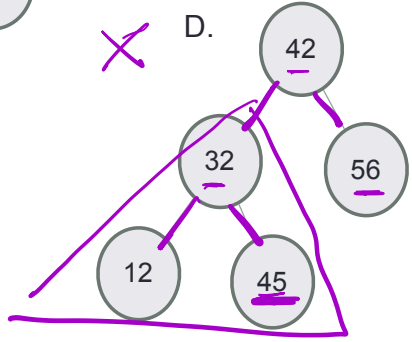


C.



not a binary tree

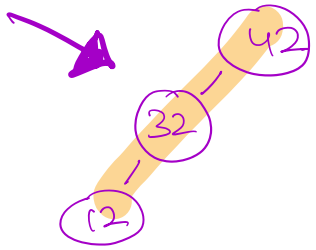
D.



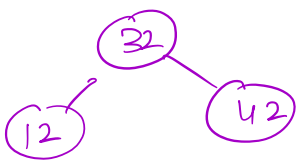
45 is more than 42 but it is in 42's left subtree

E. More than one of these

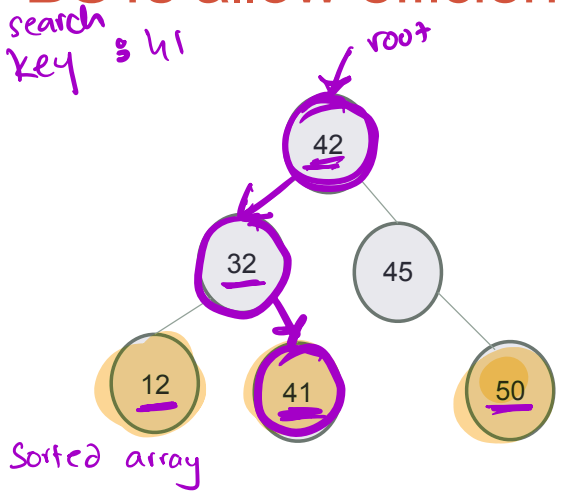
Insert: 42, 32, 12

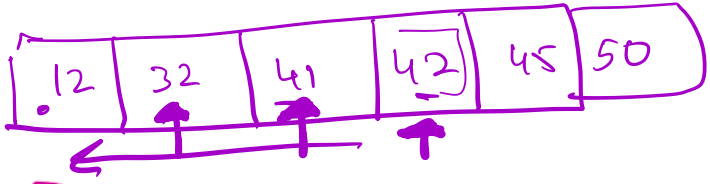


32, 42, 12



BSTs allow efficient search!





- Start at the root;
- Trace down a path by comparing **k** with the key of the current node x:
  - If the keys are equal: we have found the key
  - If **k** < key[x] search in the left subtree of x
  - If **k** > key[x] search in the right subtree of x

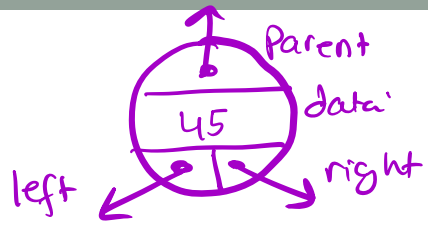
Search for 41, then search for 53

A node in a BST

```
class BSTNode {

public:
  BSTNode* left;
  BSTNode* right;
  BSTNode* parent;
  int const data;

  BSTNode( const int & d ) : data(d) {
    left = right = parent = nullptr;
  }
};
```



45

node in a linked list

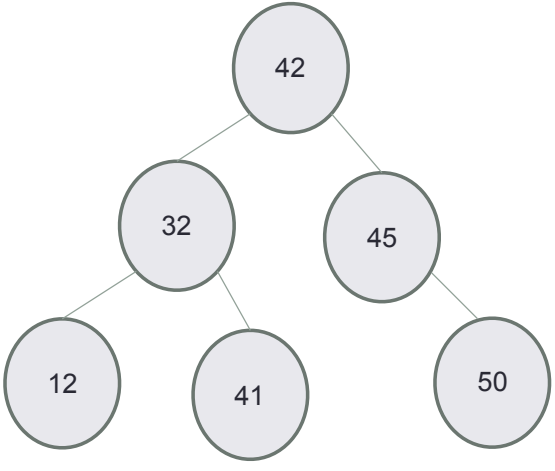
initializer list

45

BSTNode *n = new BSTNode {45};

Define the BST ADT

Operations
Search
Insert
Min
Max
Successor
Predecessor
Delete
Print elements in order



Traversing down the tree

BSTNode* n = root;

- Suppose n is a pointer to the root. What is the output of the following code:

```
n = n->left;
n = n->right;
cout<<n->data<<endl;
```

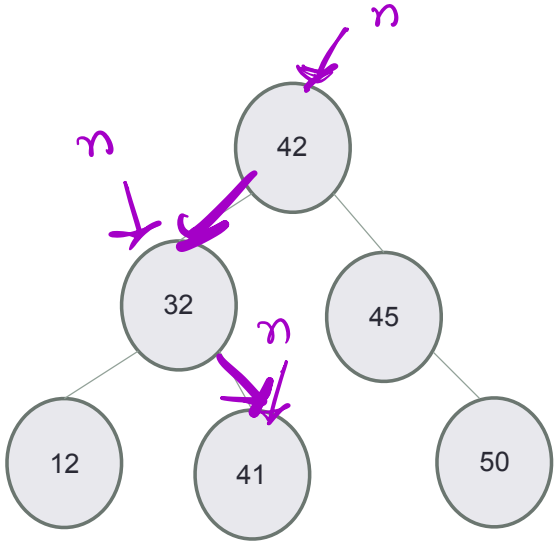
A. 42

B. 32

C. 12

D. 41

E. Segfault

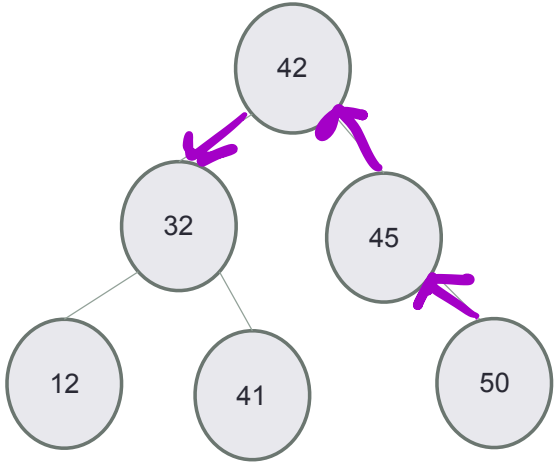


Traversing up the tree

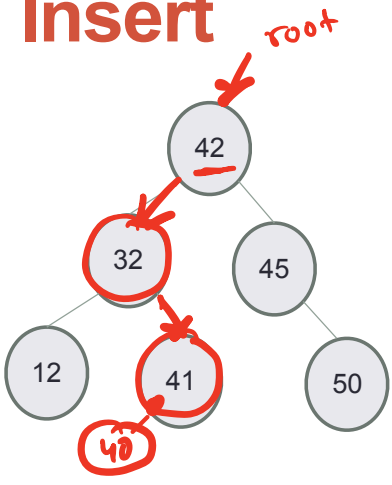
- Suppose n is a pointer to the node with value 50.
- What is the output of the following code:

```
n = n->parent; //
n = n->parent;
n = n->left;
cout<<n->data<<endl;
```

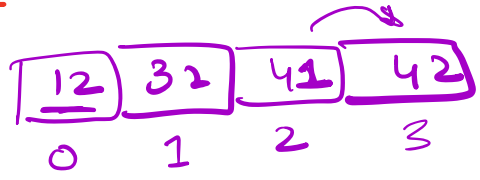
A. 42
B. 32
C. 12
D. 45
E. Segfault



Insert



- Insert 40
- Search for the key
- Insert at the spot you expected to find it



In the case of ~~for~~ sorted array we would need to move the elements over on each insert (we might need to move all elements in the worst case)

The above BST is obtained by inserting keys in the following order:

42, 32, 41, 12, 45, 50