RECURSION
GDB
BINARY SEARCH TREES

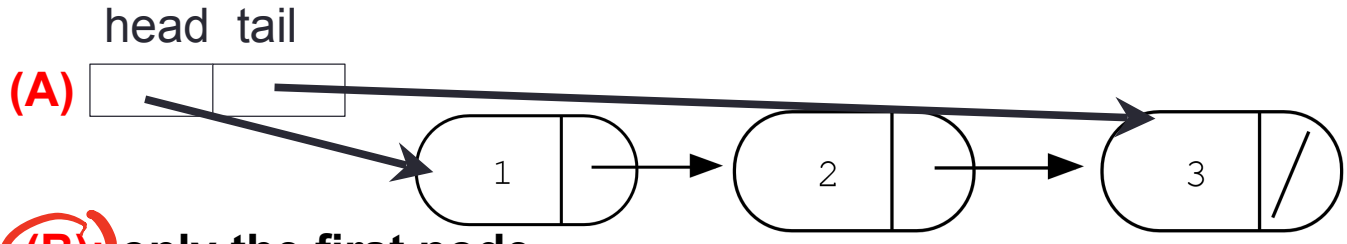
Problem Solving with Computers-II

Concept Question

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
class Node {
    public:
       int info;
       Node *next;
};
```

Which of the following objects are deleted when the destructor of Linked-list is called?



(B): only the first node

(C): A and B

(D): All the nodes of the linked list

(E): A and D

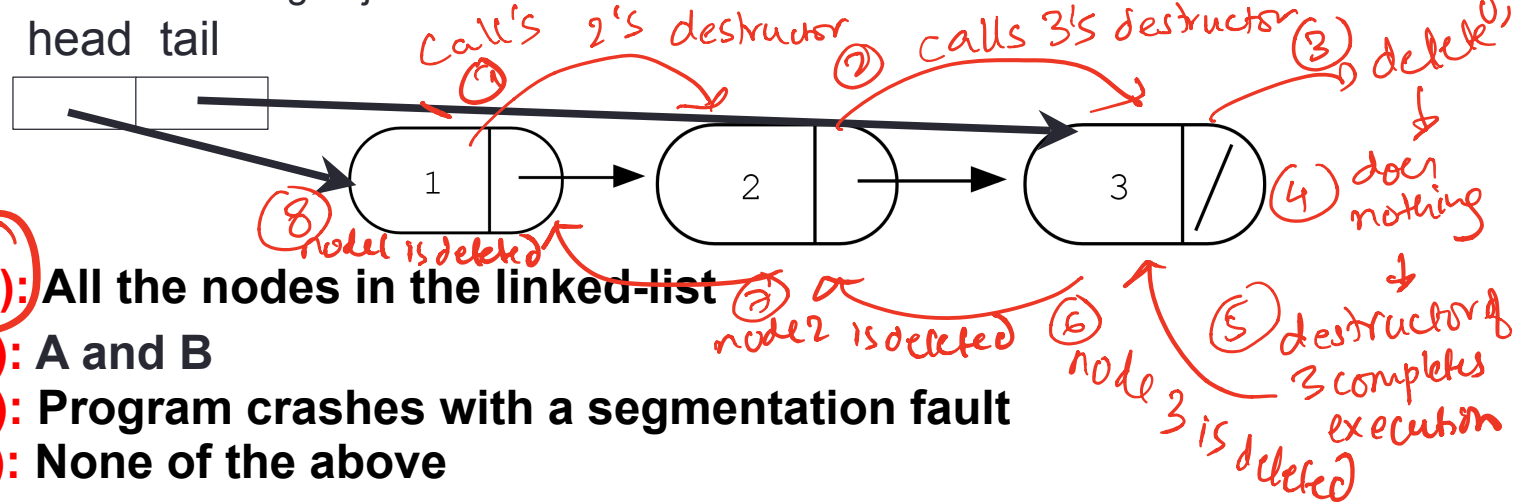
Concept question

```
LinkedList::~LinkedList(){
   delete head;
}
```

This calls the destructor of Node for node 1

```
Node::~Node(){
   delete next;
}
```

Which of the following objects are deleted when the destructor of Linked-list is called?



(B): All the nodes in the linked-list

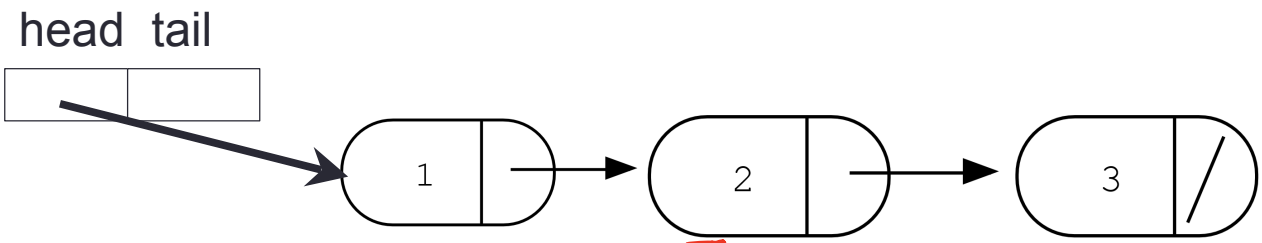
(C): A and B

(D): Program crashes with a segmentation fault

(E): None of the above

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```



p

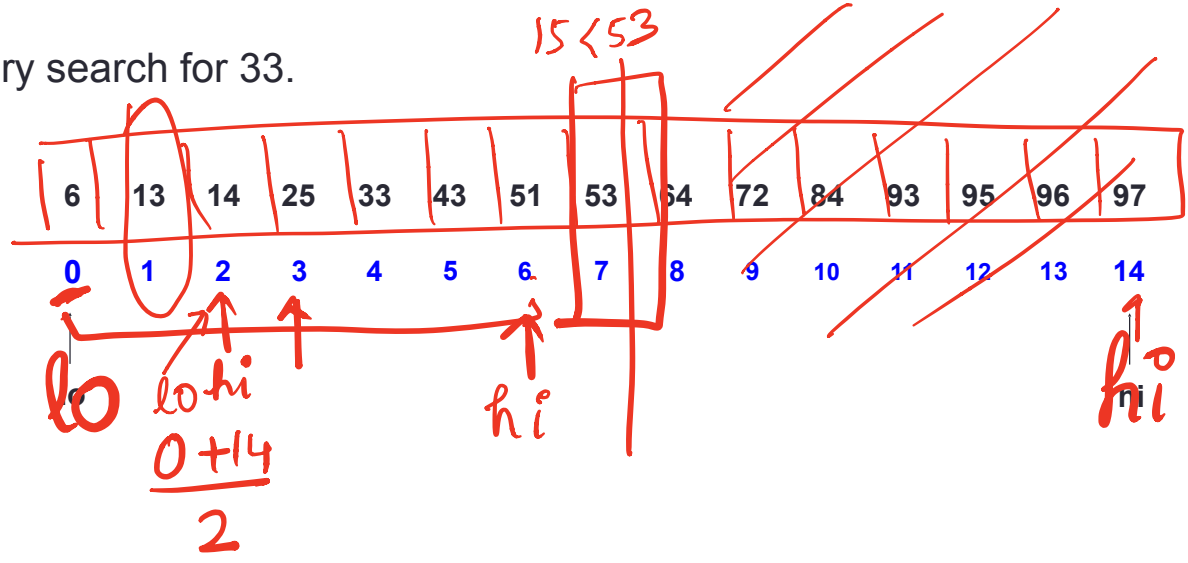
If you only want to delete node 2, first isolate it, (set its next pointer to null) then call delete.

what happens if we say delete p;

→ Part of the chain starting at node 2 is deleted

Binary Search

- Binary search. Given `value` and sorted array `a[]`, find index `i` such that `a[i]` = `value`, or report that no such index exists.
- Invariant. Algorithm maintains `a[lo]` ≤ `value` ≤ `a[hi]`.
- Ex. Binary search for 33.



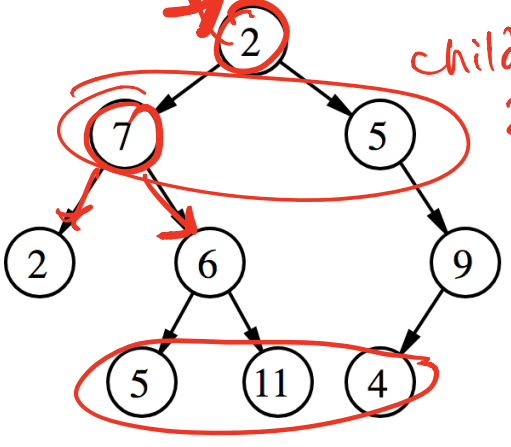
GDB: GNU Debugger

- To use gdb, compile with the -g flag
- Setting breakpoints (b)
- Running programs that take arguments within gdb (r arguments)
- Continue execution until breakpoint is reached (c)
- Stepping into functions with step (s)
- Stepping over functions with next (n)
- Re-running a program (r)
- Examining local variables (info locals)
- Printing the value of variables with print (p)
- Quitting gdb (q)
- Debugging segfaults with backtrace (bt)

Demo debugging using gdb.

* Refer to the gdb cheat sheet: https://ucsb-cs24.github.io/m19/lectures/GDB-cheatsheet.pdf

Trees



one field in a larger struct (record)
one piece of data
hierarchy
children of 2
leaf nodes

A tree has following general properties:

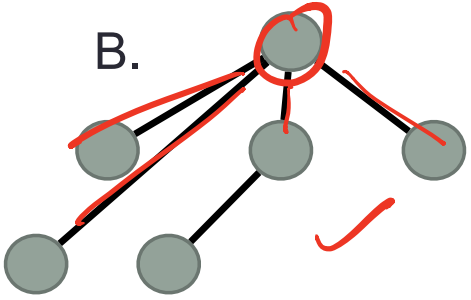
- One node is distinguished as a **root**;
- Every node (exclude a root) is connected by a directed edge *from* exactly one other node;
  A direction is: *parent -> children*
- *Leaf node: Node that has no children*

Which of the following is/are a tree?

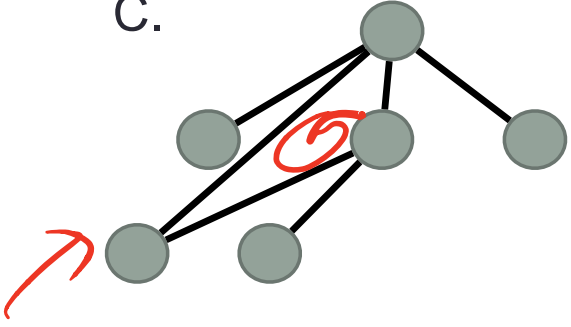
A.

Single node tree

B.



C.



D. A & B

E. All of A-C

Binary Search Trees

- What are the operations supported?

  Same as sorted array + fast insert & delete

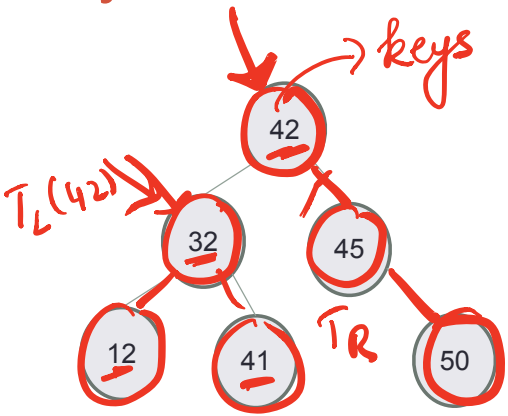
- What are the running times of these operations?

  This class we'll build some intuition, formalize next lecture

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

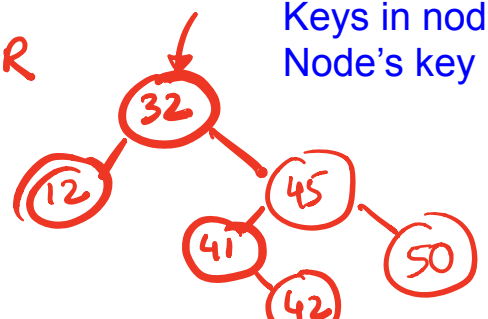
  We'll go over the important operations. This exercise improves your coding skills

Binary Search Tree – What is it?



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

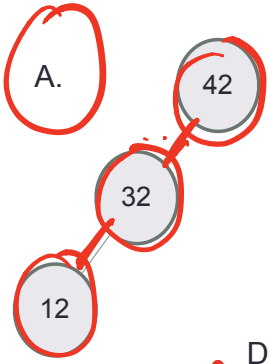
For any node,
Keys in node's left subtree <= Node's key
Node's key < Keys in node's right subtree



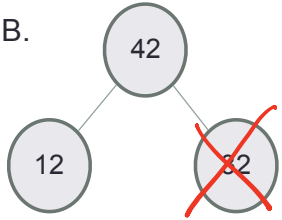
Do the keys have to be integers?

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

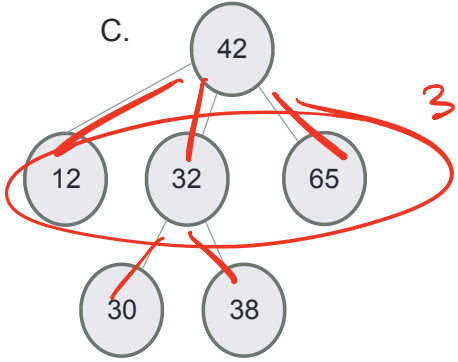
A.



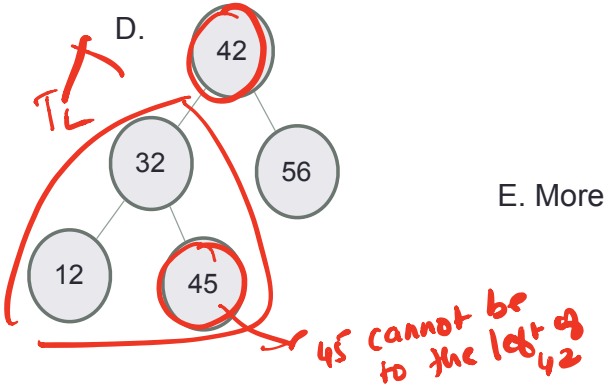
B.



C.

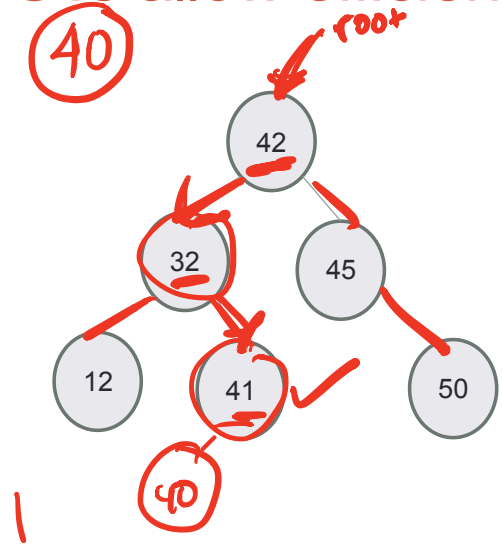


D.



E. More than one of these

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 = 0;
  }
};
```

Define the BST ADT

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

given a key, find the next largest key

given a key, find the next smaller value