M15 On-Your-Own Practice Questions Selected Answers and Examples

Selected answers are below in red. The file attached on this page contains some coding example solutions. Note that these show only *one possible solution*. There are many other possible coding solutions!

FinalExamPracticeFiles.zip (https://ccsf.instructure.com/courses/47904/files/7254430/download?wrap=1) (https://ccsf.instructure.com/courses/47904/files/7254430/download?download_frd=1)

Question 1: Evaluating Recursive Methods

What is the output of the following recursive method calls? I recommend you do **not** simply run them! Instead, trace out what is happening by hand to make sure you understand.

Choice A:

```
System.out.println(recMethod1(5, 1)); // 4

recFactorial1(4); // 4 3 2 1 (and returns 24)
recFactorial2(4); // 2 3 4 (and returns 24)

public int recMethod1(int x, int y) {
   if(x==y) {
      return 0;
   } else {
      return recMethod1(x-1, y) +1;
    }
}

public int recFactorial1(int x) {
   System.out.print(x);
   if(x > 1) {
      return x * recFactorial1(x-1);
   } else {
      return 1;
   }
}

public int recFactorial2(int x) {
   if(x > 1) {
      int fac = x * recFactorial2(x-1);
      System.out.print(x);
      return fac;
   } else {
      return 1;
   }
}
```

Choice B:

```
int[] a = {6, 2, 4, 6, 2, 1, 6, 2, 5};

System.out.println(recMethod2(a, 2, 0));  // 3
System.out.println(recMethod2(a, 2, 9)); // 0

System.out.println(recMethod3(a, 0)); // 0


public int recMethod2(int[] arr, int b, int j) {
   if(j<arr.length) {
      if(arr[j] != b) {
         return recMethod2(arr, b, j+1);
       } else {
         return 1+recMethod2(arr, b, j+1);
       }
   } else {
     return 0;
    }
}
public int recMethod3(int[] arr, int n) {
   int sum = 0;
   if(n<arr.length-1) {
      recMethod3(arr, n+1); // BUG HERE!!!
   } else {
      sum = arr[n];
   }
   return sum;
}
```

Choice C:

```
chainA: 2 -> 3 -> 1 -> 6 -> 4 // 2 3 1 6 7 2 4 3 (prints the values
of all but the last, then prints the +1 of the values as the recursi
on "winds down"
chainB: 3 -> 5 -> 4 -> 2 // 3 5 4 5 6 4
chainC: 4 // prints nothing
chainD: an empty chain // throws an exception and crashes

call recMethod4 with each chain

public void recMethod4(Node<Integer> current) {
   if(current.next!=null) {
       System.out.println(current.data);
       current.data = current.data+1;
       recMethod4(current.next);
       System.out.println(current.data);
   }
}
```

Question 4: Tracing Simple Sorts

Trace out following sorting algorithms with the datasets below.

1. insertion sort: 4, 1, 8, 7, 2, 5, 3
   - first pass: 1, 4, 8, 7, 2, 5, 3 (the 1 shifts to the left of the 4)
   - second pass: 1, 4, 8, 7, 2, 5, 3 (no changes made- the 8 is already in place)
   - third pass: 1, 4, 7, 8, 2, 5, 3 (the 7 shifts to the left of the 8)
   - fourth pass: 1, 2, 4, 7, 8, 5, 3 (the 2 shifts into place)
   - fifth pass: 1, 2, 4, 5, 7, 8, 3 (the 5 shifts into place)
   - sixth pass: 1, 2, 3, 4, 5, 7, 8 (the 3 shifts into place)
2. selection sort: 5, 3, 9, 7, 6, 2, 1
   - first pass: 1, 3, 9, 7, 6, 2, 5 (the minimum value, 1, is swapped into place- swapped with the 5)
   - second pass: 1, 2, 9, 7, 6, 3, 5 (the next minimum value 2 is swapped into place- swapped with the 3)
   - third pass: 1, 2, 3, 7, 6, 9, 5 (the 3 is swapped with the 9)
   - fourth pass: 1, 2, 3, 5, 6, 9, 7 (the 5 is swapped with the 7)
   - fifth pass: 1, 2, 3, 5, 6, 9, 7 (no changes made- the 6 is already in place)
   - sixth pass: 1, 2, 3, 5, 6, 7, 9 (the 9 is swapped with the 9)
3. Shell sort: 5, 6, 9, 10, 12, 2, 3, 8, 7
   - first pass: gap = 9/2 = 4; add one to make it odd, so gap = 5
   - 5 - - - - 2 - - - becomes 2 - - - - 5 - - -
   - - 6 - - - - 3 - - becomes - 3 - - - - 6 - -
   - - - 9 - - - -8 - becomes - - 8 - - - - 9 -
   - - - - 10 - - - - 7 becomes - - - 7 - - - - 10
   - no more swaps, so the array is:
   - 2, 3, 8, 7, 12, 5, 6, 9, 10
   -
   - second pass: gap = 5/2 = 2; add one to make it odd, so gap = 3
   - 2 - - 7 - - 6 - - becomes 2 - - 6 - - 7 - -
   - - 3 - - 12 - - 9 - becomes - 3 - - 9 - - 12 -
   - - - 8 - - 5 - - 10 becomes - - 5 - - 8 - - 10
   - no more swaps, so the array is:

- 2, 3, 5, 6, 9, 8, 7, 12, 10
-
- third pass: gap = 3/2 = 1; which is insertion sort; the final array is:
- 2, 3, 5, 6, 7, 8, 9, 10, 12

Question 6: Tracing Sequential Search

Trace a sequential search of the following sorted list:

```
A D H M X Y
0 1 2 3 4 5
```

1. Search for E
   - 0 1 2; when we reach H, we've passed where the target would have been, so we can stop looking
2. Search for M
   - 0 1 2 3; target found
3. Search for Z
   - 0 1 2 3 4 5; target not found

Trace a sequential search of the following unsorted list:

```
X A H Y D M
0 1 2 3 4 5
```

1. Search for E
   - 0 1 2 3 4 5; target not found
2. Search for D
   - 0 1 2 3 4; target found
3. Search for Z
   - 0 1 2 3 4 5; target not found

Question 7: Tracing Binary Search

Trace a binary search of the following sorted list. List the values of first, last, and mid for each pass through the list.

```
A D H M X Y
0 1 2 3 4 5
```

1. Search for A
   - first = 0, last = 5, mid = 2; A is < than H so last gets updated to mid-1
   - first = 0, last = 1, mid = 0; target found
2. Search for E

- first = 0, last = 5, mid = 2; E is < than H so last gets updated to mid-1
- first = 0, last = 1, mid = 0; E is > than A so first gets updated to mid+1
- first = 1, last = 1, mid = 1; E is > than D so first gets updated to mid+1
- first = 2, last = 1; search is over because first > last; target not found

3. Search for L
   - first = 0, last = 5, mid = 2; L is > than H so first gets updated to mid+1
   - first = 3, last = 5, mid = 4; L is < than X so last gets updated to mid-1
   - first = 3, last = 3, mid = 3; L is < than M so last gets updated to mid-1
   - first = 3, last = 2; search is over because first > last; target not found
4. Search for P
   - first = 0, last = 5, mid = 2; P is > than H so first gets updated to mid+1
   - first = 3, last = 5, mid = 4; P is < than X so last gets updated to mid-1
   - first = 3, last = 3, mid = 3; P is > than M so first gets updated to mid+1
   - first = 4, last = 3; search is over because first > last; target not found
5. Search for Z
   - first = 0, last = 5, mid = 2; Z is > than H so first gets updated to mid+1
   - first = 3, last = 5, mid = 4; Z is > than X so first gets updated to mid+1
   - first = 5, last = 5, mid = 5; Z is > than Y so first gets updated to mid+1
   - first = 6, last = 5; search is over because because first > last; target not found

Question 8: Tracing Stacks and Queues

1. Trace the contents of a queue after each the following statements:

```
        queue.enqueue(2); FRONT 2 BACK
        queue.enqueue(3); FRONT 2 3 BACK
        queue.enqueue(7); FRONT 2 3 7 BACK
        queue.dequeue(); FRONT 3 7 BACK
        queue.enqueue(queue.getFront()); FRONT 3 7 3 BACK
        queue.enqueue(4); FRONT 3 7 3 4 BACK
        queue.enqueue(queue.dequeue()); FRONT 7 3 4 3 BACK
        queue.getFront(); FRONT 7 3 4 3 BACK (front value is returned, but i
        gnored)
        queue.enqueue(1); FRONT 7 3 4 3 1 BACK
        queue.enqueue(5); FRONT 7 3 4 3 1 5 BACK
        queue.enqueue(queue.dequeue()); FRONT 3 4 3 1 5 7 BACK
```

2. Trace the contents of a priority queue after each statement executes.

   pq is a priority queue such that lower numbers have higher priority. Assume that the first character of the object specifies its priority. For example, 1a has priority 1. 2b has higher priority than 3a; 2b has the same priority as 2e.

```
        pq.add(2d); FRONT 2d BACK
```

```
pq.add(1c); FRONT 1c 2d BACK
pq.add(2a); FRONT 1c 2d 2a BACK
pq.add(3b); FRONT 1c 2d 2a 3b BACK
pq.add(pq.getFront()); FRONT 1c 1c 2d 2a 3b BACK
pq.add(1a); FRONT 1c 1c 1a 2d 2a 3b BACK
pq.add(pq.remove()); FRONT 1c 1a 1c 2d 2a 3b BACK
pq.add(3a); FRONT 1c 1a 1c 2d 2a 3b 3a BACK
```

3. Trace the contents of a deque after each of the following statements:

```
deque.addToFront(2); FRONT 2 BACK
deque.addToBack(3); FRONT 2 3 BACK
deque.addToFront(7); FRONT 7 2 3 BACK
deque.removeFront(); FRONT 2 3 BACK
deque.addToFront(deque.getBack()); FRONT 3 2 3 BACK
deque.addToBack(4); FRONT 3 2 3 4 BACK
deque.addToFront(deque.removeBack()); FRONT 4 3 2 3 BACK
deque.addToBack(deque.getFront()); FRONT 4 3 2 3 4 BACK
deque.addToBack(1); FRONT 4 3 2 3 4 1 BACK
deque.addToFront(5); FRONT 5 4 3 2 3 4 1 BACK
```

4. Trace the contents of a stack after each of the following statements is executed:

```
stack.push(1); BOTTOM 1 TOP
stack.push(5); BOTTOM 1 5 TOP
stack.push(3); BOTTOM 1 5 3  TOP
stack.pop(); BOTTOM 1 5 TOP
stack.push(stack.peek()); BOTTOM 1 5 5 TOP
stack.push(7); BOTTOM 1 5 5 7 TOP
stack.pop(); BOTTOM 1 5 5  TOP
stack.push(6); BOTTOM 1 5 5 6 TOP
stack.peek(); BOTTOM 1 5 5 6 TOP (top value is returned, but ignore
d)
stack.push(stack.pop()); BOTTOM 1 5 5 6 TOP
stack.pop(); BOTTOM 1 5 5 TOP
```

5. Trace the contents of a stack and queue after the following statements are executed:

```
queue.enqueue(10); queue: FRONT 10 BACK
queue.enqueue(5); queue: FRONT 10 5 BACK
stack.push(queue.getFront()); queue: FRONT 10 5 BACK; stack: BOTTOM
10 TOP
stack.push(7); queue: FRONT 10 5 BACK; stack: BOTTOM 10 7 TOP
queue.enqueue(stack.pop()); queue: FRONT 10 5 7 BACK; stack: BOTTOM
10 TOP
queue.enqueue(stack.peek()); queue: FRONT 10 5 7 10 BACK; stack: BOT
TOM 10 TOPP
stack.push(queue.dequeue()); queue: FRONT 5 7 10 BACK; stack: BOTTOM
```

```
        10 10 TOP
        stack.push(queue.dequeue()); queue: FRONT 7 10 BACK; stack: BOTTOM 1
        0 10 5 TOP
        queue.enqueue(queue.dequeue()); queue: FRONT 10 7 BACK; stack: BOTTO
        M 10 10 5 TOP
```

Question 9: Balanced Parentheses

What is the left on the stack when the trace is done and how does this tell you whether or not the parentheses are balanced?

1. a * (b + c * (d - e) + f) empty stack; balanced parentheses
2. (a *(b + (c * d)- e + f) (left on the stack; extra left parenthesis
3. a * (b + c * d) - e + f) empty stack; extra closed parenthesis
4. (a * b) + (c * d) - (e + f) empty stack; balanced parentheses

Question 10: Evaluating Post-Fix Expressions

Determine if each of the following is a valid post-fix expression. If it is, evaluate the expression.

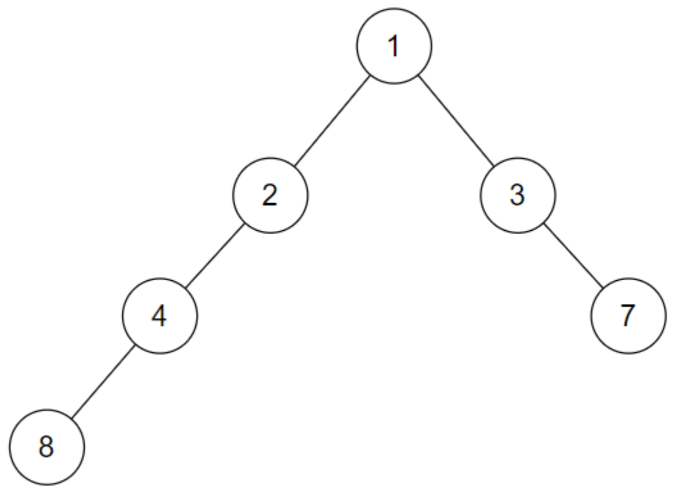
1. 3 4 * 6 2 / + 15
2. 3 5 3 + * 6 / 4
3. 2 4 8 2 / + * 16
4. 1 6 9 + 3 / + * invalid- there is only one value on the value stack when the * is reached (6 * ??? nothing there!)

Question 11: The Heapform Array

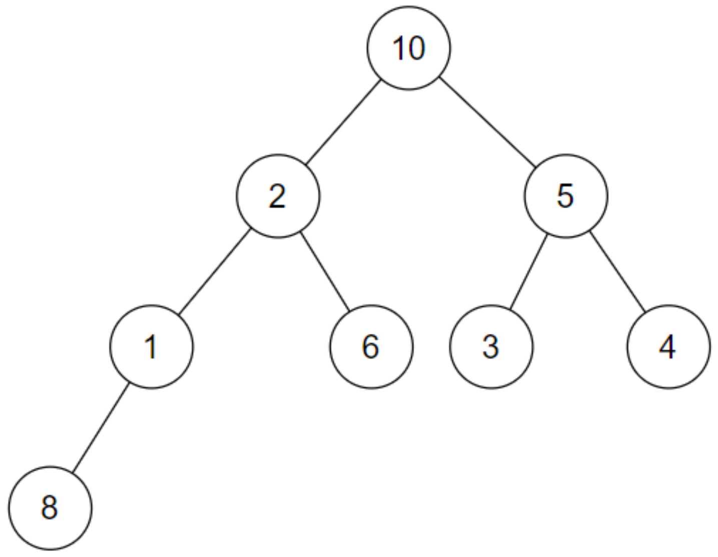
Draw the trees represented by the heapform arrays below. (* represents an empty space in the array.)

Is each of these tree a heap? If so, it is a maxheap or minheap?

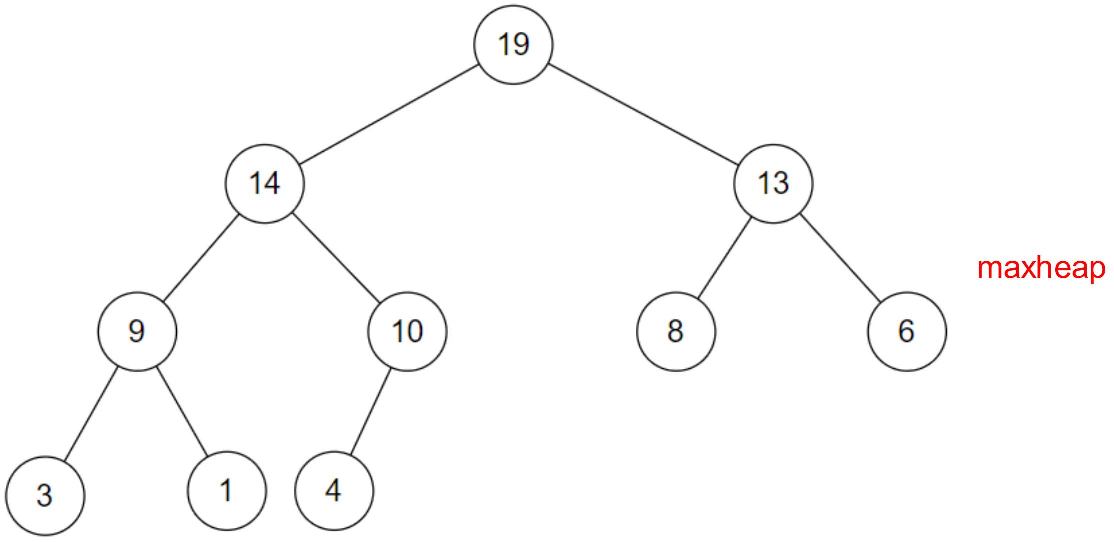
1. [*, 1, 2, 3, 4, *, *, 7, 8]
2. [*, 10, 2, 5, 1, 6, 3, 4, 8]
3. [*, 19, 14, 13, 9, 10, 8, 6, 3, 1, 4]



not a heap because it's not complete



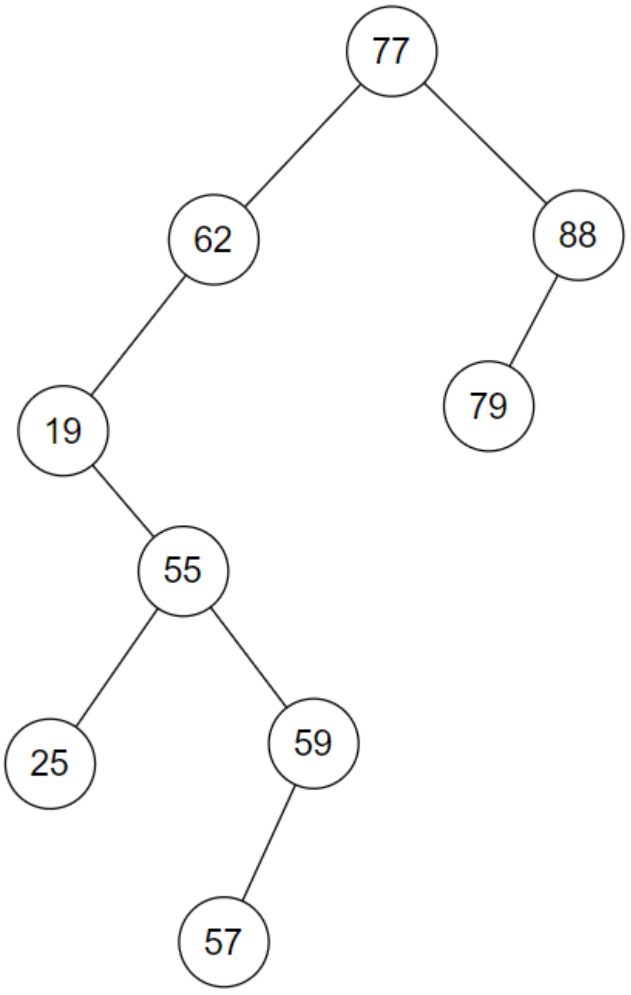
complete, but does not have max or min structure



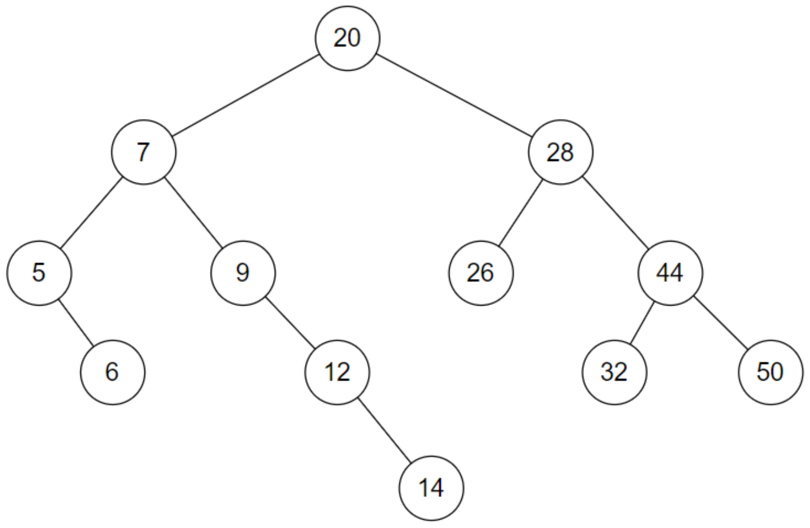
Question 14: Binary Search Tree Creation

Draw the binary search trees that result from inserting the sequences in the specified order. Are your trees unique for this order? What about for these numbers inserted in a different order?

- 77 62 88 19 79 55 59 25 57
- 20 7 9 12 5 14 6 28 44 50 32 26



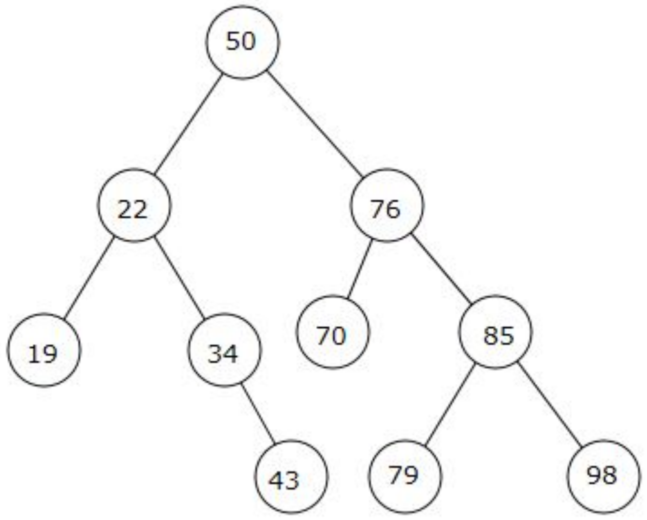
This BST is unique for this insertion order. This is the only BST that can be build if inserting the numbers in this order. If you inserted the numbers in a different order, you would get another BST.



This BST is unique for this insertion order. This is the only BST that can be build if inserting the numbers in this order. If you inserted the numbers in a different order, you would get another BST.

Question 15: Tree Traversals

What is the pre-order, in-order, and post-order of the binary search tree below?



Pre-Order: 50 22 19 34 43 76 70 85 79 98

In-Order: 19 22 34 43 50 70 76 79 85 98

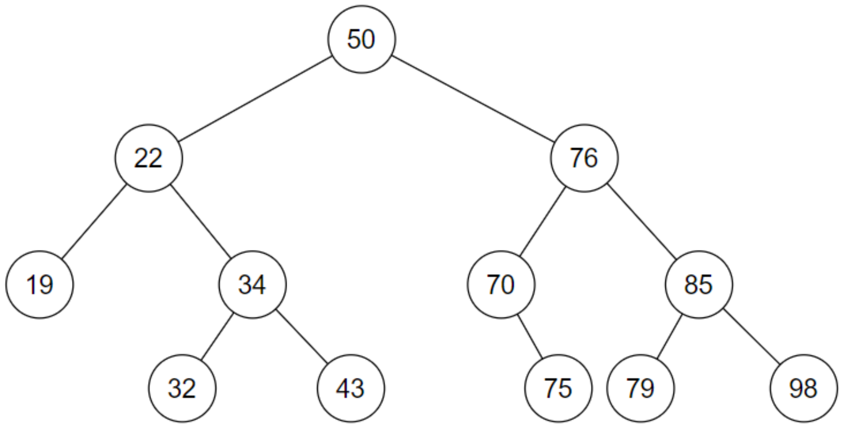
Post-Order: 19 43 34 22 70 79 98 85 76 50

Question 16: Tree Add and Delete

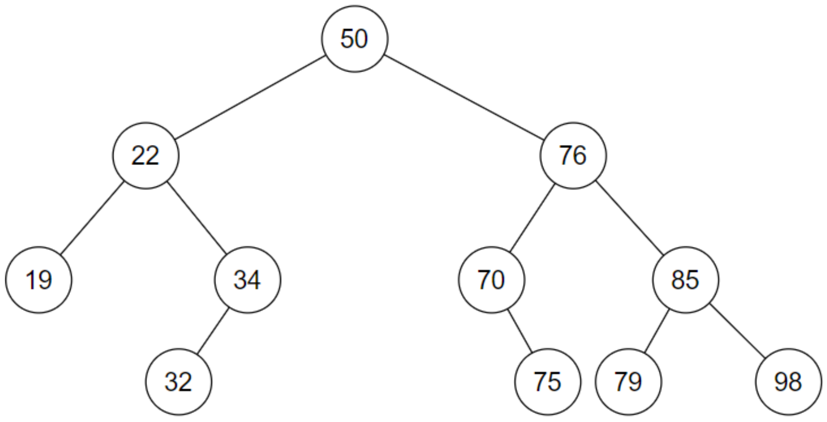
Draw the binary search tree above from the question above after inserting value 75 and then 32.

Now draw the binary search tree after deleting the value 43, then 85, then 50.

After adds:

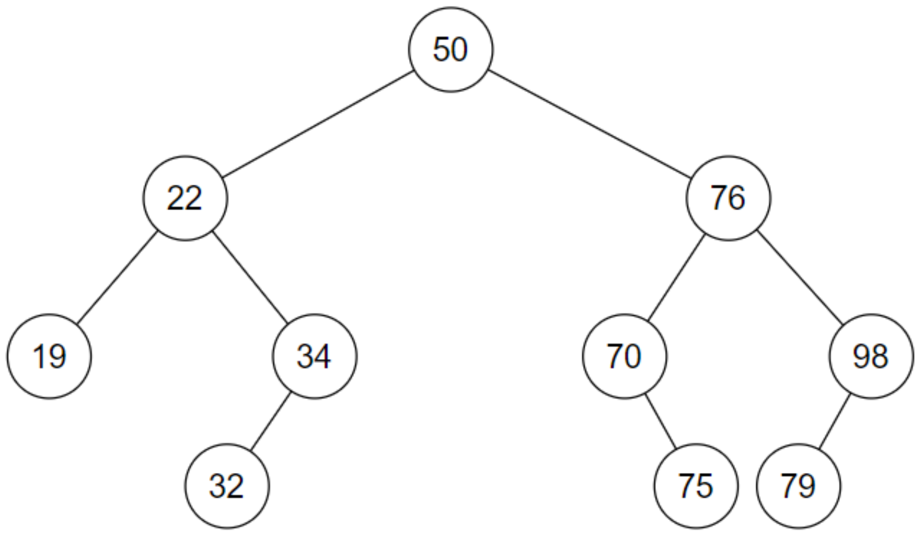


After deleting 43:



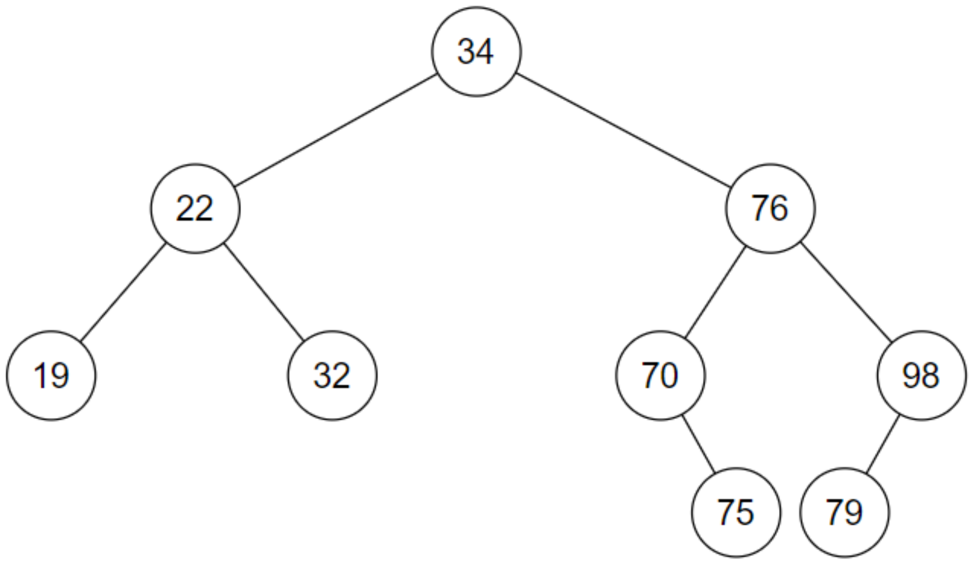
One way of deleting 85- replace with 98, the left-most node in the right subtree, which is also the node before 85 in the inorder traversal.

(You could also have replaced 85 with 79, which is the right-most node in the left subtree, and the node after 85 in the inorder traversal.)



One way of deleting 50- replace with 34, the right-most node in the left subtree, and the node before 50 in the inorder traversal. The 32 also needs to be swapped in to replace the 34.

(You could also have replaced 50 with 70, which is the left-most node in the right subtree, which is also the node after 50 in the inorder traversal. In this case, the 75 would also need to be swapped in to replace the 70.)



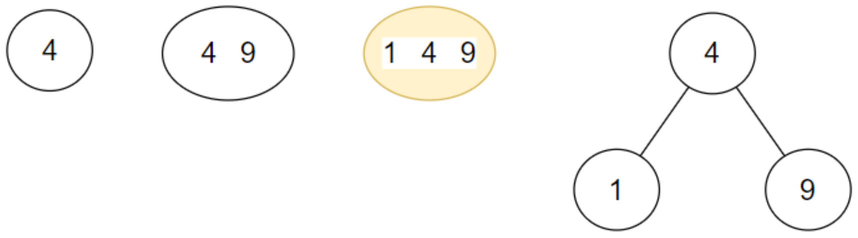
Question 17: B-Trees

Draw the 2-3 tree that results from the following sequence. Show each tree that results from each addition.

4 9 1 3 2 7 8 5 6

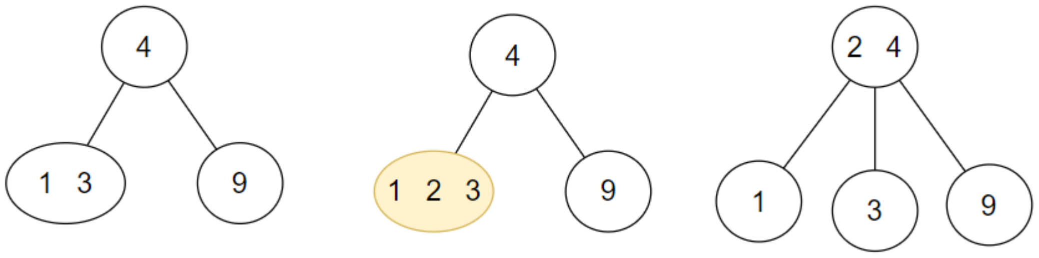
2-3 Tree:

Trees shown in yellow are not actual trees- they are intermediary steps that occur before the tree can be formed.



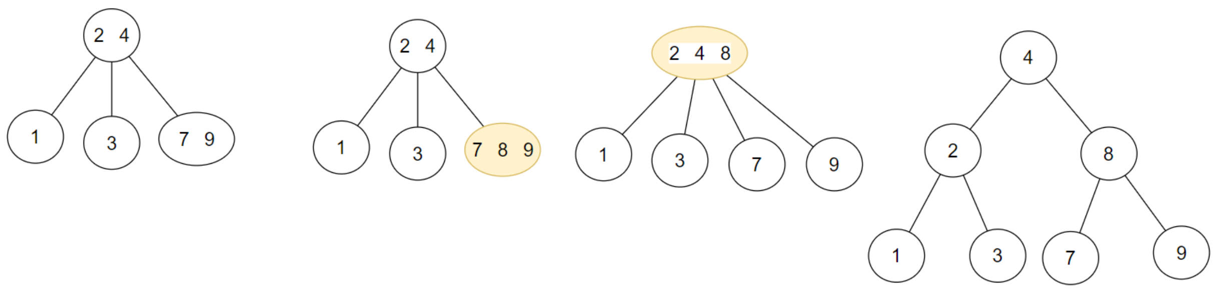
When adding 1, we pretend to create a 4-node (a node with 3 values: the 1-4-9 node). But 2-3 trees don't actually allow these nodes- we just pretend it's there and then split by bumping the middle value up a level. If there is no level above, we make a new node for the root.

Note that after each addition, the tree follows the properties of a binary search tree.



When adding 2, we again pretend to create a 4-node (1-2-3) and then we split and bump up the middle value. Since there is already a node above, that value joins that node (to create the 2-4 node).

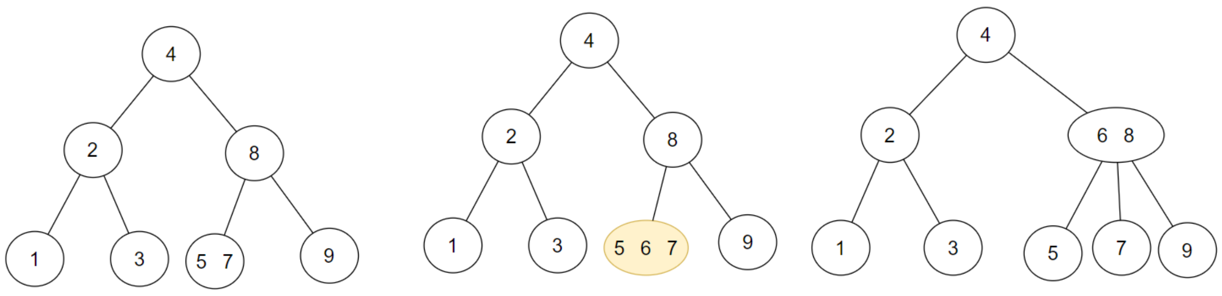
Note again that after each addition, the tree follows the properties of a binary search tree. In this case, the "left child" of the 2-4 node contains values smaller than 2, the "middle child" contains values between 2 and 4, and the "right child" contains values greater than 4.



Note that we never create a new node during an add! We only create new nodes during splits. (And of course the very first add!)

Again note that the binary search tree properties are upheld!

When we add 8, we pretend to create a 4-node (7-8-9). We then split and bump up the 8. This creates another 4-node (the 2-4-8 node), so we split again.



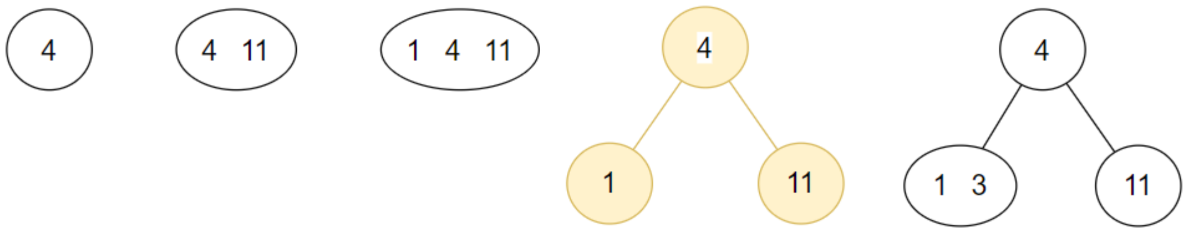
Again note that new nodes are only created during splits, not during adds of values. Adds of values only change 2-nodes (nodes with one value) to 3-nodes or change 3-nodes to 4-nodes (which then must be split). During the split, new nodes can be created.

Draw the 2-4 tree that results from the following sequence. Show each tree that results from each addition.

4 11 1 3 2 7 10 5 6 8

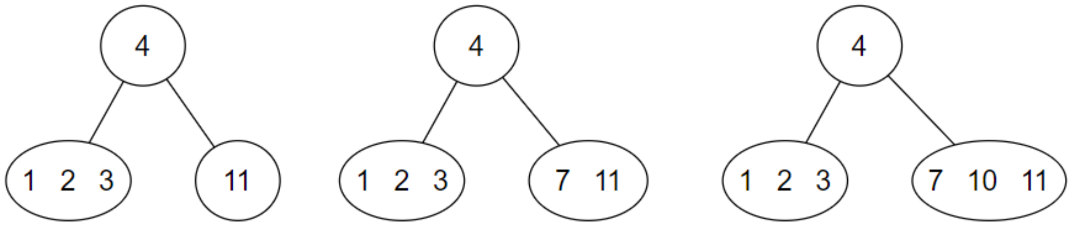
2-4 Tree:

Trees shown in yellow are not actual trees- they are intermediary steps that occur before the tree can be formed.

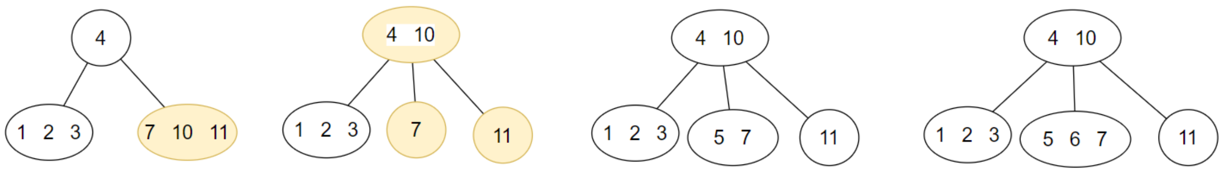


In a 2-4 tree, we **are** allowed to have 4-nodes (nodes with 3 values). We only split when we encounter a 4-node during an add as part of our search. When we add the 3, we encounter the 1-4-11 node. So we pause our search and split that node. Then we continue on to place the 3 where it belongs.

Similar to a 2-3 tree, new nodes are never created during adds (except for the very first add). They are only created from splits.

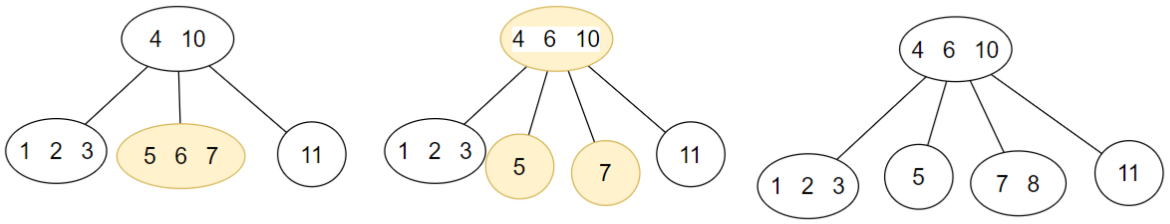


Again, note that 4-nodes are allowed to exist in the tree.



When we go to add the 5, we encounter the 7-10-11 node. So we pause our search and split this node. We then place the 5 where it belongs.

After the 5 is placed, notice again the the tree has the binary search tree properties. The "left child" of the 4-10 node contains values less than 4, the "middle child" contains values between 4 and 10, and the "right child" contains values greater than 10.

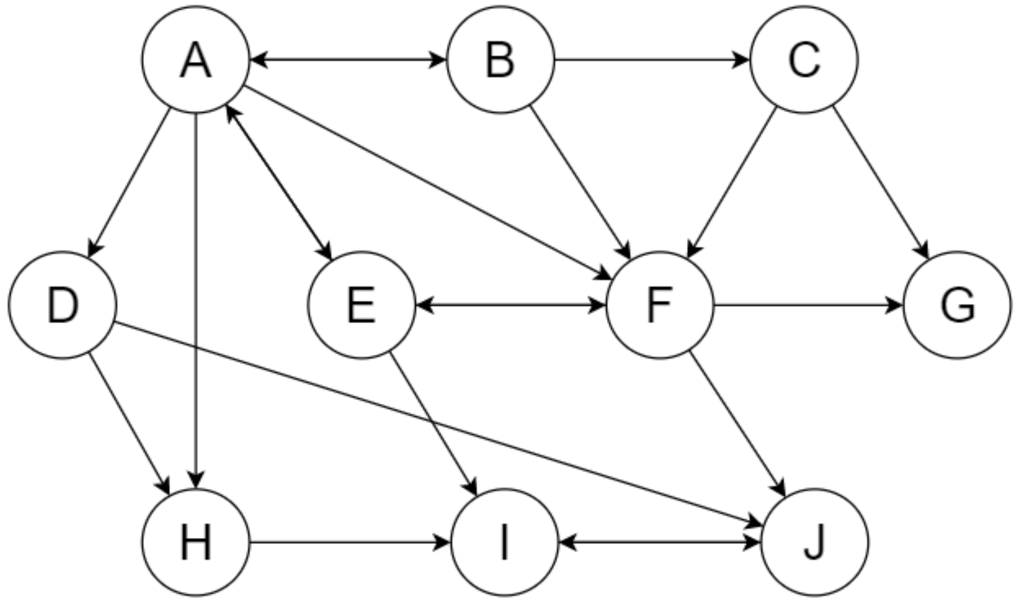


In searching for where to add the 8, we encounter the 5-6-7 node. So we pause our search and split. Note that this split creates another, new 4-node. But we do **not** split this node. We just continue on to search for where to place the 8. If a new 4-node is created by a split (meaning that it is "above" where we are looking), we do not split it- we just continue on.

If we were to have another value being added after the final tree shown, the 4-6-10 node would be encountered as the first node in the search and, at that point, it would be split.

Note again the binary search tree properties of the final tree- the "left most" child contains values smaller than 4, the "second from the left" child contains values between 4 and 6, the "second from the right" child contains values between 6 and 10, and the "right most" child contains values bigger than 10.

Question 19: Graph Traversal and Ordering



What is a depth-first traversal of the graph below, starting at A? starting at D? Is this the only possible correct traversal?

One possible depth first starting at A: A B C F E I J G D H

This is not the only possible correct traversal, there are many other correct traversals. (For example, one other correct traversal is: A H I J E F G D B C. There are many others.)

One possible depth first starting at D: D H I J.

This is not the only possible correct traversal. Another possible correct traversal is D J H I.

What is a breadth-first traversal of the graph below, starting at A? starting at D? Is this the only possible correct traversal?

One possible breadth first starting at A: A B D E F H C J I G

This is not the only possible correct traversal, there are many other correct traversals. A correct traversal would have all of the blue-shaded vertices come first, but arranged in any order. These all come first because these vertices are A's neighbor. The traversal would then have all of the yellow-shaded vertices last, in any order. These vertices all come last because they are all one step away from A (meaning you travel two edges to reach them from A).

One possible breadth first starting at D: D H J I.

This is not the only possible correct traversal. The other possible correct traversal is D J H I.

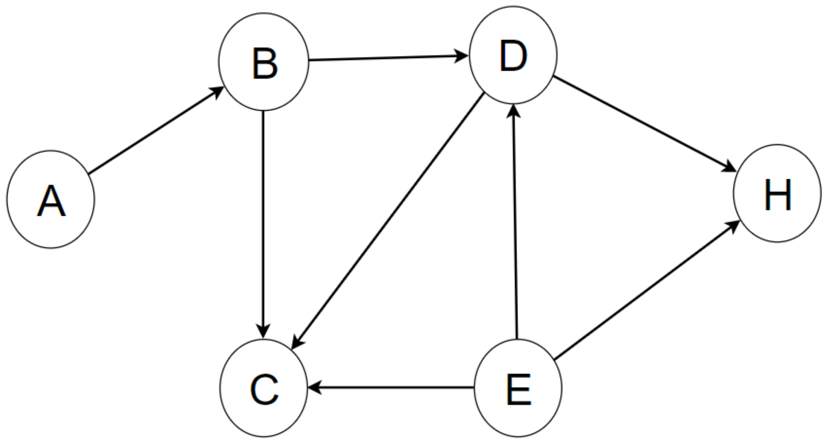
What is a shortest path from A to G? from E to J? Is this the only possible correct path?

A shortest path from A to G is A F G. In this case, this is the only correct shortest path.

A shortest path from E to J is E F J. This is not the only possible correct shortest path. Another correct example is E I J.

Can you create a topological ordering for this graph? If yes, what is a valid ordering? Is it the only possible correct ordering? If not, why not?

You cannot create a topological ordering because this graph is cyclical, meaning it has at least one cycle. An example of a cycle is A B F E A.



What is a depth-first traversal of the graph below, starting at A? starting at E? Is this the only possible correct traversal?

One possible depth-first starting at A: A B C D H. There are other possible traversals (example: A B D H C).

One possible depth-first starting at E: E C D H. There are other possible traversals (example: E D C H).

What is a breadth-first traversal of the graph below, starting at A? starting at E? Is this the only possible correct traversal?

One possible breadth-first starting at A: A B C D H . There are other possible traversals (example: A B D C H).

One possible breadth-first starting at E: E C D H. There are other possible traversals (example: E H D C).

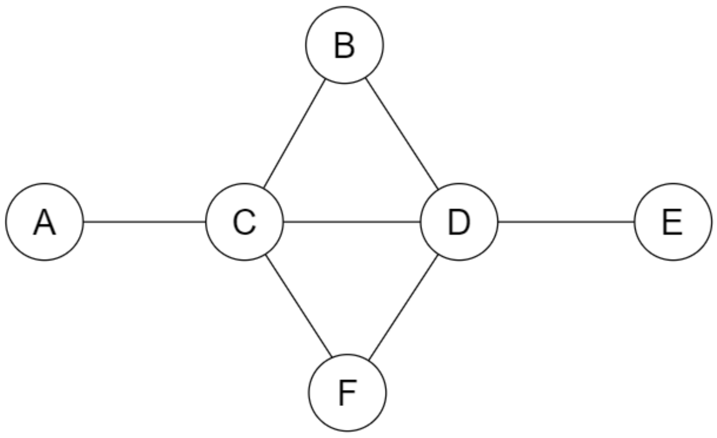
Can you create a topological ordering for this graph? If yes, what is a valid ordering? Is it the only possible correct ordering? If not, why not?

Yes, you can create a topological ordering because the graph is acyclical (it has no cycles). One possible topological ordering for the graph: A B E D C H

There are other possible correct orderings (example: A B E D H C).

Question 20: Graph Representations

What is the adjacency matrix of the graph below? What is the adjacency list? Is the graph weighed? directed? connected? complete?



	A	B	C	D	E	F
A			T			
B			T	T		
C	T	T		T		T
D		T	T		T	T
E				T		
F			T	T		

A -> C

B -> C D

C -> A B D F

D -> B C E F

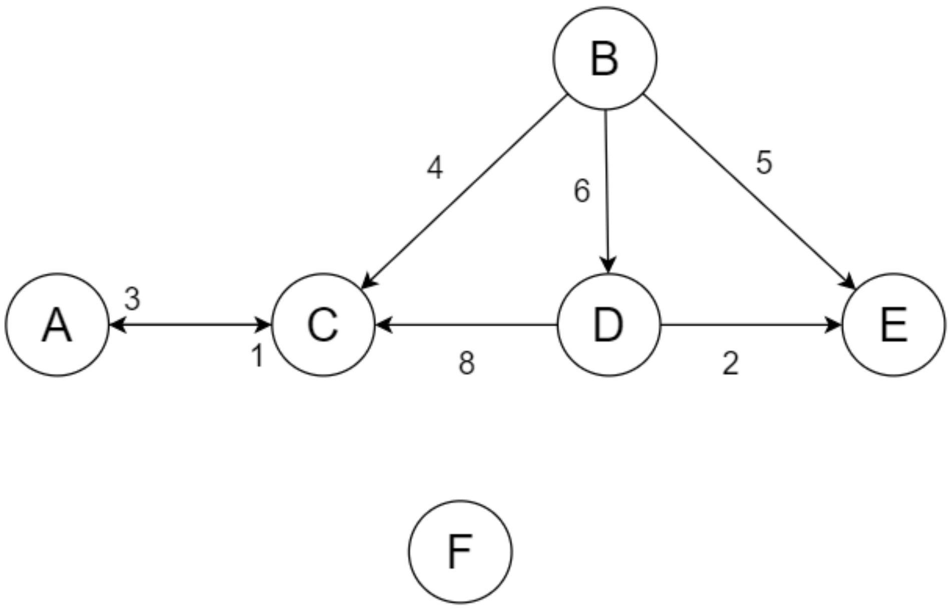
E -> D

F -> C D

This graph is undirected and unweighted. The graph is connected but not complete.

Draw a graph with the adjacency matrix provided below. Is the graph weighed? directed? connected? complete?

	A	B	C	D	E	F
A			1			
B		4		6	5	
C	3					
D			8		2	
E						
F						



This graph is directed and weighted. The graph is not connected and not complete.