

Read each question carefully. When writing code, note that not all questions require a complete class, or even a complete method. Keep in mind that this review is not intended to be comprehensive. While studying for the exam, it would be prudent to consider variations on the questions presented here.

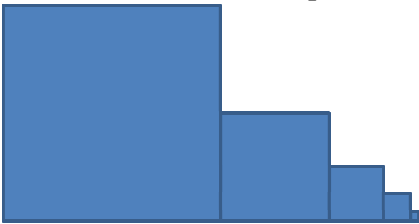
Assume you want to implement a priority queue using an array of arrays. Using Big-O expressions, explain the advantages & disadvantages of this approach regarding both speed & storage requirements.

The primary advantage is that if the number of different priorities are known in advance, the array of arrays approach will give $O(\text{constant})$ time inserts (provided a given priority buffer is not full) and removals. If the destination buffer is full, insert becomes $O(x)$ where x is the size of the buffer (as we'll need to copy the contents to a new, larger buffer).

The primary disadvantages to this approach are:

- 1) If the number of different priorities are not known in advance, performance will be at worst $O(d*x)$ where d is the number different priorities and x is the buffer size for each priority.
- 2) the space required is $O(d*x)$, but it might not all be in use, especially if there are few elements & very many different priorities.

Assume you have access to a method `Draw.square(x,y,s)` which draws a square of size s pixels at position (x,y) on the screen. Write a recursive method that takes a starting position and size & draws the pattern below to the screen (don't worry about things like drawing off the edge of the monitor).



```
public void drawPattern(int x, int y, in size)
{
    // we can't draw anything smaller than a pixel
    if(size > 0)
    {
        Draw.sqaure(x,y,s);
        // need to move left by s pixels & cut the size in half
        drawPattern(x+s, y, s/2);
    }
}
```

In big O notation indicate the performance of the following operations on the indicated data structures:

SinglyLinkedListQueue with the front of the queue is at the tail

// add element e to the queue

insert(e) $O(1)$

// remove element from the queue

remove() $O(n)$ Because we have to scan through the entire sequence to find the new tail

CircularArrayQueue

// add e element to the queue (assuming there is not enough room)

insert(e) $O(n)$ Because we need to copy the entire array into a larger one

// remove element from the queue

remove() $O(1)$

ArrayHeap

// add e element to the heap

insert(e) $O(\log n)$

// add element to the heap (assuming there is not enough room)

insert(e) $O(n)$

LinkedHeap

// add e element to the heap

insert(e) $O(\log n)$

// remove element from the heap

remove() $O(\log n)$

Note: I should have been more specific about the nature of the binary search trees. The intention was that they were for storing/searching numerical values.

LinkedBinarySearchTree

// determine if element e is in the tree

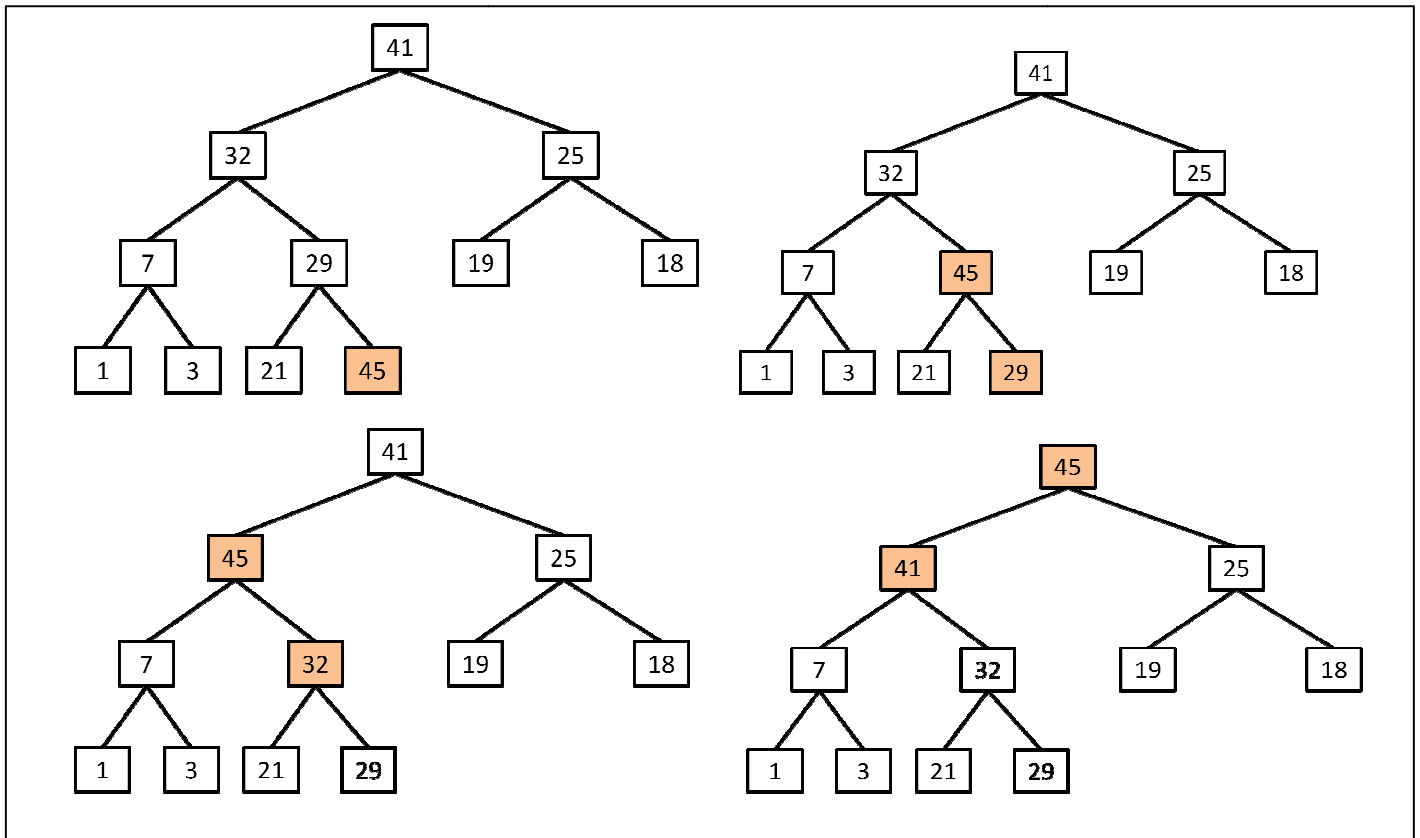
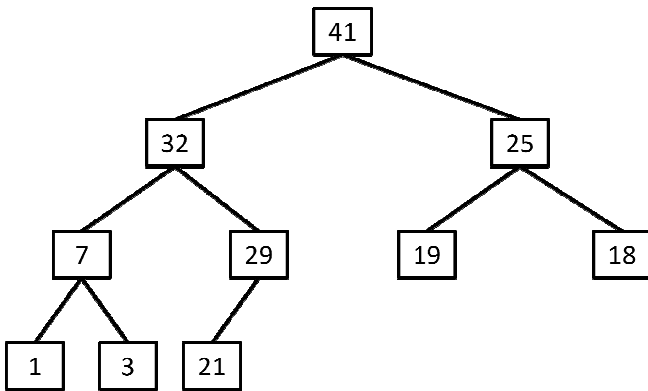
find(e) $O(d)$ where d is the depth of the tree. If the tree is balanced, this becomes $O(\log n)$. In a worst case scenario (completely skewed tree) it is $O(n)$

List the invariants of a binary search tree.

Each node has at most 2 nodes.

For a given node e: $e.leftChild.data < e.data < e.rightChild.data$

Starting with heap shown below, diagram the steps required to add the element 45.



Define the depth of a node as it applies to trees.

Depth of a tree is the max depth of all its nodes; the depth of a node is defined as the number of links it is from the root of the tree. The root node by definition has a depth of zero.

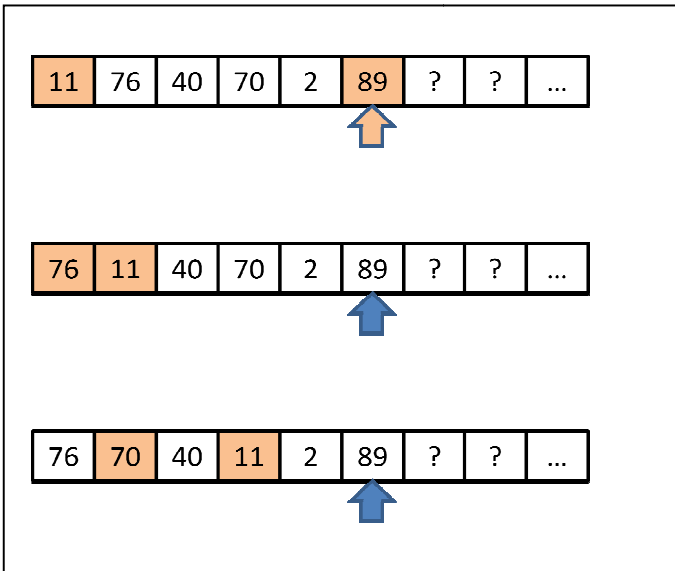
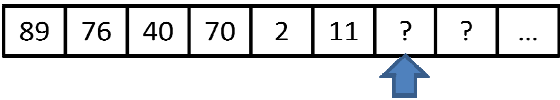
Define a leaf node.

A leaf node is any node in the tree that that does not have children.

Define what it means for a tree to be full.

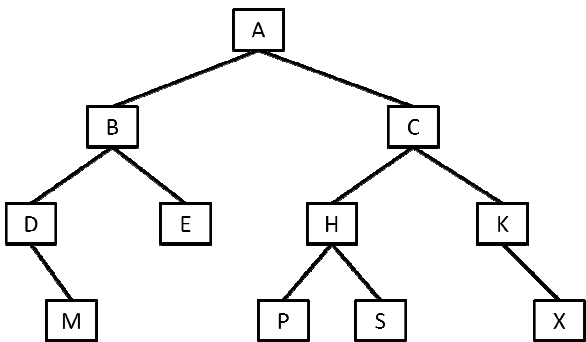
A tree is said to be full if every leaf has the same depth, and every interior node has the maximum number of allowed children. For instance, in a binary tree, all parent nodes must have two children for the tree to be full.

Starting with the heap shown below as an array, diagram the steps required to remove the largest element. Note: the arrow indicates the index marked by manyItems.



Note: in this answer, I've opted to swap the removed value to the manyItems spot – this isn't strictly required, I could have left it with the original value. The reason is that that position is not actually part of the heap, so the value there is not of great consequence.

Write the preorder traversal for the tree shown below:



The preorder traversal is:
 A B D M E C H P S K X

Entries in a binary search tree which allows duplicates are "ordered". What is the meaning of this statement?

- A. Parent > LeftChild, Parent > RightChild
- B. Parent < LeftChild, Parent < RightChild
- C. Parent > LeftChild, Parent < RightChild
- D. Inorder traversal yields an ordered list
- E. both D & B
- F. both D & A

Note: this question had an incorrect answer. Technically C was not correct as since the question did not strictly forbid duplicate elements in the tree. D is correct whether duplicates are allowed or not.

Which of the following is true of heaps?

- A. they are FIFO data structures
- B. they are LIFO data structures
- C. they can be used for priority queues
- D. they can sort a list in linear time
- E. they are always full
- F. none of the above

Which of the following is true about tail recursion?

- A. it is generally faster than looping
- B. it is generally slower than looping
- C. it can always be replaced with looping
- D. both A & C
- E. both B & C
- F. none of the above

Which of the following is true about B-Trees?

- A. they are binary trees
- B. all leaf nodes are at the same depth
- C. for a given node, data[j] >= all elements in subtree[j]
- D. for a given node, the # of subtrees == the number of elements
- E. the tree is symmetric
- F. None of the above

Which of the following is true of queues?

- A. they are FIFO data structures
- B. they are LIFO data structures
- C. they are non linear data structures
- D. both A & C
- E. both B & C
- F. none of the above