

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.

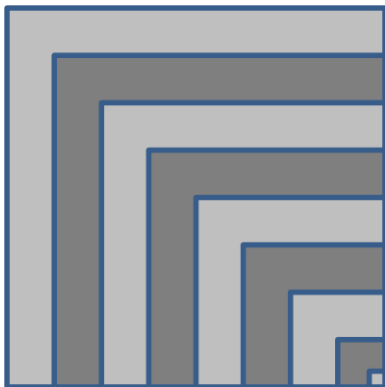
Assume you want to implement a priority queue using a linked list of queues. Using Big-O expressions, explain the advantages & disadvantages of this approach regarding both speed & storage requirements.

Let d be the total different number of priorities & n be the total number of elements in the queues. Let c refer to some constant. The advantage is that this approach is that its memory use is $O(n)$.

Removing the next item in the queue can be done in $O(c)$ provided the queues are stored in ascending order in the linked list (since the highest priority queue would always be at the head).

The main disadvantage is that inserting an item on average takes time $O(d)$ since we have to go through the linked list looking for the appropriate queue to insert it into.

Assume you have access to a method `Draw.square(x,y,s,b)` which draws a square of size s pixels at position (x,y) on the screen w/ a given brightness (1.0 is white, 0.0 is black). Write a recursive method that takes a starting position and size & draws the pattern below to the screen. Assume each squares below are reduced by 20 pixels at each step. Don't worry about getting the colors exactly correct.



```
public void drawIt(x,y,s,b)
{
    // can't draw something <= to zero!
    if(s > 0)
    {
        drawSquare(x,y,s,b);
        // swap colors for the next square
        if(b <= 0.4)
            b = 0.6;
        else
            b = 0.4;
        // next square needs to move right & down by 20,
        // shrink by 20 & use the new color
        drawIt(x+20,y-20,s-20,b);
    }
}
```

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) $O(\text{constant})$

// remove element from the queue

remove() $O(n)$: we have to find the new tail

SinglyLinkedListQueue with the front of the queue is at the head

// add element e to the queue

insert(e) $O(\text{constant})$

// remove element from the queue

remove() $O(\text{constant})$

CircularArrayQueue

// add e element to the queue (assuming there is enough room)

insert(e) $O(\text{constant})$

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

insert(e) $O(n)$: must copy all data to a new array

LinkedHeap

// add e element to the heap

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

// remove root element from the heap

remove() $O(\log n)$

LinkedBinarySearchTree

// determine if element e is in the tree

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

// remove all occurrences of element e is in the tree

removeAll(e) $O(\text{heightOfTree})$: it might not be $O(\log n)$ because the tree might not be balanced

// remove all elements descending from & including the given node k

prune(k) $O(\text{constant})$: set the parent of k so that it has null as a child instead of k

List the invariants of a heap.

(1) The value of any node is \geq that of its children

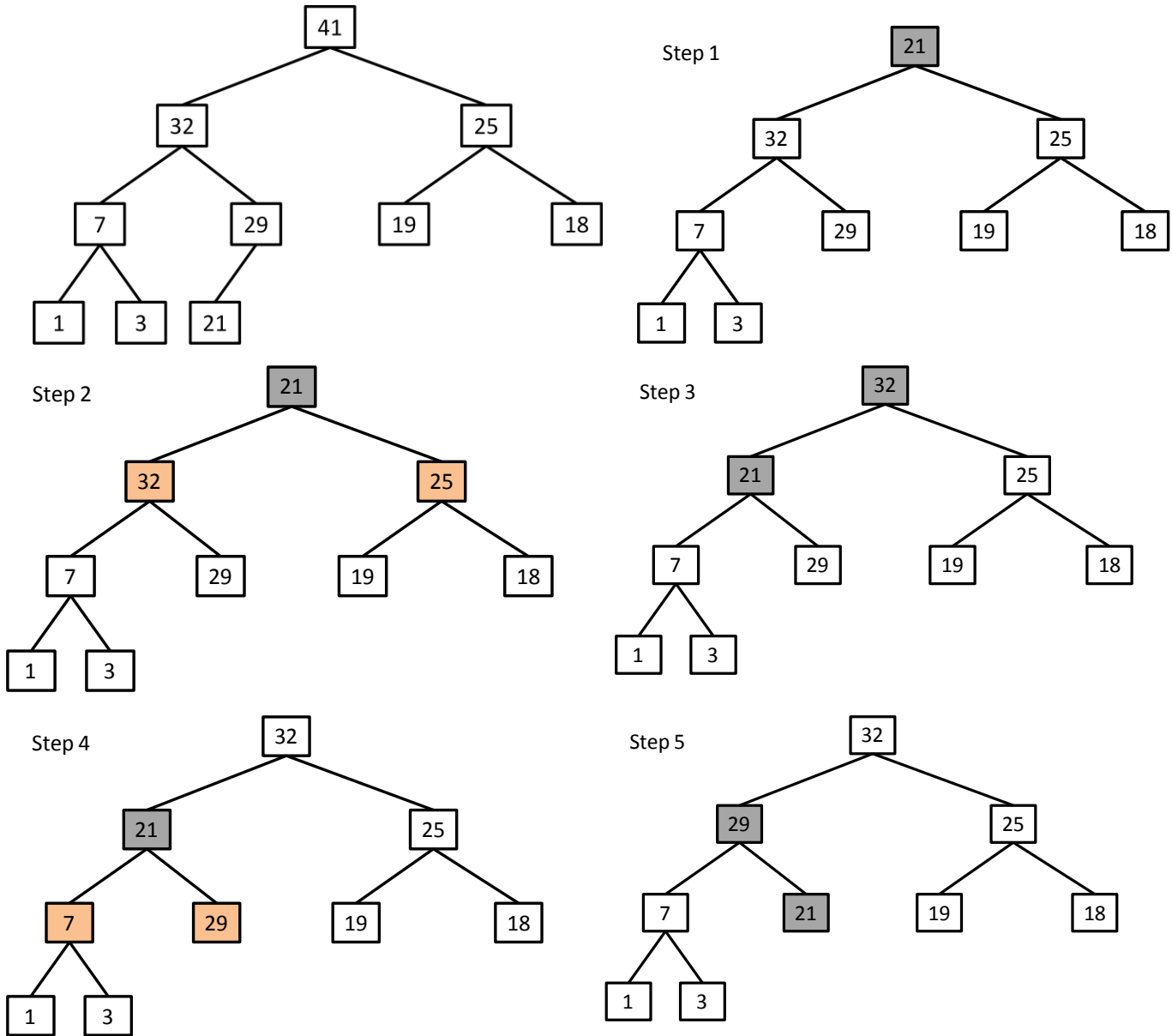
(2) The tree is always complete meaning

(a) every level except the deepest must be as full as possible and

(b) the lowest level fills from left to right

Note: heaps are always binary trees

Starting with heap shown below, diagram the steps required to remove the root element.



As formally as possible, define the sibling relationship as it pertains to nodes in a tree.

Two nodes, say x and y , are siblings, if and only if the parent of x is also the parent of y .

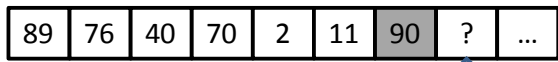
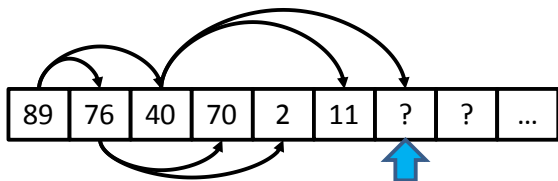
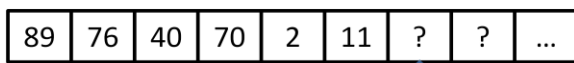
Define an inner node.

A node, say x , is an inner node if and only if x is the parent of some other node, say y .

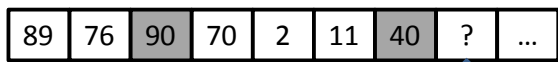
Define what it means for a binary tree to be complete.

- (1) Every level except the deepest must be as full as possible
- (2) The lowest level is filled from left to right

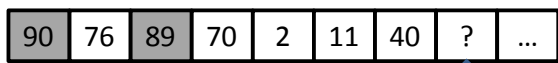
Starting with the heap shown below as an array, diagram the steps required to add the element 90. Note: the arrow indicates the index marked by manyItems.



Add to open spot, move manyItems



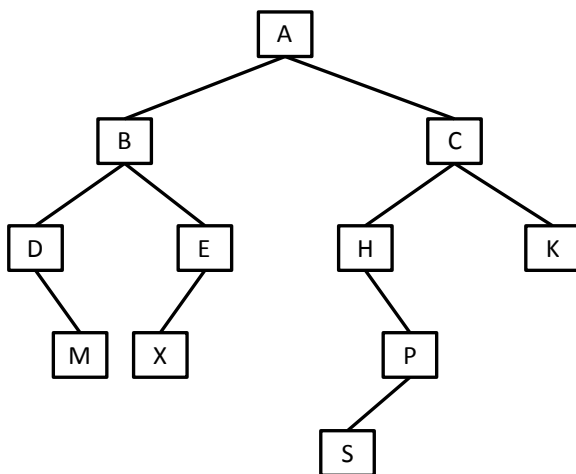
Trade w/ parent



Trade w/ parent again & we're done



Write the pre, post & in-order traversals for the tree shown below:



Pre: A B D M E X C H P S K

In: D M B X E A H S P C K

Post: M D X E B S P H K C A

Entries in a binary search tree 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

Which of the following is true of queues?

- A. they are FIFO data structures
- B. they are LIFO data structures
- C. they can be used for heaps queues
- D. they can sort a list in linear time
- E. they support random access
- 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 BSP-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 heaps?

- 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