Week 4 Quiz

TOTAL POINTS 10

1. Assume you are storing a complete binary tree as a contiguous list of keys in an array such that the root's key is stored at location 1 of the array, the keys from all of the nodes at the next level of the tree are stored in left-to-right order in subsequent locations in the array, then similarly for all of the nodes of each subsequent level. 1 point

   At what array location would the key stored in the node that is the left child of the right child of the root?

   Enter answer here

2. When using an array to store a complete tree, why is the root node stored at index 1 instead of at the front of the array at index 0? 1 point

   - ○ We use index zero as a guard to prevent overstepping the root when propagating up the tree from its leaf nodes, which would cause a memory access fault.
   - ○ Array index 0 is used to store the number of nodes in the complete tree stored in the array.
   - ○ We avoid using index 0 to avoid confusion with the value of 0 (**nullptr**) that we normally store in the child pointer of a node to indicate that child does not exist.
   - ○ This makes the math for finding children and parents simpler to compute and to explain.

3. When is a binary tree a min-heap? 1 point

   - ○ When the leaf nodes represent the smallest values in the tree, and every leaf node is smaller than the root.
   - ○ When every node's value is greater than its parent's value.
   - ○ When every node's value lies between the maximum value of its left child's subtree and the minimum value of its right child's subtree.
   - ○ When every node's value is less than its parent's value.

4. How should one insert a new value into a heap to most efficiently maintain a balanced tree? 1 point

   - ○ Maintain the heap as a balanced binary search tree. Walk down the tree from the root exploring the left children first, then the right children, until a node is found that is greater than the new value. Insert a new node with the new value at that position and make the previous node the left child of that new node. Rebalance the tree if the height balance factor magnitude of the new node or its parent exceeds one.
   - ○ Maintain the heap as an AVL tree. Walk down the tree from the root exploring the left children first, then the right children, until a node is found that is greater than the new value. Insert a new node with the new value at that position and make the previous node the left child of that new node. Then call the appropriate rotation routine to rebalance the tree if the height balance factor magnitude of the new node or its parent reaches two.
   - ○ Maintain the heap as an array. Walk down the tree from the root at position 1 in the array, exploring the left children first, then the right children, until a node position is found whose value is greater than the new value. Copy the value at that position and all subsequent positions in the array to one greater position in the array, and store the new value at that position.
   - ○ Maintain the heap as a complete tree and insert a new value at the one new node position that keeps the tree as a complete tree. Then continually exchange the new value with the value of its parent until the new value is in node where it is greater than the value of its parent.

5. The removeMin operation removes the root of a min-heap tree. Which of the following implements removeMin efficiently while maintaining a balanced min-heap tree? 1 point

   - ○ Replace the root value with the value of the last leaf (rightmost node at the bottom level) of a complete binary tree, and delete the last leaf. Then repeatedly exchange this last-leaf value with the smaller of the values of its node's children until this last-leaf value is smaller than the values of its node's children, if any.

- Increment the address used to indicate the base location of the array storing the complete binary tree.
- Set the root value to +infinity. If the left child is smaller than the right child, perform a Right-Rotation, otherwise perform a Left-Rotation. Repeat this process at the new infinity-node location until the infinity node is a leaf, then remove and delete it.
- Delete the root and if the root has two children, then merge its left subtree with its right subtree by inserting each right subtree node value into the left subtree. Then delete the right subtree.

6. How many nodes of a complete binary tree are leaf nodes? 1 point

- Unknown. Could be one. Could be all.
- About a fourth.
- About half.
- About the square root.

7. Recall that the heapifyDown procedure takes a node index whose children (if any) are heaps, but the value of the node might not satisfy the heap property compared to its children's values. This procedure then swaps the node's value with the smallest child value larger than it (if any), and then calls itself on that smallest child node it just swapped values with to further propagate that value down the heap until it finds a valid location for it. 1 point

```
template <class T>
void Heap<T>::_heapifyDown(int index) {
    if (!_isLeaf(index)) {
        T minChildIndex = _minChild(index);
        if (item_[index] > item_[minChildIndex] ) {
            std::swap( imem_[index], item_[minChildI]);
            _heapifyDown(minChildIndex);
        }
    }
}
```

When you call heapifyDown on a given node, what is the maximum number of times heapifyDown is called (including that first call) to find a valid location for the initial value of that node?

- The maximum number of times heapifyDown is called is the number of non-leaf nodes in its subtree.
- The maximum number of times heapifyDown is called is one plus the height of the node.
- The maximum number of times heapifyDown is called is the number of nodes in its subtree.
- heapifyDown is only called once since its children are already heaps.

8. What is the run-time algorithmic complexity of calling heapifyDown on every non-leaf node in a complete tree of n nodes? 1 point

- O(1)
- O(n lg n)
- O(n)
- O(n^2)

9. Which of the following is the fastest way to build a heap of n items? 1 point

- Create a heap with a single node holding the first item. Then insert the remaining n-1 items into the heap.
- Create a complete tree of the items in any order. If this is not a heap, then swap the items between two randomly chosen nodes and check again. Repeat the random swapping until you get a heap.
- Create a complete tree of the items in any order, then call heapifyUp on every node in the tree from the root down

- ◯ Create a complete tree of the items in any order, then call heapifyUp on every node in the tree from the root down to the leaf nodes.
- ◯ Create a complete tree of the items in any order, then call heapifyDown on every non-leaf node from the bottom of the tree up to the root.

10. Which of the following is NOT a step of the heap sort algorithm?

1 point

- ◯ Insert the next item into the current heap.
- ◯ Remove the root node.
- ◯ Run heapifyDown on every non-leaf node.
- ◯ Load the data in any order into a complete tree.

Save Submit