### Chapter 16 Tree Implementations - Answers

**1. Why is it natural to implement a binary tree using a link-based structure?**

**Answer:** The edges in the binary tree can naturally be represented using links (pointers). Nodes can be dynamically allocatedand de-allocated as necessary.

**2. Why is it somewhat difficult to implement a binary tree using an array-based structure?**

**Answer**: It is difficult because you must represent a non-linear structure (tree) in a linear structure (array).You use an array of nodes where the links are represented using indexes. You must write code to manage the portion of the array that contains nodes that are not currently part of the tree (a free list of nodes). In other words, you have to provide memory management code that simulates what dynamic memory allocation does for you in C++.

**3. The textbook gives you the code for a general inorder traversal of a binary tree on page 486 of the textbook. Write a corresponding method called preorder to perform a preorder traversal.**

**Answer:**

template<class ItemType>

void BinaryNodeTree<ItemType>::preorder(void visit(ItemType&),

std::shared\_ptr<BinaryNode<ItemType>> treePtr) const

{

if (treePtr != nullptr)

{

ItemType theItem = treePtr->getItem();

visit(theItem);

preorder(visit, treePtr->getLeftChildPtr());

preorder(visit, treePtr->getRightChildPtr());

} // end if

} // end preorder

**4. Starting with an empty binary search tree, in what order would you have to add data items to get the binary search tree in Figure 15-18 on page 471 of your text?**

**Answer:** There are several possible orders that would result in this same tree.Two possibilities are:

60, 20, 10, 40, 30, 50, 70

60, 20, 70, 10, 40, 30, 50

As long as each value that you add is stored in a child of a node you have already added, you will get the same tree.

### Chapter 15 Trees - Answers

**1. What is a tree? What is a parent? What is a child? What is a root?**

**Answer:** A tree is a non-linear, hierarchical data structure that consists of nodes and edges. Each node contains data and links to zero or more successors which we call the children of the node. Every node (except one) has exactly one predecessor which we call the parent of the node. Every tree that contains nodes has exactly one node that has no predecessor which we call the root of the tree. A tree can be empty (contains no nodes).

**2. What is a subtree?**

**Answer:** A subtree is a tree node and all its descendants (and the edges that conect them).

**3. What is a leaf?**

**Answer:** A node that has no children.

**4. What is a binary tree?**

**Answer:** See page 448.

**5. What is the height of a tree as defined in the textbook?**

**Answer:** See page 450.

**6. What is a full binary tree? What is a complete binary tree? What is a balanced binary tree?**

**Answer:** See pages 451 - 452.

**7. What is a binary search tree?**

**Answer:** See page 463.

**8. What is the worst-case efficiency of a retrieval operation on a binary search tree?**

**Answer:** The worst case is when each node in the tree has at most one child. In this case the tree will resemble a linked listand the worst case efficeincy is O(n).

**9. What is the worst-case efficiency of a retrieval operation on a balanced binary search tree?**

**Answer:** In a balanced binary search tree, the retrieval operation is O(log n).

**10. What is the worst-case efficiency of a traversal operation on a binary tree?**

**Answer:** A traversal operation requires that each node in the tree be visited. The efficiency is O(n).

### Chapter 13 Queues and Priority Queues - Answers

**1. Are entries in the ADT queue ordered by position or by value?**

**Answer:**The value of an entry does not matter. The entries are kept in a sequence (ordered by position). The position is determined by the order that entries are inserted into the queue. The entry that has been in the queue the longest will be at the front of the sequence. The entry added most recently will be at the end (back) of the sequence.

**2. When inserting an entry in a queue, can the client specify the position of the new entry?**

**Answer:**No. The enqueue operation always places the new entry at the back of the queue.

**3. Can the client retrieve a copy of any entry from the queue?**

**Answer:**No. The peekFront() operation can only retrieve a copy of the entry at the front of the queue.

**4. Can the client remove any entry from the queue?**

**Answer:** No. The dequeue() operation removes the entry at the front of the queue.

**5. Are entries in the ADT priority queue ordered by position or by value?**

**Answer:** The position of a new entry is determined by the ADT based on a priority value which is provided by the client. So entries are "ordered" by value.

**6. When inserting an entry in a priority queue, can the client specify the position of the new entry?**

**Answer:** The client cannot specify a position relative to the existing entries. However, the client does specify a priority value which is used by the adT to determine the position of the new entry..

**7. Can the client retrieve a copy of any entry from the priority queue?**

**Answer:** No. The peekFront() operation returns a copy of the entry with the highest priority value.

**8. Can the client remove any entry from the priority queue?**

**Answer:** No. The dequeue() operation removes the entry with the highest priority value.

**9. Which of the following ADTs would be most appropriate for each of the following situations? Choose from: stack, queue, list, sorted list, priority queue.**

1. **an employer who fires the most recently hired person**  
   **Answer:** stack (last-in first-out)
2. **a list of names in alphabetic order**  
   **Answer:**sorted list
3. **airplanes that stack above a busy airport waiting to land**  
   **Answer:** queue if planes get to land in the order in which they arrive (first in first out), or maybe a priority queue if planes get to land in order based on their priority
4. **people who are put on hold when calling customer service**  
   **Answer:** queue
5. **customers who take a number at a deli counter**  
   **Answer:** queue
6. **a grocery list**  
   **Answer:** list
7. **integers that need to be sorted**  
   **Answer:** sorted list

### Chapter 12 Sorted Lists and Their Implementations - Answers

**1. The ADT List allows the client to specify the position in the list for a new entry. How is the position of a new entry decided in the ADT Sorted List?**

**Answer**: The client just specifies the entry to be inserted. The ADT insertSorted method decides where to place the entry based on its value. That is, the value of the entry determines its position in the sorted list.

**2.Assuming that wordListPtr is a pointer to a sorted list as described in the textbook, write C++ code to display the last entry in the list.**

**Answer:**  
 int numberOfWords = wordListPtr->getLength();

std::cout << wordListPtr->getEntry(numberOfWords) << std::endl;

**3. Given that nameListPtr points to an empty sorted list that is initially empty, and the following sequence of names is added:**

**nameListPtr->insertrSorted("Romeo"); nameListPtr->insertrSorted("Juliet"); nameListPtr->insertrSorted("Sierra");**

**What is returned by the call:**

**nameListPtr->getPosition("Romeo");**

**Answer:** 2

**4. In the ADT Sorted List, what is the efficiency (time complexity) of the getEntry(position) operation in the linked-based implementation (section 12.2 of the text)?**

**Answer:** The efficiency is O(n).

**5. In the ADT Sorted List, what is the efficiency (time complexity) of the getEntry(position) operation in the implementation based on the ADT List (array-based version - see section 12.3)?**

**Answer:** The efficiency is O(1) for this array-based implementation.

### Chapter 14 Queue and Priority Queue Implementations - Answers

**1. If you implement the ADT queue using an array with the queue entries stored at indexes 0 through number\_of\_entries - 1, what disadvantage do you have?**

**Answer:** Say you add the first entry at index 0, the second at index 1, etc. That means that index 0 holds the front entry and index number\_of\_entries - 1 holds the back entry. Adding a new entry (enqueue operation) is O(1). The problem comes when you need to remove the front entry (dequeue operation). This would require that all remaining entries be moved one position to the left (to keep the front entry at index 0). This makes the dequeue operation O(n).

**How can you fix this?**

**Answer:** Don't move the entries around in the array. Let the part of the array that holds the entries "move around". You can keep track of the front and back of the queue by storing the index of the front entry and the index of the back entry. But there is still one more problem to solve.

As you add and remove entries, the part of the array that is occupied by entries will "drift to the right". Eventually you will add an entry at the last index in the array. But there may very well be unused locations at the beginning of the array where entries have been removed. To solve this problem, you use what is called a circular array. After you add an entry at the last index of the array, you need to wrap around to add the next entry at the beginning of the array. You need to do the same thing when you remove entries.

To add an entry (enqueue), you need to increment the back index to get the location of the new back entry:

   back = (back + 1) % DEFAULT\_CAPACITY;

To remove an entry (dequeue), you need to increment the front index to get the index of the new front entry:

   front = (front + 1) % DEFAULT\_CAPACITY;

**2. In the array-based implementation of a queue, how do you tell if the queue is full or empty?**

**Answer:** An easy solution is to store a count of the number of entries. The queue is empty if count == 0. The queue is full if count == DEFAULT\_CAPACITY.

**3. You can use a linked list with a single pointer to the head node to store the entries for a queue. What disadvantage would you have with this implementation? How can you solve this problem?**

**Answer:** For a queue you need to be able to add entries at the back and remove entries from the front. Assume that you store the front entry in the head node and you store the most recent entry in the last (tail) node. Removing the front entry (dequeue operation) is O(1). Adding a new entry (enqueue operation) is O(n). It would be nice if both operations were O(1).

**How can you solve this problem?**

**Answer:**Add a second pointer to the queue - a pointer to the tail node. This allows an enqueue operation that is O(1).

**4. Later in the textbook we will cover a technique that can be used to implement a priority queue very efficiently.Meanwhile, what is an easy way to implement a priority queue?**

**Answer:** A priority queue can be implemented by keeping the entries in order sorted by the priority. Since this will be very similar to a sorted list, you can create a simple priority queue by using a sorted list.

**Chapter 11 - Sorting Algorithms and their Efficiency - Questions**

The Radix Sort will not be covered on the exam. For the rest of the sorts covered in the chapter, I want you to have a basic idea of how each sort works. You should be able to describe each sort in a few sentences.

1. What is the worst-case time complexity of the Selection Sort?

2. What is the worst-case time complexity of the Bubble Sort?

3. What is the worst-case time complexity of the Insertion Sort?

4. What is the worst-case time complexity of the Merge Sort?

5. Consider the Quick Sort algorithm using the first element as the pivot. What is the worst-case time complexity?

6. Consider the Quick Sort algorithm using median-of-three pivot selection. What is the average-case time complexity?.

7. If you use median-of-three pivot selection with Quick Sort, you can only sort arrays of more than 3 entries.You must use another method to sort once Quick Sort breaks the array into pieces that have size smaller than 4. Additionally, Quick Sort is inefficient when sorting small arrays of 10 or 15 entries. What sort does the textbook suggest you use once the array size gets "small enough"?