

Q13. You can use an array to store a binary tree in exactly the way that this is done for heaps. (Root at position 0, left child of root and 1, right child of root at 2, etc). Generally, the only time this is actually used is for heaps. Why?

Q14. You are given code that implements all but the remove function for an UNORDERED binary tree. Write a remove function for this style of tree. You can assume the other functions in TreeInterface have all been implemented correctly and are available for your use.

Q17. Write the function "int countFullNodes()" which returns the number of nodes in the binary tree that have 2 children.

q21. Binary search trees are order dependent in the sense that the tree is dependent on precise order in which items are added to and deleted from the tree. Give an example of order dependence of BSTs. Are AVL BSTs also order dependent? Support your yes or no answer with examples

Q8: Suppose I wanted to replace the standard string sorting algorithm (as implemented by compareTo) with an algorithm where the value of a letter is based on the frequency of letter use. So, for instance, e is the most common so it sorts first, then t, then a, ...

Here is the complete order of letter frequency (for English)

e,t,a,o,i,n,s,h,r,d,l,c,u,m,w,f,g,y,p,b,v,x,j,x,q,z

Write a class with the following definition:

```
public class StringFrequencyOrder extends String
```

Note that String implements the Comparable interface, so all you actually have to do is to override the compareTo method. You may use any data structure to store the letter frequency information. If you cannot write such a class, at least write a compareTo function (that is consistent with the Comparable interface). You may assume that this class (or your compareTo method) only deals with lower case letters.

Q8. The usual procedure for building a heap is $O(n * \log n)$. There is a faster procedure that is $O(n)$. Is it worth it when you are using heapsort? Explain.

Q5. You can use a pair of queues (call them A and B) to implement a Stack as follows:

Stack push: just do an offer onto queue A.

Stack pop: poll every item off of queue A onto queue B except the last item.

With the last put it in a temporary variable tempV

swap queueA and queueB

return tempV

```
public StackQ {
    Queue queueA;
    Queue queueB;
    public StackQ() {
        queueA = new Queue();
        queueB = new Queue();
    }
}
```

```
}
```

Given this start of an implementation of QStack, write offer and poll methods. In the code you write you may NOT have any more uses of “new Queue()”

What is the big-O running time of each of your methods.

Q4. Suppose you have a hashtable using Quadratic probing. What is the worst case running time to insert a key/value into such a hashtable. You should assume that the capacity of the hashtable is C and that it currently contains n items. Explain your answer.

Problem 1:

Consider the following definition of a node for a tree in which each node can have an arbitrary number of children.

```
public class Q1Node<R> {
    final R payload;
    // Pointer to the head of a linked list of child nodes
    // This link goes down the tree
    // So if the current node is at depth N, then
    // firstChild is at depth N+1
    Q1Node<R> firstChild;
    // Pointer (in a single linked list style) to a sibling node
    // If the current node is at depth N then nextChild is also
    // at depth N
    Q1Node<R> nextChild;
    public Q1Node(R pload) {
        payload=pload;
        firstChild=null;
    }
    public void addChild(Q1Node n) {
        n.nextChild = firstChild;
        firstChild = n;
    }
}
```

Write a method that returns the total number of nodes in such a tree .

Question 5:

```
public class LinkedBinaryTree<E extends Comparable<E>> {
    protected class Node
    {
        E payload;
        Node right;
        Node left;
        public Node(E e)
        {
            payload=e;
            right=null;
            left=null;
        }
    }
}
```

```
    }  
    public String toString()  
    {  
        return payload.toString();  
    }  
    /** The root of the tree */  
    protected Node root;  
  
    // Other stuff as needed  
}
```

Given the tree and Node definition above, write a method returns the depth of the lowest node that has 2 children.