Problem 1 TRUE OR FALSE QUESTIONS (4 Points each)
Brief justification is required for credit.

(a) \(100n^2 = O(n^3)\). Hint: recall the definition of the big-Oh notation which involves the existence of two constants, \(C > 0\) and \(n_0\).

**SOLUTION** TRUE: We must show that there is a \(C > 0\) and \(n_0\) such that
\[100n^2 \leq Cn^3\]
for \(n \geq n_0\). But 100\(n_2\) \(\leq n_3\) for all \(n \geq 100\). So we can choose \(C = 1\) and \(n_0 = 100\). Other natural choices include \(C = 100, n_0 = 1\) or \(C = 10, n_0 = 10\).

**REMARK:** to get full credit, you must explicitly indicate the two numerical constants \(n_0\) and \(C\). Any "big-Oh" statement says that there exists two implicit constants \(C > 0\) and \(n_0\).

**Comments:**

(b) \(e^n = O(n^8)\). Hint: recall that \(e^n = \sum_{k \geq 0} \frac{n^k}{k!}\).

**SOLUTION** FALSE: from the series for \(e^n\), we conclude that \(e^n > \frac{n^9}{9!} > Cn^8\) for all \(n > 9!C\).
OK, this argument is slick, and you might miss many details involved. So, let us do this more slowly: First, observe that \(e^n > n^9/9!\). WHY? Because we have omitted positive terms from the infinite series of \(e^n\), and retained the term corresponding to \(k = 9\).
Now are are ready to get a contradiction: suppose for some choice of \(C\) and \(n_0\), we have that, whenever \(n > n_0\), then \(e^n \leq Cn^8\). This means that \(n^9/9! < Cn^8\). This means \(n^9 < (9!C)n^8\). This means \(n < 9!C\). BUT THIS IS NONSENSE: I can choose any \(n\) greater than \(n_0\) and greater than \(9!C\) to get a contradiction.

**REMARK:** Of course, 8 is not particularly special here. The same argument shows that \(e^n \neq O(n^k)\) for ANY \(k\).

Many students just use generalities like “\(e^n\) is growing exponentially fast, and so grows much faster than any polynomial”. That is not precise enough – again, you must get down to the constants like \(C\) and \(n_0\). Remember that \(C, n_0\) is to Computer Science what delta-epsilon is to Calculus.

**Comments:**

(c) The rotation operation \(\text{rotate}(u)\) applied to a binary tree node \(u\) is an \(O(1)\) operation.

**SOLUTION** TRUE: you only need a constant number (at most 6) of pointer statements.

**REMARK:** some students think that \(6 = O(1)\) is false. Of course, this is a grave error. Others think that \(\text{rotate}(u)\) may involve \(\Omega(n)\) work in a tree with \(n\) nodes – but that is not true.
(d) $n^{1/\lg n} = 2$. Hint: take logs on both side.

**SOLUTION** TRUE: let $x = n^{1/\lg n}$. Taking logs to base 2, $\lg x = \lg n (1/\lg n) = 1$. So $x = 2$.

**Comments:**

Problem 2 SHORT QUESTIONS

(a) (2 Points) Name a superclass of the class `Integer` in Java.

**SOLUTION** Object class.

**Comments:**

(b) (3 Points) What is wrong with the statement:

```java
MyList<int> il = new MyList<int>( );
```

**SOLUTION** You cannot instantiate the type of a generic class with a primitive type like `int`. You must use a class type such as `Integer` instead.

**REMARK:** To get full credit, it is NOT enough to just say that we cannot use `int` or that you must use `Integer` instead. You must indicate the general principle involved, namely, class types (or Object types) must be used instead of primitive types.

**Comments:**

(c) (5 Points) Name a concrete data structure in which sentinels is a useful concept. What are the pros and cons of using sentinels in your example?

**SOLUTION** Sentinels are useful in the **doubly linked list data structure**: here, we have a special head and tail nodes to act as sentinels. **Pros:** sentinels allow you to have uniform code that allows you to avoid many special cases. **Cons:** you waste a bit of extra space. Every doubly linked list, has at least two nodes, even if empty.

**Comments:** Some non-reasons students give:

“Pro – sentinels allow you to insert at both two ends of the list.”

“Con – you have to manage dummy nodes the the ends of the list.”

**OTHER COMMENTS:** A linked list class will have a reference to the first node in the list. With sentinels, you never need to change this reference (which always point to the head sentinel). Some students say nothing about the ”cons”, so lose 1 point.

(d) (5 Points) If $u$ is the node of a BST, let $h(u)$ denote the height of the subtree rooted at $u$. Write the recursive formula for $h(u)$. Hint: use $u.left$ and $u.right$. 
SOLUTION Let $u_L$ and $u_R$ be the left and right children of $u$. Then
\[
h(u) = \begin{cases} 
-1 & \text{if } u = \text{null} \\
1 + \max\{h(u_L), h(u_R)\} & \text{else}.
\end{cases}
\]

Comments: PITFALLS: Some students provide incomplete analysis of the base cases, giving, for instance:
\[
h(u) = \begin{cases} 
0 & \text{if } u_L = \text{null} \text{ and } u_R = \text{null} \\
1 + \max\{h(u_L), h(u_R)\} & \text{else}.
\end{cases}
\]
One problem with this solution is that it is implicitly assumed that $u \neq \text{null}$. A second problem is that it misses the case where exactly one of $u_L, u_R$ is null. You can fix this by adding additional cases, of course, but it is clearly inferior to our solution. Hope you see the extreme usefulness of treating the special case of $u = \text{null}$ (just as it is useful in counting numbers to have 0).

NOTE that many students actually write the Java code to compute the height. That is not necessary, but you do not lose any points for doing it.

(e) (6 Points) The postorder of a BST $T$ produces the following sequence of keys: $(1, 3, 5, 4, 8, 7, 9, 6, 2)$. Please reconstruct $T$ from this information.

![Figure 1: Solution for Postorder tree](image)

SOLUTION See Figure 1.

Comments: As some of you noted, you can construct this tree by inserting into an empty tree starting from the end of the list: i.e., insert 2, then 6, then 9, then 7, ... and finally insert 1.

(f) (8 Points) Consider the following function:

```java
static int strange(int n){
    for (int i=0; i< n; i++){
        int j = 1<<i; // NOTE: This assigns j the value $2^i$ (exponentiation)
        while (j>0) {
            j--;
        }
    }
}
```

Give an asymptotic bound for the running time of `strange(n)`.
SOLUTION \( T(n) = \sum_{i=0}^{n-1} (1 + 2^i) \leq 2 \sum_{i=0}^{n-1} 2^i = 2(2^n - 1) \). In short, \( T(n) = O(2^n) \). This sum is the simplest example of a geometric series.

COMMENTS: In fact, \( T(n) = \Theta(2^n) \). Some students say \( T(n) = O(n2^n) \), since each term in the summation is \( O(2^n) \). This is correct of course, but not tight. I award 3 out of 4 points in this case.

Comments:

(g) (6 Points) I created an AVL tree to store the 26 letters of the alphabet. What is the maximum possible height? Hint: let \( \mu(h) \) be the minimum number of nodes in an AVL tree of height \( h \) — what is a recursive formula for \( \mu(h) \)?

SOLUTION Answer is 5. Using the hint, we see that \( \mu(0) = 1, \mu(1) = 2 \) and for \( h \geq 1, \mu(h+1) = 1 + \mu(h) + \mu(h-1) \). Thus we see \( \mu(2, 3, 4, 5, 6) = (4, 7, 12, 20, 33) \) (hopefully this notation is clear). Since \( \mu(5) \leq 26 < \mu(6) \), a tree with 26 nodes have height at most 5.

Comments:

Problem 3 AVL TREES.
Consider the AVL tree \( T_0 \) in Figure 2. All keys in this tree must be integers, and each part of this question is independent of the others.

![Figure 2: Insertion to AVL tree](image)

(a) (5 Points) Please show the tree after we insert key 59 into \( T_0 \).
(b) (5 Points) Please show the tree after we insert key 46 into \( T_0 \).
(c) (3 Points) What is the largest key \( k \) in the range 20 and 60 that you can insert into \( T_0 \) which does not cause any rotation?
(d) (3 Points) What is the smallest key \( k \) in the range 20 and 60 that you can insert into \( T_0 \) so that you need to do a double-rotation?

SOLUTION (a) and (b): See Figure 3.
(c) Largest key to insert without rotation is 54.
(d) Smallest key whose insertion causes double-rotation is 24.

Comments:

Problem 4 JAVA PROGRAMMING. (20 Points)
To get full credit, you must write Java code that will not generate any compile error. In our BST<Integer> class (recall hw3), we have a method called public String search(Integer
which searches for a key \( k \), and returns the String data associated with the key. In this question, we want you to add another method called \( \text{public Integer searchIndex(int i)} \) which returns the \( i \)-th smallest key. E.g., if \( i = 1 \), you return the smallest key in the BST. If \( i = n \) where \( n \) is the size of the BST, you return the largest key.

Assume that we maintain the size of the subtree rooted at each node \( u \). This value is stored in the field \( u.size \). Write the Java code to implement \( \text{searchIndex(int)} \). HINT: it may be useful to write another helper function.

**SOLUTION**

We assume a helper method (also) called \( \text{searchIndex} \) but which (in addition to the index) has a node as an argument:

```java
public Integer searchIndex(int i){
    return searchIndex(i, root);
}

private Integer searchIndex(int i, BinaryNode<Integer> u){
    if (i < 1 || i > u.size)
        throw new RuntimeException("Index out of bounds");
    if (i <= u.left.size)
        return searchIndex(i, u.left);
    if (i == u.left.size+1)
        return u.key;
    return searchIndex(i-1-u.left.size, u.right);
}
```

**Comments:** Most students did an inorder listing of the keys, store into an array, and return the \( i \)-th item in the array. This is an extremely inefficient solution (you need to copy the entire tree into a new array. It also does not use the \( u.size \) information we provided. You get no more than 10 points for this solution.

If you do not check for index out of range, I take off one point. Note the simple way to throw exception in Java above. Please learn this syntax.

**Problem 5** INDUCTION (10 Points)

Ulysses in our class derived this interesting relation connecting \( \phi \) (golden ratio) and \( F_n \) (the
Fibonacci numbers, assuming that $F_n = n$ for $n = 0, 1$:

$$\phi^n = \phi F_n + F_{n-1}, \quad (n \geq 1)$$

Please prove this by induction (note that we ignore the case $n = 0$).

| SOLUTION | Check that the relation holds for $n = 1, 2$: $\phi^1 = \phi(F_1) + F_0 = \phi(1) + 0 = \phi$, and $\phi^2 = \phi(F_2) + F(1) = \phi(1) + 1 = \phi + 1 = \phi^2$. For $n > 2$,

|   | $\phi^{n+1} = \phi(\phi F_n + F_{n-1})$ (by induction hypothesis)  
|   | = $F_n(\phi^2) + \phi F_{n-1}$ (rearrangement)  
|   | = $F_n(1 + \phi) + \phi F_{n-1}$ ($\phi^2 = 1 + \phi$)  
|   | = $\phi F_n + F_{n-1} + F_n$ (rearrangement)  
|   | = $\phi F_{n+1} + F_n$ ($F_{n+1} = F_n + F_{n-1}$) |

Comments: PITFALLS: some students gave only one base case, for $n = 1$. Any proof that does not exploit the two relations

$$\phi^2 = 1 + \phi, \quad F_{n+1} = F_n + F_{n-1}$$

cannot be correct! After all, you must use the properties of $\phi$ and $F_n$ somewhere.