1. Consider the following heap represented as an array indexed by 1 through 9.

![Heap Diagram]

For each of the following questions, use the original heap. Do NOT use the heap obtained after each answer.

(a) Delete the element 99 from the heap shown in the above figure.
(b) Delete the element 23 from the heap shown in the above figure.
(c) Delete the element 8 from the heap shown in the above figure.

---

(a) Element 99 is deleted and replaced by the last element, that is 11. The heap is then restored by moving the largest child up at each level, if necessary.

(b) Element 23 is deleted and replaced by the last element, that is 11. There is no need to restore the heap the deleted node is a leaf node.
(c) Element 8 is deleted and replaced by the last element, that is 11. The heap is then restored by moving the largest child up at each level, if necessary.

2. Answer the questions about the following binary search tree (ordered tree):

(a) Print the tree when the nodes are traversed in preorder.
(b) Print the tree when the nodes are traversed in postorder.
(c) Print the tree when the nodes are traversed in inorder.
(d) Which key is the immediate predecessor of key $b$?
(e) Which key is the immediate successor of key $j$?

(a) In the preorder traversal, we visit the root, and then recursively visit the left subtree, followed by the right subtree. Therefore, the order of visits is:

$$j, (i, (c, b, a), d), (h, g, f)$$

(b) In the postorder traversal, we recursively visit the left subtree, followed by the right subtree, and then the root. Therefore, the order of visits is:

$$((b, a, c), d, i), (g, f, h), j$$

(c) In the inorder traversal, we recursively visit the left subtree, then the root, followed by the right subtree. Therefore, the order of visits is:

$$((b, c, a), i, d), j, (g, h, f)$$

(d) None. $b$ is the smallest key in the tree.
(e) To identify the immediate successor of $j$, we have to pick the leftmost key in the right subtree, which is $g$ in the above case. This is also clear from the inorder traversal.

3. Analyze the following program fragment using “big-oh” analysis in terms of $n$. Show your work.

```plaintext
for i := 1 to n do
    Sum := Sum + A[i, j] + i;
for j := i to n*5 do
    Sum := Sum + A[i, j] + i;
```

The inner for loop has $O(1)$ operations inside, which are executed $5n - i + 1$ times. The value of $i$ ranges between 1 and $n$, and therefore, the inner loop is executed between $4n + 1$ and $5n$ times. Thus, the total number of the operations for the inner loop is $O(n)$ in all cases.

The outer loop is executed $O(n)$ times; it includes $O(1)$ operations for Sum and $O(n)$ operations for the inner loop. Thus, the total number of operations is $O(n^2)$. 
4. Using the big-O notation, describe the worst-case complexity of insertion, retrieval, and deletion in an **ordered binary tree**. Assume that the number of keys stored is \( n \). Briefly justify your answers.

Each operation — insertion, retrieval, and deletion — takes \( O(1) \) steps at each level of the tree. If the ordered binary tree is skewed, then we need \( O(n) \) operations. If the ordered binary tree is balanced, then we need \( O(\log n) \) operations. (As discussed in the class, the height of a balanced binary tree is \( O(\log n) \)).

In the worst case, we need \( O(n) \) operations.

5. Analyze the following using the big-O notation. You may assume the worst-case scenario. The complexity of insertion, retrieval, and deletion in an **unordered** linked list. Only one line explanation is needed for each operation.

If the linked list is unordered, the new element can be inserted in any place. Therefore, we need only \( O(1) \) operations.

If the linked list is unordered, we may have to visit all the nodes to retrieve an element. Therefore, in the worst case, we need \( O(n) \) operations.

For deletion, we need to retrieve the element first, which takes \( O(n) \) operations in the worst case. Once the element is found, the deletion of the node itself takes only \( O(1) \) operations.

6. Consider the following singly-linked list used to implement a stack.

```
ptr1

A -> B -> C -> D -> nil

ptr2
```

Which element should be the top element of the stack? A or D? In other words, which pointer (ptr1 or ptr2) should correspond to the top pointer? Explain.

ptr1 has to correspond to the top element. When an element is popped from the stack, we have to remove the top node. If the top node is pointed by ptr2, we have to set the next pointer of the node that contains C to nil. This cannot be done as ptr2 cannot be used to retrieve the previous node.
7. Consider the two general trees given in Figure 6.10 (page 178) in the textbook. Show the resulting tree (or forest) after processing the equivalence pair \((E, X)\) based on the weighted union rule and path compression.

The resulting tree will have \(E - \rightarrow R\) and \(R' - \rightarrow R\). All others remain the same.

8. For the binary tree shown below

![Binary Tree Diagram]

(a) Write its sequential representation by using the symbol / to mark a NULL link.
(b) Write its sequential representation by using the symbol \( )\) to mark an internal node and / to mark an empty child node of an internal node.
(c) Write its sequential representation by using the symbol \( )\) to indicate the end of a child list.

(a) \(jicb//a//d//heg//f//\)
(b) \(j'i'c'badh'gf\)
(c) \(jicb(a))d))heg(f)))\)

9. (Programming.) Let \(Q\) be a non-empty queue and let \(S\) be an empty stack. Using only the stack and queue abstract data type functions and a single Object variable \(x\), write a function \(\text{reverse}(\text{Queue} \ Q, \text{Stack} \ S)\) to reverse the order of the elements in \(Q\).
reverse(Queue Q, Stack S){
    private Object x;
    while (!Q.isEmpty()){
        x = Q.dequeue();
        S.push(x);
    }
    while (!S.isEmpty()){
        x = S.pop(x);
        Q.enqueue(x);
    }
}

10. (Programming.) Write a recursive function named verbsearch that takes as input a binary tree (not a BST) and a value K, and returns true if value K appears in the tree and false otherwise. The function should look like the following:

bool search(BinNode rt, int K){
    if (rt==null) return false;
    else if (rt.element().key() == K) return true;
    else if (search(rt.left(), K)) return true;
    else return search(rt.right(), K);
}

11. (Programming.) Write a function that takes as input a general tree and returns the number of nodes in that tree. Write the function to use the GTreeNode interface discussed in class.

public GTcount(GTreeNode rt){
    int count=1;
    if (rt==null) return 0;
    GTreeNode temp = rt.leftmost_child();
    while (temp != null){
        count += GTcount(temp);
        temp = temp.right_sibling();
    }
    return count;
}