Homework #3

TA: *Name:* *ID:* .

Course: *Data Structures and Algorithms (VE 281)*

Question 1

Determine whether the following propositions are true or false.

(*a*) A complete tree is a tree where every node has either 0 or 2 children.
True False

(*b*) A complete tree is a tree where every level except the last is necessarily filled, and in the last level, nodes are filled in from left to right.
True False

(*c*) A complete tree is a tree where every node in the left subtree must be of lower priority than that of the root, and every node in the right subtree must be of greater priority than that of the root.
True False

(*d*) Heapsort has worst-case $\Theta(nlogn)$ time complexity.

(*e*) Percolate-up has an average-case time complexity of $\Theta(logn)$.
True False

(*f*) Dequeue is done by simply removing the root.
True False

(*g*) Enqueue in a min-heap is done by inserting the element at the end, and then calling percolate-up.
True False

(*h*) $[2, 13, 8, 16, 13, 10, 40, 25, 17]$ is a representation of a min-heap.
True False

(*i*) $[3, 5, 6, 7, 12, 15, 14, 9, 10, 11]$ is a representation of a min-heap.
True False

(*j*) Proceeding from the bottom of the heap to the top, while repeatedly calling percolateDown() can initialize a min-heap.
True False

(*k*) Proceeding from the top of the heap to the bottom, while repeatedly calling percolateUp() can not initialize a min-heap.
True False

Question 2

For the tree in Figure 1, show the order in which the nodes are visited during the following tree traversals (The nodes in the tree are from A to I):

(*a*) Pre-order depth-first traversal.
Answer:

(*b*) Post-order depth-first traversal.
Answer:

(*c*) In-order depth-first traversal.
Answer:

(*d*) Level-order traversal.
Answer:

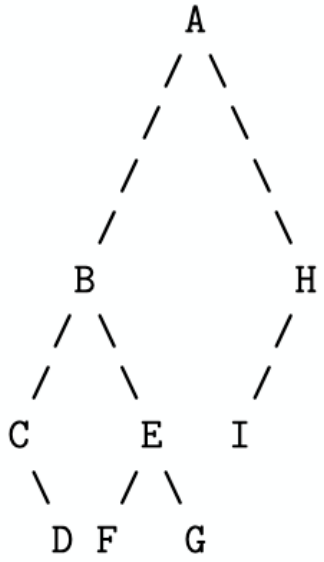

Figure 1: tree

Question 3

Suppose the node of a binary search tree is defined as follows.

```
struct node {
    Key key;
    node * left;
    node * right;
};
```

Implement the following function which gets the predecessor of a given key in the tree:

```
node* getPred(node* root, Key key);
// REQUIRES: The tree rooted at "root" is non-empty.
//       "key" is in the tree rooted at "root".
// EFFECTS: Return the predecessor of "key" in the tree rooted at "root".
//       Return NULL if there is no predecessor
```

You can assume the following function is availabe:

```
node* findMax(node* root);
// REQUIRES: The tree rooted at "root" is non-empty.
// EFFECTS: Return the node with the maximal key in the tree rooted at "root".
```

Complete the getPred() function below:

```
node* getPred(node* root, Key key)
{
    node * leftAncestor = NULL;
    while (root->key != key) {
       if (root->key < key) {



       } else {



       }
    }
    if (root->left == NULL) {


    } else {


    }
}
```

Question 4

Given a key set $\{1, 4, 5, 10, 16, 17, 21\}$, please draw **ONE** BST of height 2, 3, 4, 5, and 6, respectively.

Answer.

Question 5

Use DFS to complete the quest below:
0s and 1s are placed into an $m \times n$ matrix. **Adjacent 1s** in the matrix are connected (here adjacent means that one 1 is on the right, left, above or below of another 1), which forms **components** (an isolated 1 is also a component). Goal: count the number of components in the matrix. Below are 2 examples.

Matrix
10000
01000
00011
00011
: three components(red, blue, and yellow).

Matrix
11011
01100
00110
00011
: two components(blue and red).

Finish the code below:

```
// Your code is allowed to modify the matrix, X.
void dfs(int** X, int i, int j, int m, int n){
    if (                                          ){

        return;
    }
    X[i][j] = 0;
    // Hint: Recur here









}

int numIslands(int** X, int m, int n){
    int count = 0;
    for(int i = 0; i < m; i++){
        for(int j = 0; j < n; j++){
            if (X[i][j] == 1){
                dfs(X, i, j, m, n);
                count ++;
            }
        }
    }
    return count;
}
```