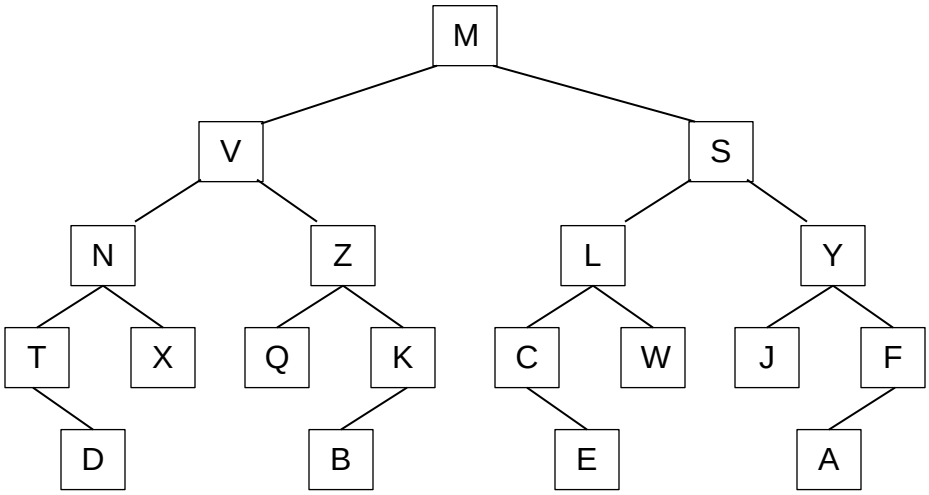
CS 101 Exam 3 **Summer 2017** **Solution**

This exam has 120 points possible (includes 20 built-in extra credit points).

1. For each algorithm below, write each indicated running time as the simplest Big O function of n and any other specified parameters. **[33 points]**

	Best case	Average case	Worst case
Radix sort, with items in range $0...r^d$	**O(d(n+r))**	**O(d(n+r))**	**O(d(n+r))**
Bucket sort, with items in range 0…m	**O(n+m)**	**O(n+m)**	**O(n+m)**
Counting sort, with items in range 0…m	**O(n+m)**	**O(n+m)**	**O(n+m)**
Heap sort	**O(n)**	**O(n lg n)**	**O(n lg n)**
Insert a key into a min-heap	**O(1)**	**O(lg n)**	**O(lg n)**
Remove the min key from a min-heap	**O(1)**	**O(lg n)**	**O(lg n)**
Find the min key in a min-heap	**O(1)**	**O(1)**	**O(1)**
Find a given key in a min-heap	**O(1)**	**O(n)**	**O(n)**
Find a given key in a binary search tree	**O(1)**	**O(lg n)**	**O(n)**
Insert a key into a binary search tree	**O(1)**	**O(lg n)**	**O(n)**
Remove a key from a binary search tree	**O(1)**	**O(lg n)**	**O(n)**

2. Write each indicated traversal of the given tree. **[12 points]**



a. Level-order traversal: **M V S N Z L Y T X Q K C W J F D B E A**

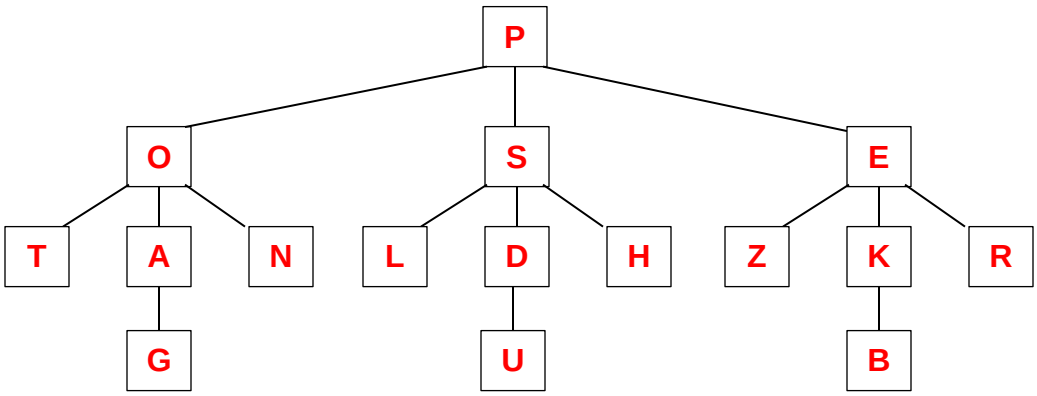
b. Preorder traversal: **M V N T D X Z Q K B S L C E W Y J F A**

c. Postorder traversal: **D T X N Q B K Z V E C W L J A F Y S M**

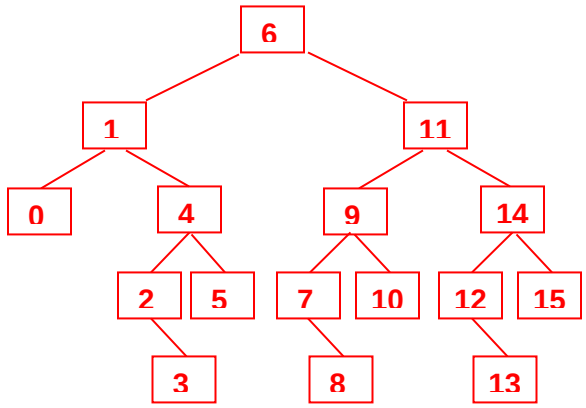
d. Inorder traversal: **T D N X V Q Z B K M C E L W S J Y A F**

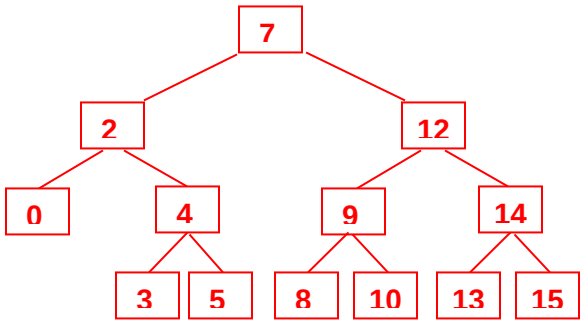
3. Draw <u>one</u> (non-binary) tree that has <u>both</u> of these given traversals. **[6 points]**

Preorder traversal: P O T A G N S L D U H E Z K B R
Postorder traversal: T G A N O L U D H S Z B K R E P

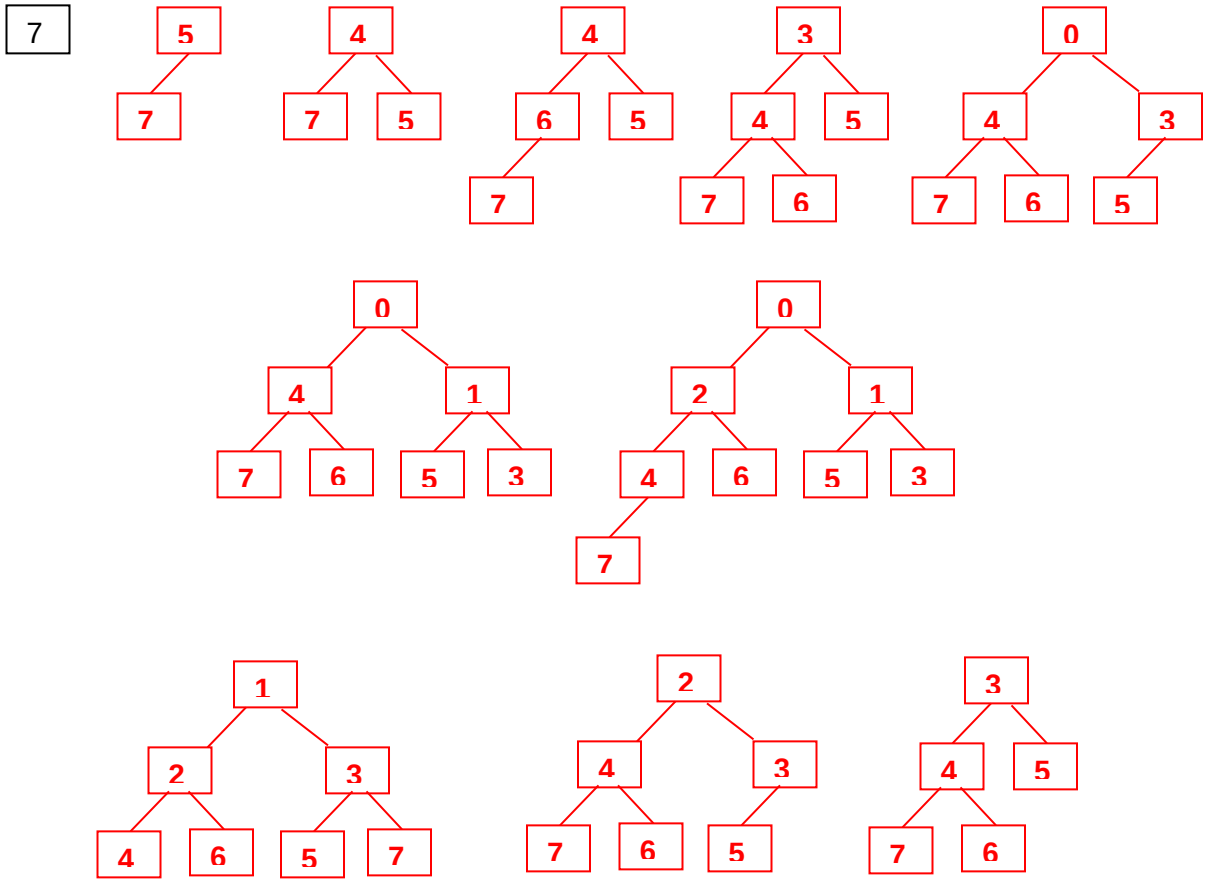


4. Given an initially empty binary search tree. First insert these keys in the order shown: 6, 11, 1, 14, 9, 12, 7, 4, 2, 15, 13, 10, 8, 0, 5, 3. Draw the binary search tree as it should appear after the final insert operation. Next remove these keys in the order shown: 6, 11, 1. Where appropriate in the remove algorithm, use the successors (not the predecessors). Redraw the binary search tree as it should appear after the final remove operation. Make sure that each tree you draw is a correctly formed binary search tree. **[12 points]**

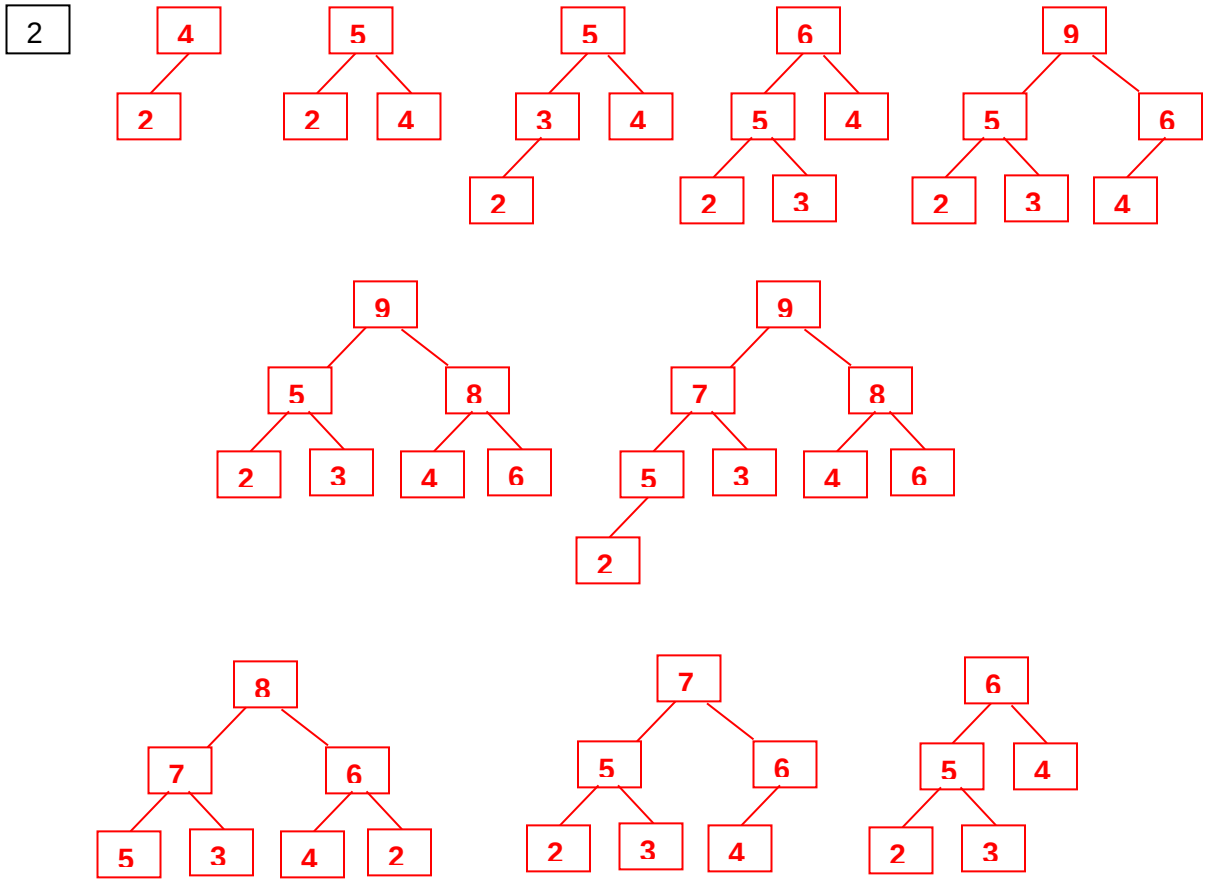




5. Given the min-ordered heap with one node shown below. First insert these keys in the order shown: 5, 4, 6, 3, 0, 1, 2. Draw the min-heap as it should appear after each insert operation. Next do three removeMin operations. Redraw the min-heap as it should appear at the end of each removeMin operation. Make sure that each tree you draw is a correctly formed min-heap. **[12 points]**

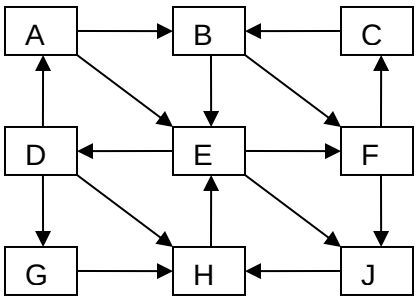


6. Given the max-ordered heap with one node shown below. First insert these keys in the order shown: 4, 5, 3, 6, 9, 8, 7. Draw the max-heap as it should appear after each insert operation. Next do three removeMax operations. Redraw the max-heap as it should appear at the end of each removeMax operation. Make sure that each tree you draw is a correctly formed max-heap. **[12 points]**



7. Answer the questions below regarding the given directed graph. **[21 points]**

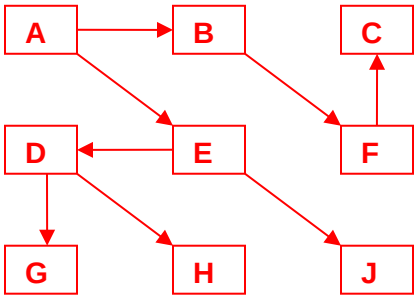
Draw the linked adjacency lists representation (only the outgoing neighbor lists), such that each neighbor list is arranged alphabetically. Also show the adjacency matrix representation.



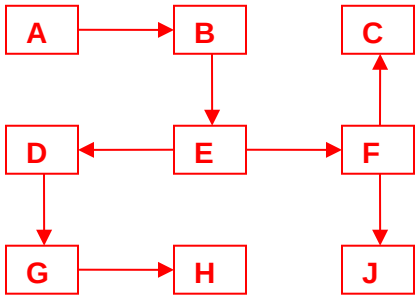
	out	
A		→B→E
B		→E→F
C		→B
D		→A→G→H
E		→D→F→J
F		→C→J
G		→H
H		→E
J		→H

	A	B	C	D	E	F	G	H	J
A	0	1	0	0	1	0	0	0	0
B	0	0	0	0	1	1	0	0	0
C	0	1	0	0	0	0	0	0	0
D	1	0	0	0	0	0	1	1	0
E	0	0	0	1	0	1	0	0	1
F	0	0	1	0	0	0	0	0	1
G	0	0	0	0	0	0	0	1	0
H	0	0	0	0	1	0	0	0	0
J	0	0	0	0	0	0	0	1	0

Starting at node A, draw the *breadth*-first search tree and write the *breadth*-first search order. Next, again starting at node A, draw the *depth*-first search tree and write the *depth*-first search order.

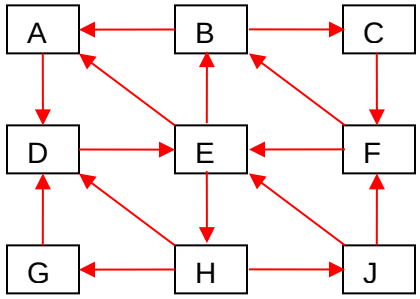


BFS: A B E F D J C G H



DFS: A B E D G H F C J

Draw the reverse of the above directed graph. Also draw its linked adjacency lists representation (only the outgoing neighbor lists), such that each neighbor list is arranged alphabetically. Also show its adjacency matrix representation.



	out	
A		→D
B		→A→C
C		→F
D		→E
E		→A→B→H
F		→B→E
G		→D
H		→D→G→J
J		→E→F

	A	B	C	D	E	F	G	H	J
A	0	0	0	1	0	0	0	0	0
B	1	0	1	0	0	0	0	0	0
C	0	0	0	0	0	1	0	0	0
D	0	0	0	0	1	0	0	0	0
E	1	1	0	0	0	0	0	1	0
F	0	1	0	0	1	0	0	0	0
G	0	0	0	1	0	0	0	0	0
H	0	0	0	1	0	0	1	0	1
J	0	0	0	0	1	1	0	0	0

8. Write three functions that each construct the reverse of a given directed graph. One function uses the adjacency matrix, one uses adjacency lists (outgoing neighbor lists only), and one uses adjacency lists (both outgoing neighbor lists and incoming neighbor lists). **[12 points]**

```
void Reverse (bool M[n][n], bool Mreverse[n][n])          // 4 points
    //  M is adjacency matrix of a directed graph.
    //  Running time is O(n^2) time where n=|V|.
{

    for (x=0;  x<n;  x++)
        for (y=0;  y<n;  y++)
            MReverse[x][y] = M[y][x];



}
```

```
void Reverse (List L[n], List Lreverse[n])                // 6 points
    //  L is outgoing adjacency lists of a directed graph.
    //  Running time is O(n+m) time where n=|V|, m=|E|.
{

    for (x=0;  x<n;  x++)
        Lreverse[x] = null;
    for (x=n-1;  x>=0;  x--)
        for (p=L[x];  p!=null;  p=p->next)
            Lreverse[p->data] = new Node (x, Lreverse[p->data]);
            // inserts x at front of list



}
```

```
void Reverse (List Out[n], List In[n], List (& Outreverse)[n], List (& Inreverse)[n])
                                          // uses pass-by-reference, 2 points
    //  Out and In are outgoing and incoming adjacency lists of a directed graph.
    //  Running time is O(1).
{

    Outreverse = In;
    Inreverse = Out;



}
```