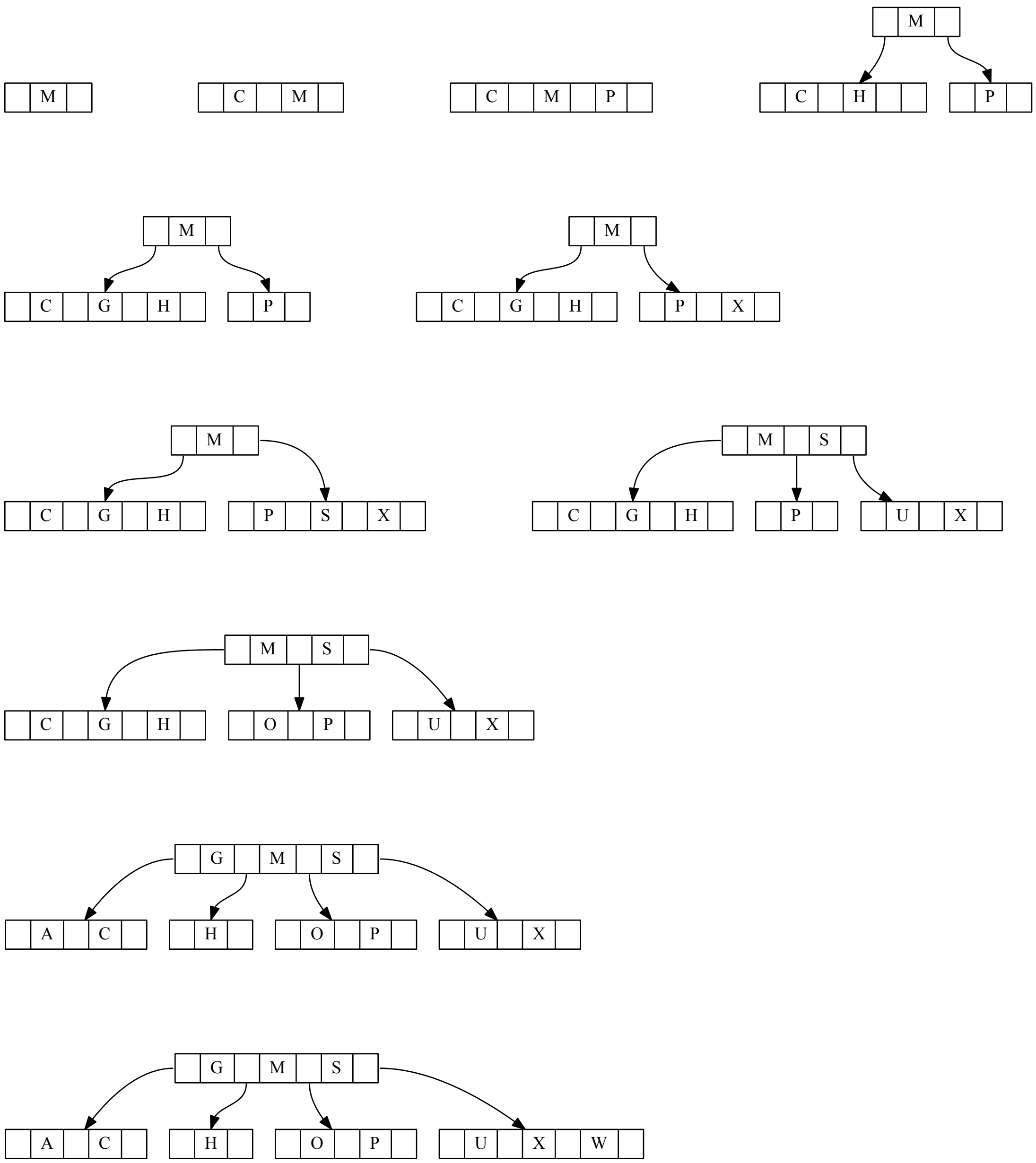
CSCD359 Homework 4, Winter 2012, Eastern Washington University. Cheney, Washington.

Name: Eric Fode **EWU ID:00530214**

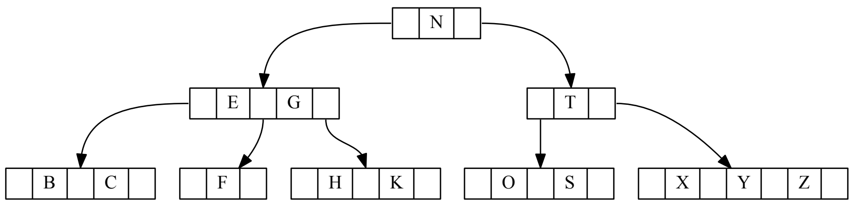
Solution for Problem 1

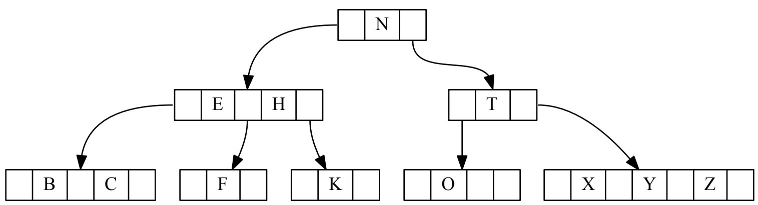


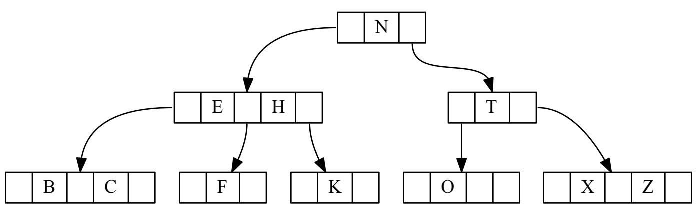
CSCD359 Homework 4, Winter 2012, Eastern Washington University. Cheney, Washington.

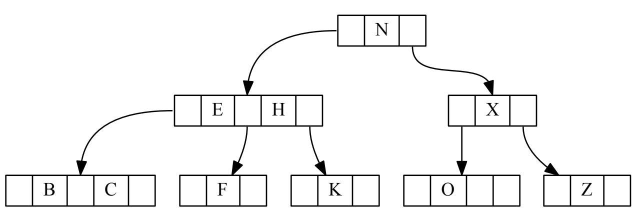
Name: Eric Fode **EWU ID:00530214**

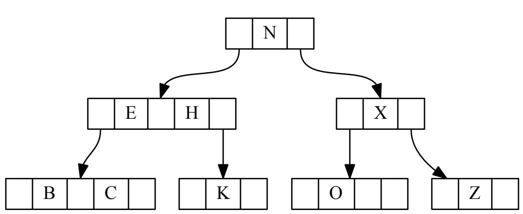
Solution for Problem 2











CSCD359 Homework 4, Winter 2012, Eastern Washington University. Cheney, Washington.

Name: Eric Fode **EWU ID:00530214**

Solution for Problem 3

Part 1 Idea: To find the max of a tree given the root of the tree simply navigate right and down until there are no more nodes to iterate through

Part 1 Algorithm:

bTreeFindMax($root$)

Input: The root of a b-tree that is in memory
Result: The max item in the b-tree

begin
    if $root == null$ **then**
        return $null$;
    childMax $\leftarrow$ `bTreeFindMax(DISKREAD(`$root.lastChild$`));`
    if childMax $== null$ **then**
        return $root$, $root[root.n - 1]$;
    else
        return childMax;

Part 1 Analysis: This should take $O(log(h))$ time including disk operations

Part 2 Idea: To find the predecessor of a node if the index of the key is 0 go to the parent and return the last key that less then the first key. If index greater then 0 check for a child nodes attached to the i'th slot for a child node (the position directly left of i) less then i if one exists the predecessor is the max of the b-tree rooted at the roots child, if no child exists then the predecessor is the key at $i - 1$.

CSCD359 Homework 4, Winter 2012, Eastern Washington University. Cheney, Washington.

Name: Eric Fode **EWU ID:00530214**

Part 2 Algorithm:

BTreePredecessor($node$, i)

Input: A node and it's i'th key

Result: $predecessorNode, j$ where "predecessorNode" is the node that contain the predecessor key which is at position j

```
begin
    if parent.exsists == falsei == 0 then
        return null;
    if i > 0 then
        if root.child(i).exists() then
            return BTreeFindMax(DISKREAD(root.child(i)));
        else
            return root, i − 1
    if i == 0 then
        curNode ← root;
        while true do
            curNode ← DISKREAD(curNode.parent);
            for index ← 0; root[i] > curNode[index] and index < curNode.n; index + + do
            if root[i] > curNode[index] then
                continue;
            else
                return curNode, index;
```

Part 2 Analysis: This should take $O(log(h)t)$ time

Solution for Problem 4

Find the number of items that can fit into memory, load that much of the array into memory, Sort it. Continue loading contiguous pieces of the array into memory and sorting them. After the whole array has been pass through begin a series of merge passes. for $i = numberOfParts$ iterate through contiguous parts of the array merging them together one chunk at a time repeat this process until $i = 1$. At this point the array will be sorted.