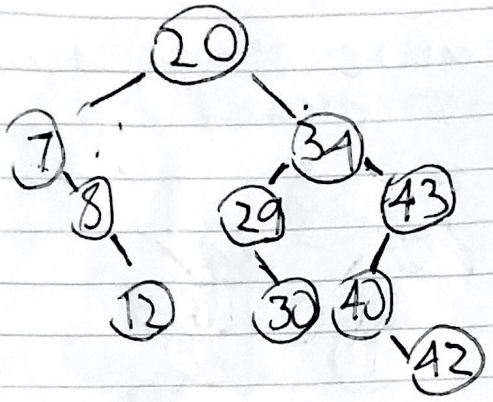
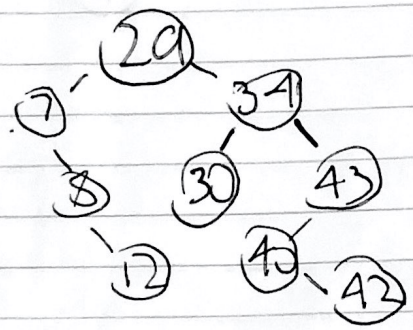
Elijah M.
CS 403
HW4

1)

a)



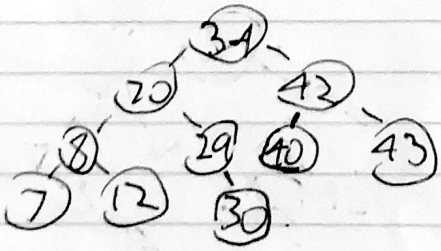
b)



steps: First find MIN of right subtree
-> 29, then replace rootnode with this minimum, after delete the value used to replace and put its successor in its place

20 -> 29 , 29 -> 30

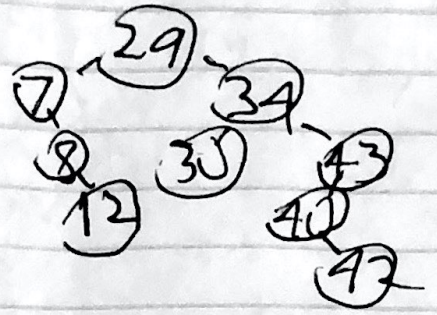
c) 34, 20, 42, 8, 29, 40, 43, 7, 12, 30



2)

a) Yes because the predecessor is always greater

b)

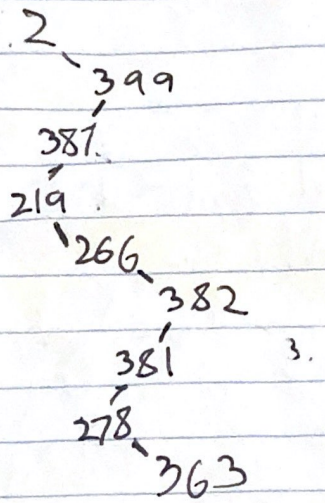


c) Def swap(R, n)

```
    if n.LeftChildren > n.RightChildren:
        temp = n.Parent
        n.Parent = temp.Parent
        n = temp
    else
        n = n.successor
```

3)

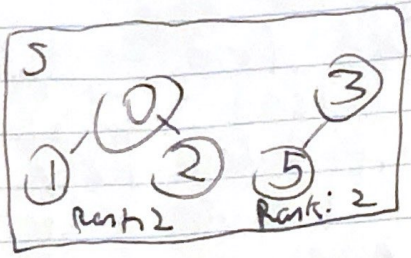
a) valid!



b) Invalid b/c the value 299 cannot appear in that order of the sequence as it is out of range (where min val is 347 and max is 621)

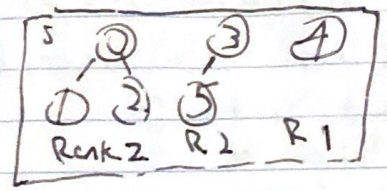
Elijah Monzon
CS 403
HW 4

1)
a) S: {[0,1,2],[3,5]}

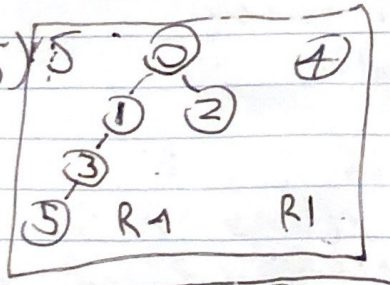


b) Makeset(4): {[0,1,2],[3,5],[4]}
Union(1,5): {[0,1,2,3,5],[4]}
Union(4,5): {[0,1,2,3,5,4]}

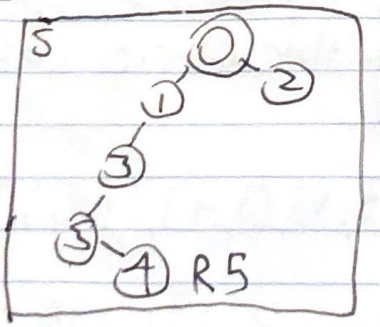
c) Makeset(4):



Union(1,5)



Union(4,5):



```
Def findChildren(G, K) //Finds leaves + level children
  for all children in K:
    if child[i] has children:
      return findChildren(G, child[i])
    else if child[i] equals K:
      return child[i]
    else add child[i] to foundChildren array
  return foundChildren
```

5)

a)
```
Def findParent(G, K)
  Parent = G[K].parent
  if parent.key equals K
    return parent
  else
    return find(G, parent.key)

Def DetectCycle(G, a, b):
  x = findParent(G, a)
  y = findChildren(G, b)
  for all yi in y:
    if yi = x
      return true
  return false
```

b)
```
Def main(G):                    Edge: U, V
  for all edges in G:
    if DetectCycle(G, Ui, Vi)
      print the cycle containing Ui and Vi
    end
```

c) {[1,2],[2,3],[3,4],[4,5,],[5,1]}

$U_i = 1, V_i = 2$

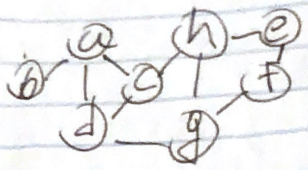
$x = 1, y = 1$

$x = y$? : TRUE

OUTPUT: [1,2,3,4,5]

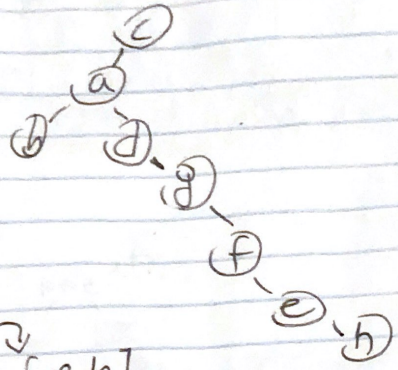
6)

G



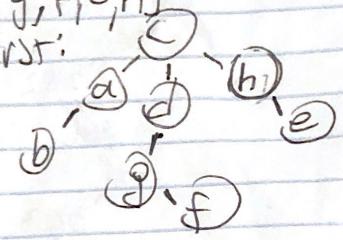
V = c

Depth-first:



Search Path: ↴
[c, a, b, d, g, f, e, h]

Breadth-first:



Search path ↴
[c, a, d, h, b, g, e, f]

G = {[a,b], [a,d], [a,c], [c,h], [c,d], [h,g], [h,e], [e,f], [f,g], [d,g]}

DFS = {[c,a], [a,b], [a,d], [d,g], [g,f], [f,e], [e,h]}

BFS = {[c,a], [c,d], [c,h], [a,b], [d,g], [h,e], [g,f]}

As shown by the lists of edges. it is clear that trees created by BFS and DFS cannot have edges that don't exist in the original graph. This is because the trees are essentially subsets of the graph G