* 1. Valid. If it visits a number lower than 363, it never visits a lower number. If it visits a number higher than 363, it never visits a higher number.
  2. Valid. Same reason.
  3. Invalid. It visits 911 then 912 2 steps later
  4. Valid. Same as a.
  5. Invalid. It visits 347 then 299.

1. 1. smallest(T)

if (empty(T)): return -1

if (leaf(T[0])): return T[0]

smallest(leftChild(T[0]))

* 1. successor(T, k)

if (T[0] == k):

if (right(T[0])):

return smallest(right(T[0]))

if (!parent(T[0])): return -1 // no successor

if (parent(T[0]) > k): return parent(T[0])

if (T[0] > k):

if (!left(T[0])): return -1 //not in tree

return successor(left(T[0]), k)

if (T[0] < k):

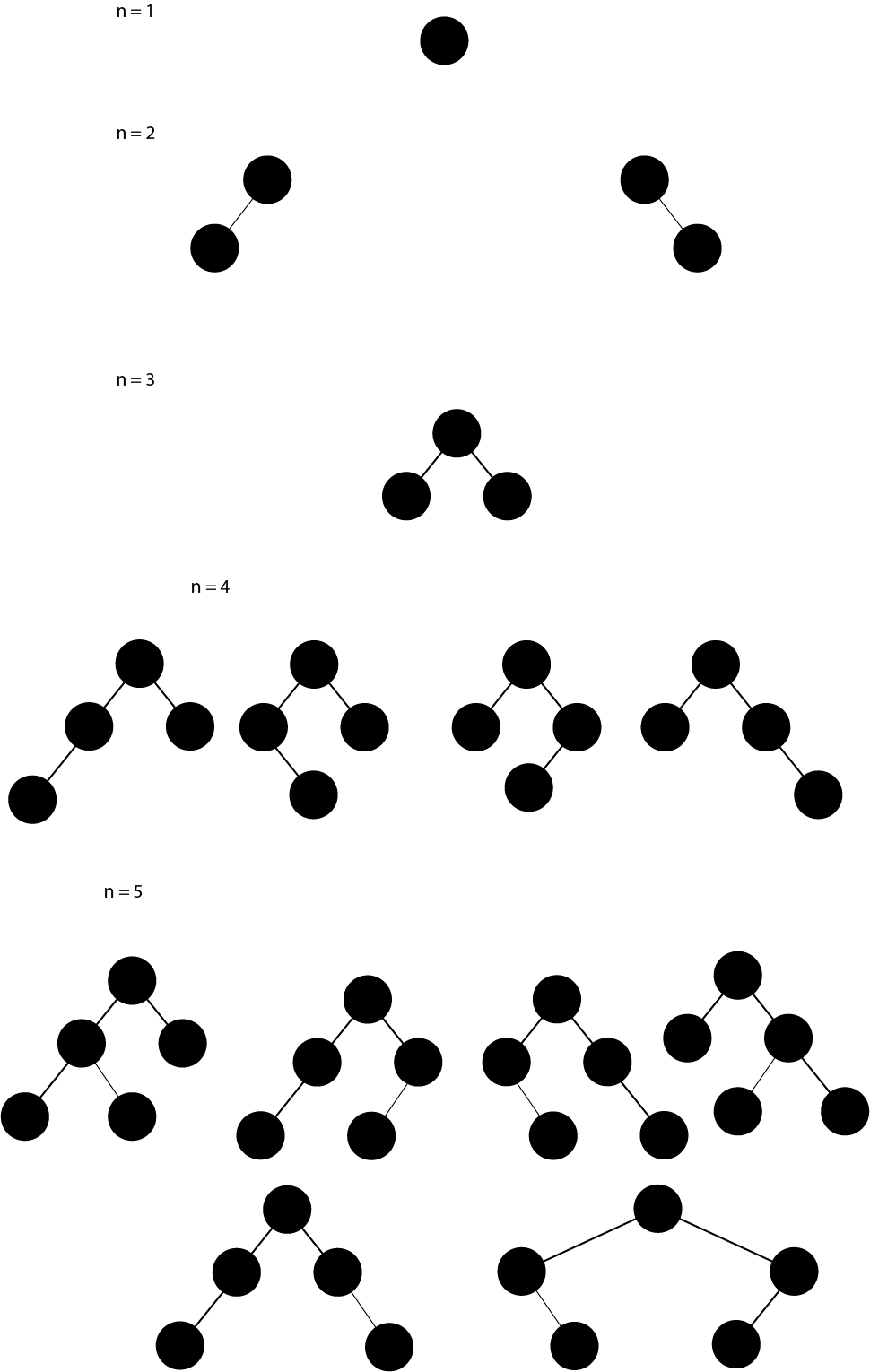
if (!right(T[0])): return -1 //not in tree

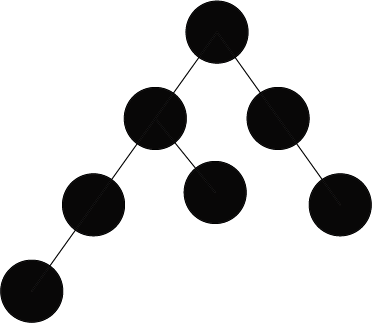
return successor(right(T[0]), k)

* 1. A = {}, B = {10,9,4}, C = {7}, b=10 > c = 7
  2. If a node has a right child, it’s successor is the smallest descendent of that right child. If a node has a left child, it is not the smallest descendent. Therefor, the successor of a node with a right child must have no left child.

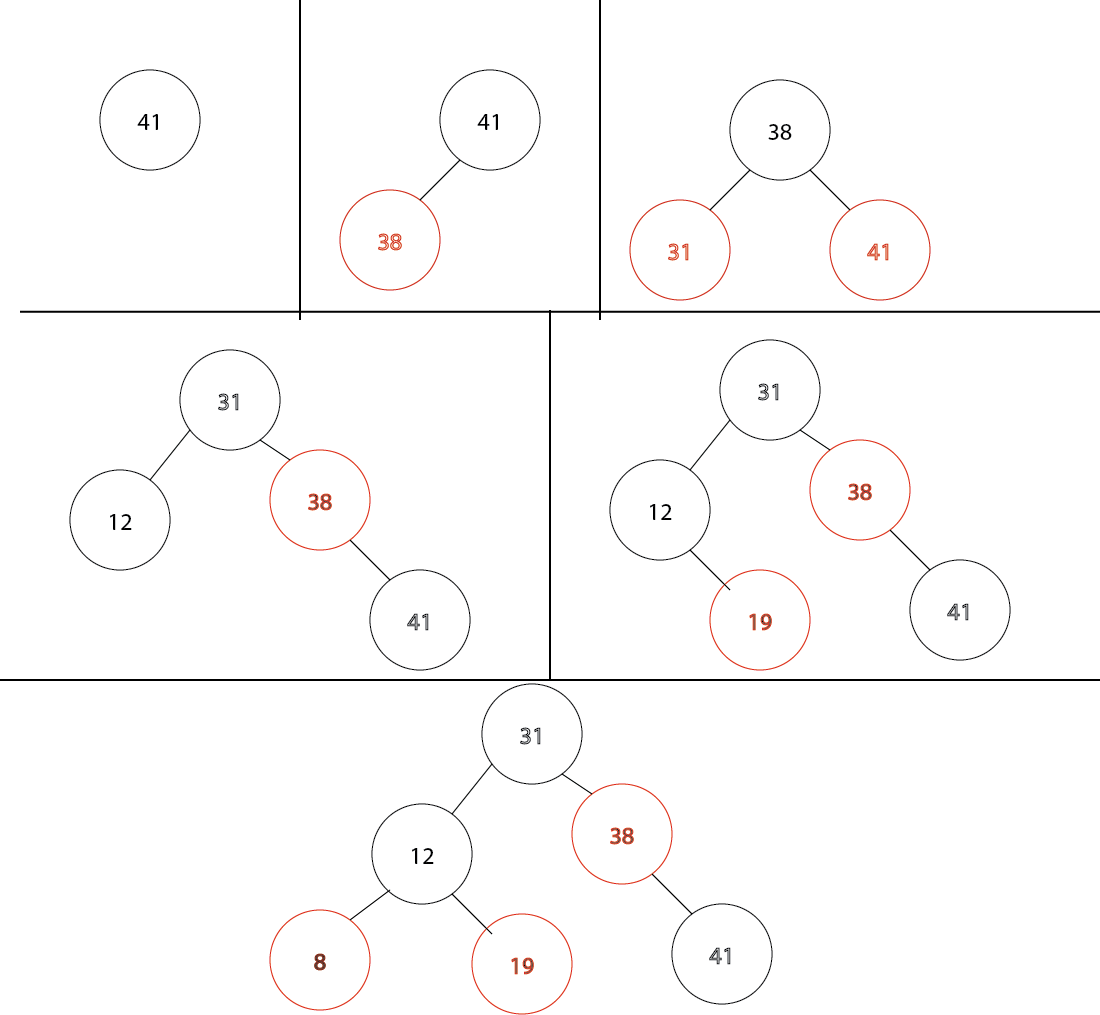
If a node has a left child, it’s predecessor is the largest descendent of that left child. If a node has a right child, it is not the largest descendent. Therefor, the predecessor of a node with a left child must have no right child.











1. 1. 1. Nodes may have more than 1 value
      2. Nodes may have more than 2 children

C(n,k):

for (i=0; i<=min(n,k):

table[i][i] = 1

for(i=0; i<=n; i++):

table[i][0] = 1

for(i=1; i<=n; i++):

for(j=1; j<i; j++):

table[i][j] = table[i-1][j-1] + table[i-1][j]

return table[n][k]

* 1. Time complexity: O(n^2)

Space complexity: O(n\*k)

* 1. A(n, k):

a = 0

for (i=2; i <=n; i++):

for(j=1;j<i; j++):

a++

return a

A(n, k):

a = 0

for(i=2; i<=n;i++):

a+= i

return a

A(n,k):

a = n^2 – 3n

return a



coinCollect(matrix[][], x, y, coins, checked[][], n, m): //this CANNOT BE parallelized.

checked[1][1] = 0

if (matrix[x][y] == 1):

coins++

checked[x][y] = max(checked[x][y], coins)

if (!(x<n) && y<m): coinCollect(matrix, x, y+1, coins, checked, n, m)

if(!(y<m) && x<n): coinCollect(matrix, x+1, y, coins, checked, n, m)

if(x<n && y<m): coinCollect(matrix, x+1, y+1, coins, checked, n, m)

return checked[n][m]



coinCollect(matrix[][], x, y, coins, checked[][], n, m): //this CANNOT BE parallelized.

checked[1][1] = 0

if (matrix[x][y] == 1):

coins++

checked[x][y] = max(checked[x][y], coins)

if (!(x<n) && y<m):

if (matrix[x][y+1] != ‘x’): coinCollect(matrix, x, y+1, coins, checked, n, m)

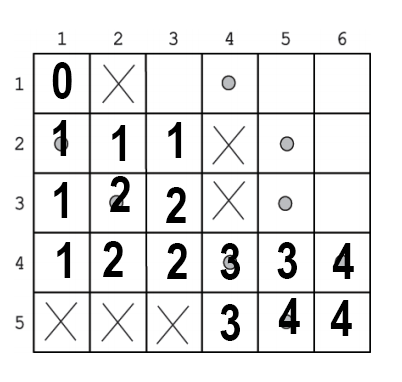
if(!(y<m) && x<n):

if (matrix[x+1][y] != ‘x’): coinCollect(matrix, x+1, y, coins, checked, n, m)

if(x<n && y<m):

if (matrix[x+1][y+1] != ‘x’): coinCollect(matrix, x+1, y+1, coins, checked, n, m)

return checked[n][m]



* 1. 6