1)

a. 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.

b. Valid. Same reason.

c. Invalid. It visits 911 then 912 2 steps later

d. Valid. Same as a.

e. Invalid. It visits 347 then 299.

2)

a.

```
smallest(T)
    if (empty(T)): return -1
    if (leaf(T[0])): return T[0]
    smallest(leftChild(T[0]))
```

b.

```
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)
```

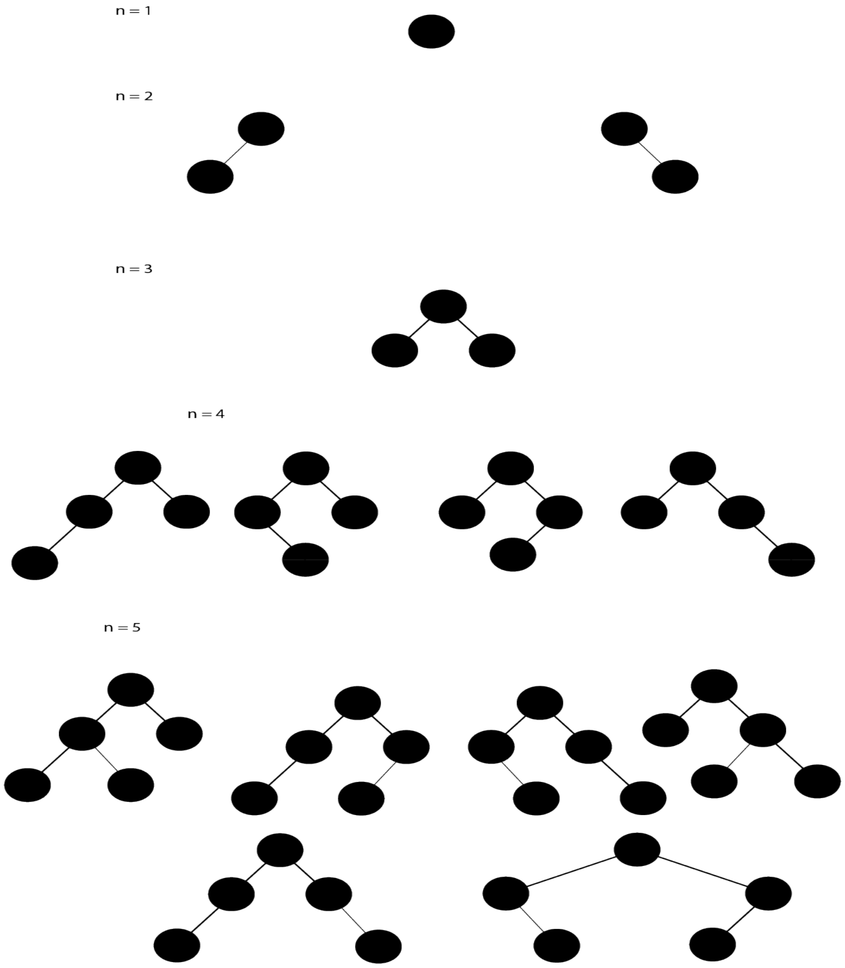
3)

a. A = {}, B = {10,9,4}, C = {7}, b=10 > c = 7

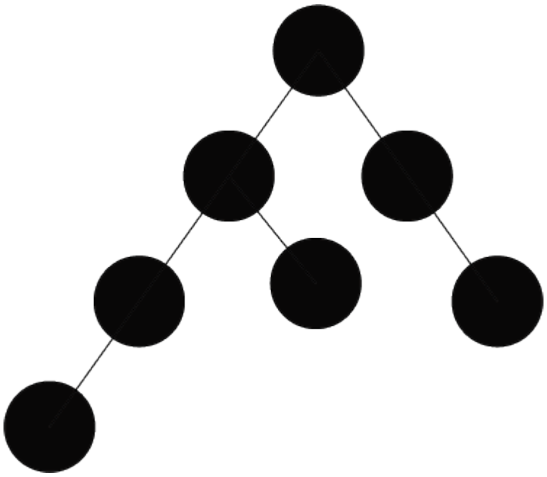
b. If a node has a right child, it's successor is the smallest descendent of that right child. If a node has a left child, it is not the smallest descendent. Therefor, the successor of a node with a right child must have no left child.
If a node has a left child, it's predecessor is the largest descendent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

4)

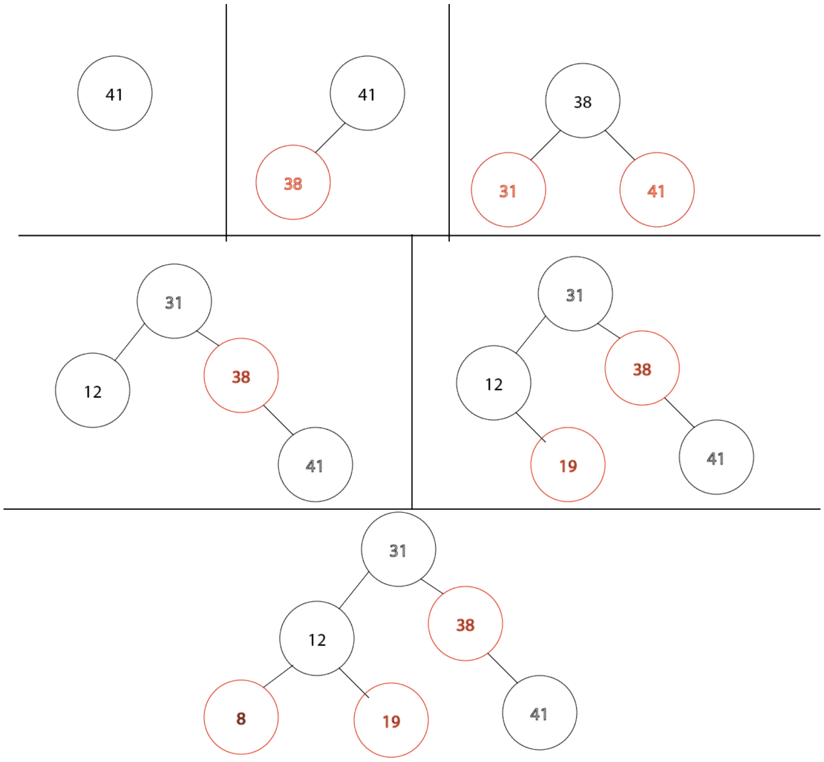
a.



b.



5)



6)

a.

i. Nodes may have more than 1 value

ii. Nodes may have more than 2 children

b. $3 \le \left\lfloor \log_{\frac{m}{2}}\left(\frac{100000001}{4}\right) \right\rfloor + 1 \rightarrow m = 585$

7)

a.

```
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]
```

b. Time complexity: O(n^2)

Space complexity: O(n*k)

c.

```
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
```

8)

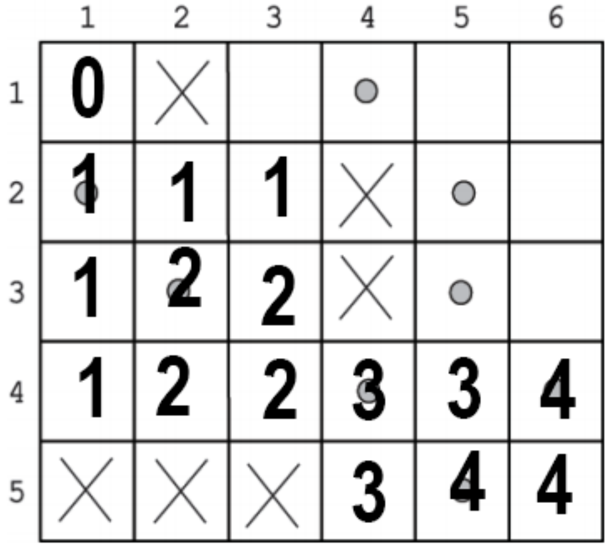
a.

```
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]
```

b.

```
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]
```

c.



	1	2	3	4	5	6
1	0	X		●		
2	1 ●	1	1	X	●	
3	1	2 ●	2	X	●	
4	1	2	2	3 ●	3	4
5	X	X	X	3	4 ●	4

d. 6