Multiplication of 3-sets in array.

In [1]:

```
from operator import mul
from functools import reduce

def deduplicate(values=[]):
    return list(set(values))

def insertionSort(values=[]):
    for idx in range(1, len(values)):
        currentValue = values[idx]
        position = idx

        while position > 0 and values[position - 1] > currentValue:
            values[position] = values[position - 1]
            position = position - 1

        values[position] = currentValue

    return values

def pipe(values):
    if len(values) < 3:
        return "Bad input. Number of elements need to be > 3"
    else:
        print("Input:", values)
        values = deduplicate(values)
        print("Deduplicate:", values)
        values = insertionSort(values)
        print("Sorted:", values)
        values = values[-3:]
        print("Last 3:", values)
        return reduce(mul, values, 1)

values = [-10, -3, 5, 6, 15, 15, -20, -12, 15, 1, 2, 0]
print("Result:", pipe(values))
```

```
Input: [-10, -3, 5, 6, 15, 15, -20, -12, 15, 1, 2, 0]
Deduplicate: [0, 1, 2, 5, 6, -20, 15, -12, -10, -3]
Sorted: [-20, -12, -10, -3, 0, 1, 2, 5, 6, 15]
Last 3: [5, 6, 15]
Result: 450
```

Binary Tree

In [2]:

```
class BinaryTreeNode:
    def __init__(self, value):
        self.left = None
```

```
        self.right = None
        self.value = value

    def setValue(self, value): self.value = value

    def getValue(self): return self.value

    def insertLeft(self, node):
        if self.left == None:
            self.left = BinaryTreeNode(node)
        else:
            tree = BinaryTreeNode(node)
            tree.left = self.left
            self.left = tree

    def getLeft(self): return self.left

    def insertRight(self, node):
        if self.right == None:
            self.right = BinaryTreeNode(node)
        else:
            tree = BinaryTreeNode(node)
            tree.right = self.right
            self.right = tree

    def getRight(self): return self.right

class BST:
    def __init__(self):
        self.root = None;

    def getRoot(self): return self.root

    def insert(self, node, value):
        if node is None:
            self.root = BinaryTreeNode(value);
        else:
            if value < node.value:
                if node.left is None:
                    node.left = BinaryTreeNode(value)
                else:
                    self.insert(node.left, value);
            else:
                if node.right is None:
                    node.right = BinaryTreeNode(value)
                else:
                    self.insert(node.right, value);

def showTree(node):
    if node is None: return

    showTree(node.getLeft())
    print(node.getValue())
    showTree(node.getRight())

def showLeft(node):
    if node is None: return
```

```
    showTree(node.getLeft())
    print(node.getValue())

def showRight(node):
    if node is None: return

    showTree(node.getRight())
    print(node.getValue())

tree = BST()
tree.insert(tree.getRoot(), 7)
tree.insert(tree.getRoot(), 2)
tree.insert(tree.getRoot(), 6)
tree.insert(tree.getRoot(), 5)
tree.insert(tree.getRoot(), 11)
tree.insert(tree.getRoot(), 5)
tree.insert(tree.getRoot(), 9)
tree.insert(tree.getRoot(), 4)

print("Complete tree:")
showTree(tree.getRoot())
print()

print("Sorted left subtree including root:")
showLeft(tree.getRoot())
print()

print("Sorted right subtree including root:")
showRight(tree.getRoot())
print()
```

```
Complete tree:
2
4
5
5
6
7
9
11

Sorted left subtree including root:
2
4
5
5
6
7

Sorted right subtree including root:
9
11
7
```

List of list of filenames

In [3]:

```
names_duplicates = ["/Users/Blockchain/file_A", "/tmp/file_B", "/opt/.file_C.mp4"
names_originals = ["/Users/Blockchain/Documents/copy_A", "/Users/Blockchain/copy_
                   "/Users/Blockchain/Music/copy_C.mp3"]
result = []

for idx in range(max((len(names_duplicates), len(names_originals)))):
    while True:
        try:
            element = (names_duplicates[idx], names_originals[idx])
        except IndexError:
            if len(names_duplicates) > len(names_originals):
                names_originals.append(None)
                element = (names_duplicates[idx], names_originals[idx])
            elif len(names_duplicates) < len(names_originals):
                names_duplicates.append(None)
                element = (names_duplicates[idx], names_originals[idx])

            continue

        result.append(element)
        break

for idx, (copy, original) in enumerate(result):
    if original is None:
        print("{}. Can't find original file for '{}'".format(idx, copy))
    else:
        print("{}. Duplicate '{}' found, original at '{}'".format(idx, copy, orig
```

```
0. Duplicate '/Users/Blockchain/file_A' found, original at '/Users/B
lockchain/Documents/copy_A'
1. Duplicate '/tmp/file_B' found, original at '/Users/Blockchain/cop
y_B'
2. Duplicate '/opt/.file_C.mp4' found, original at '/Users/Blockchai
n/Music/copy_C.mp3'
3. Can't find original file for '/bin/file_not_found.txt'
```

In [ ]: