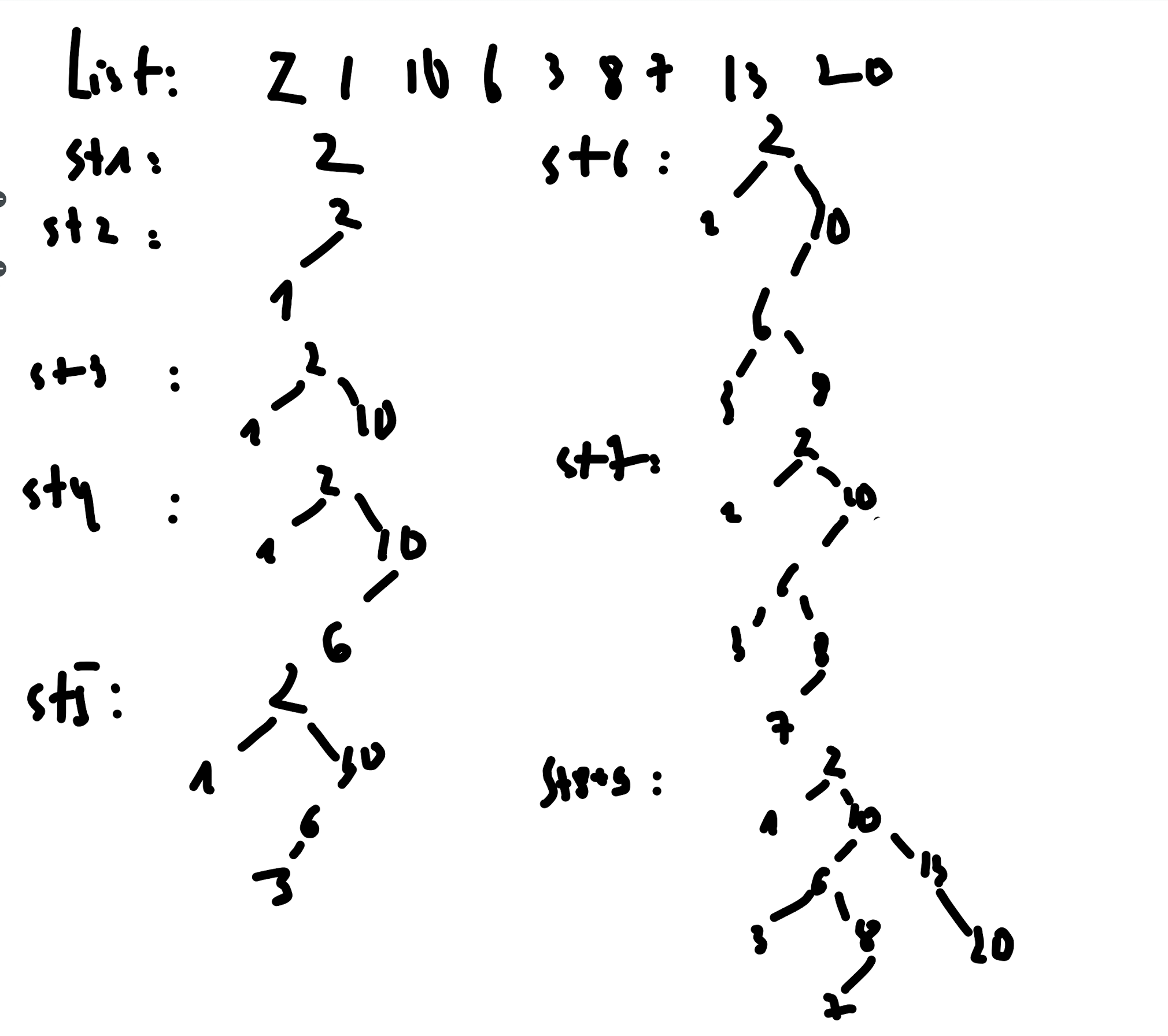
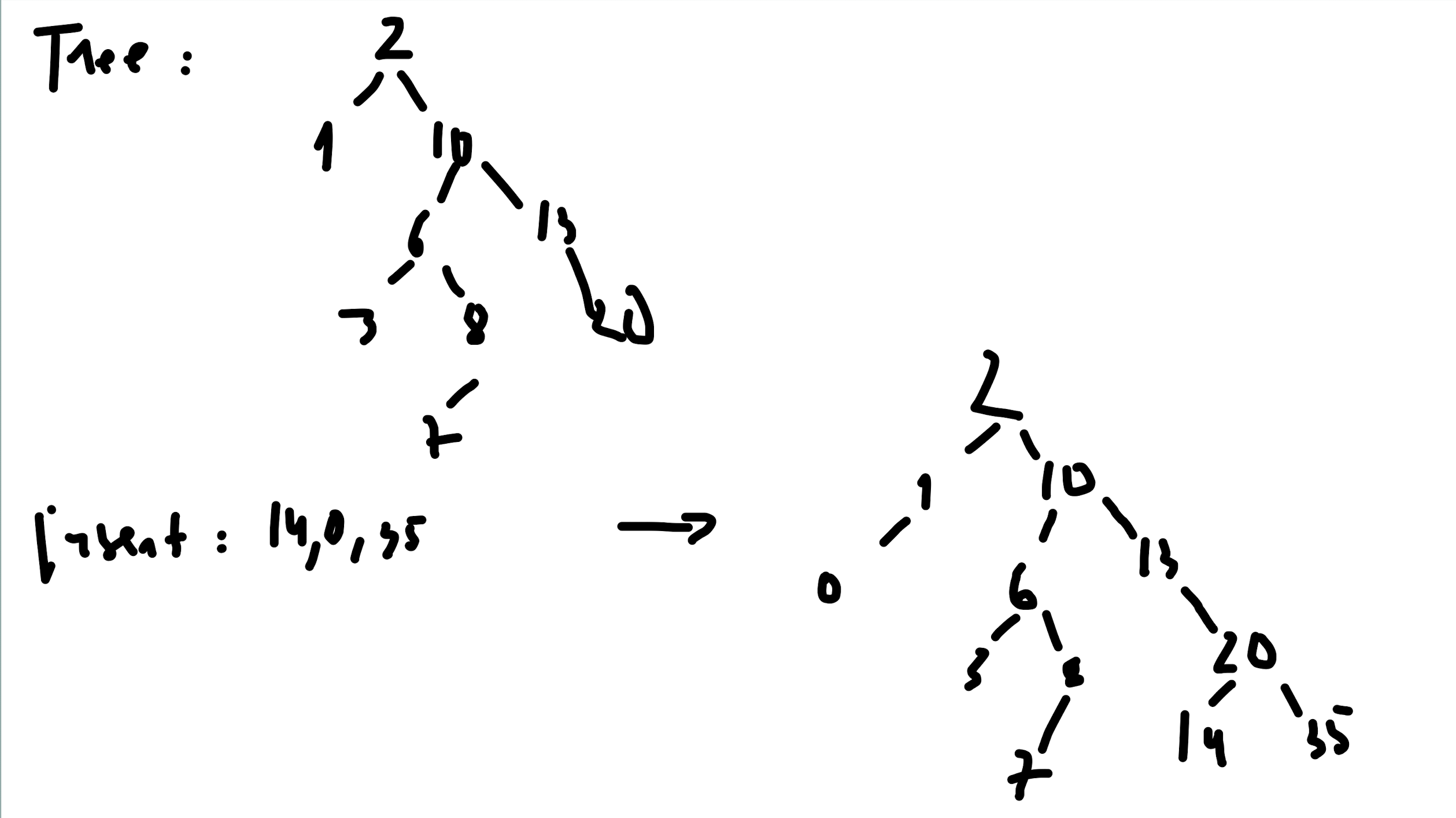
EX2:

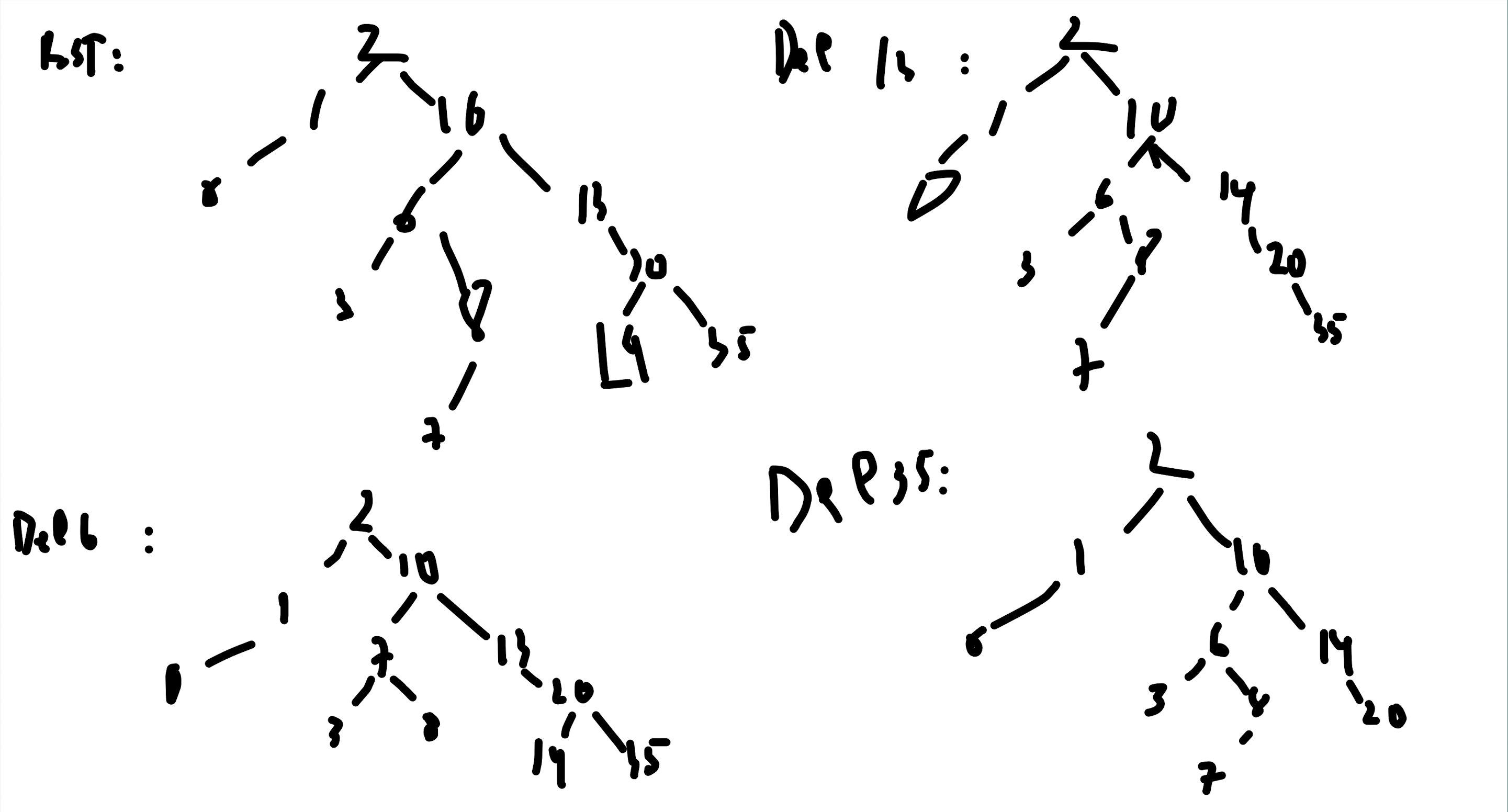
- Insert list to BST



- insert 14, 0, 35 to BST

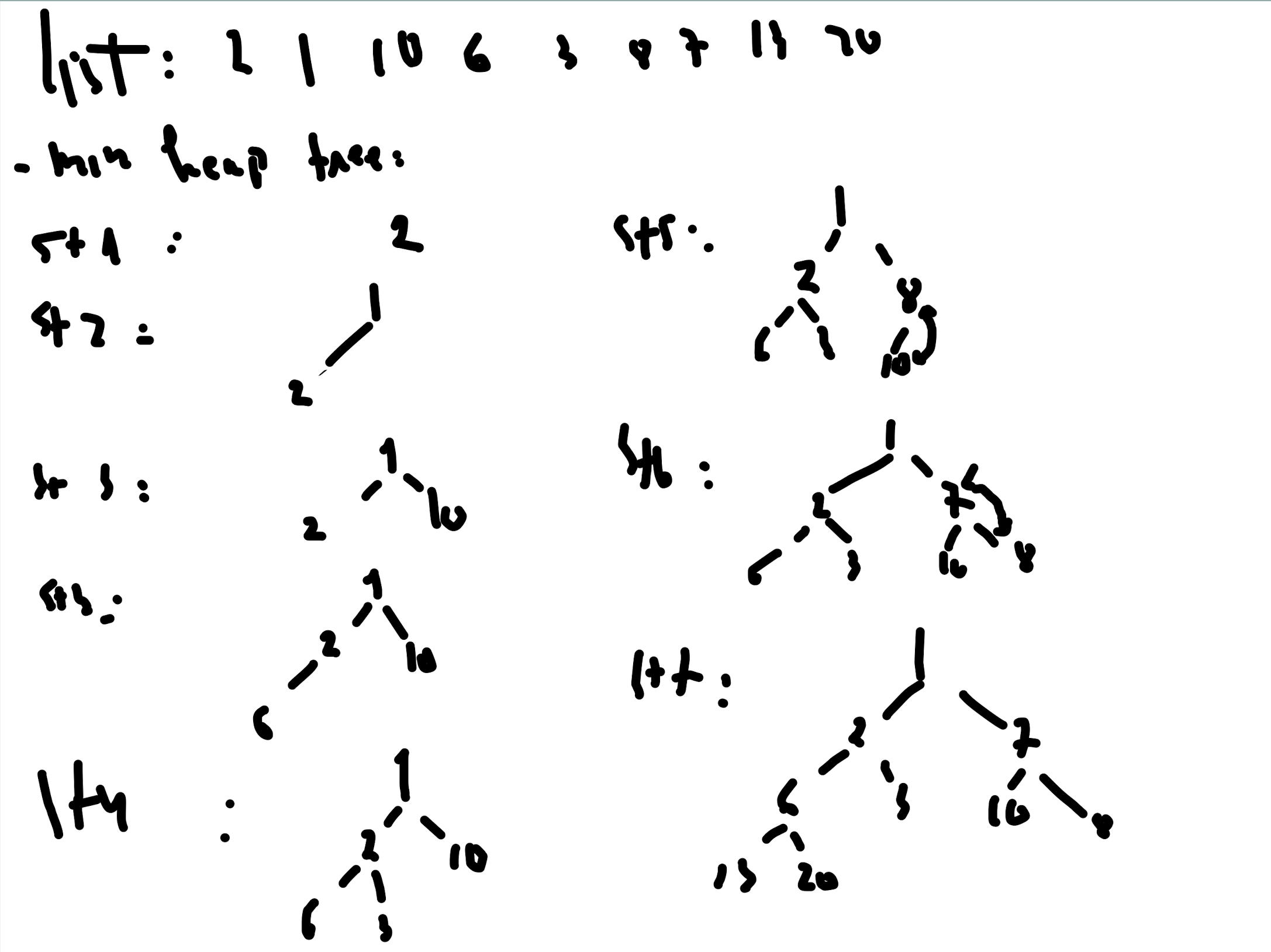


- Delete 6, 13, 35

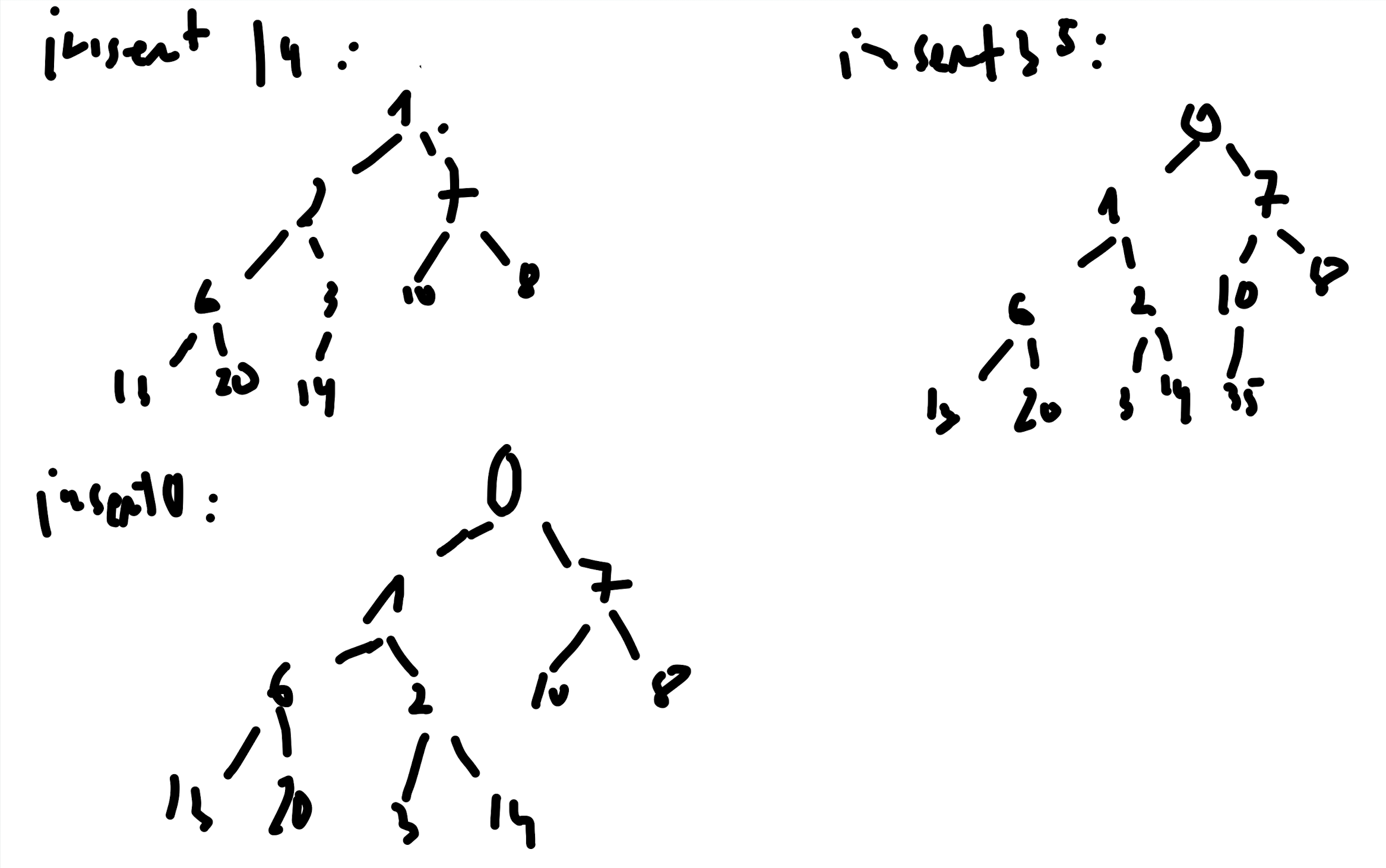


EX3:

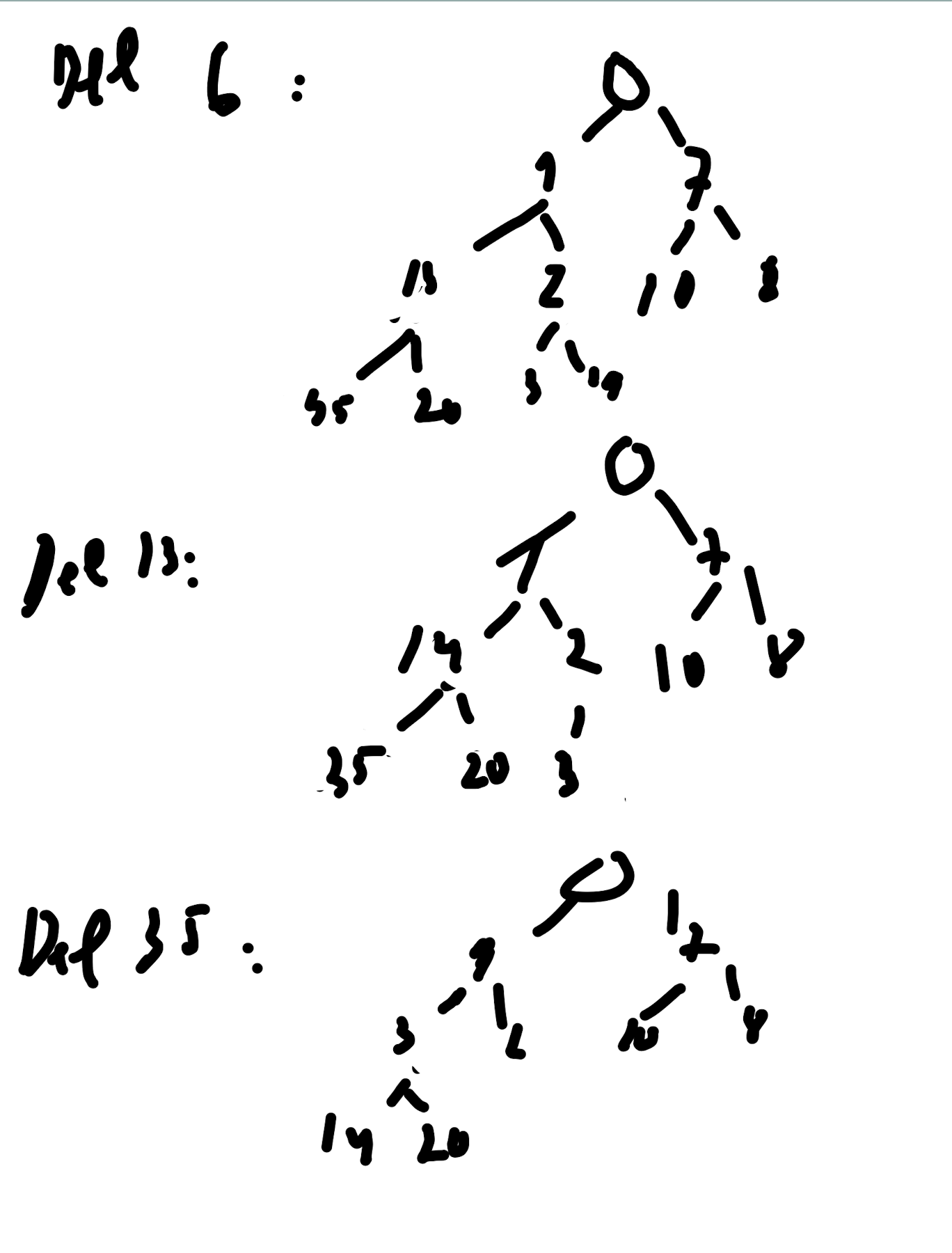
- Insert list: 2 1 10 6 3 8 7 13 20 to min heap tree



- inser14, 0 , 35

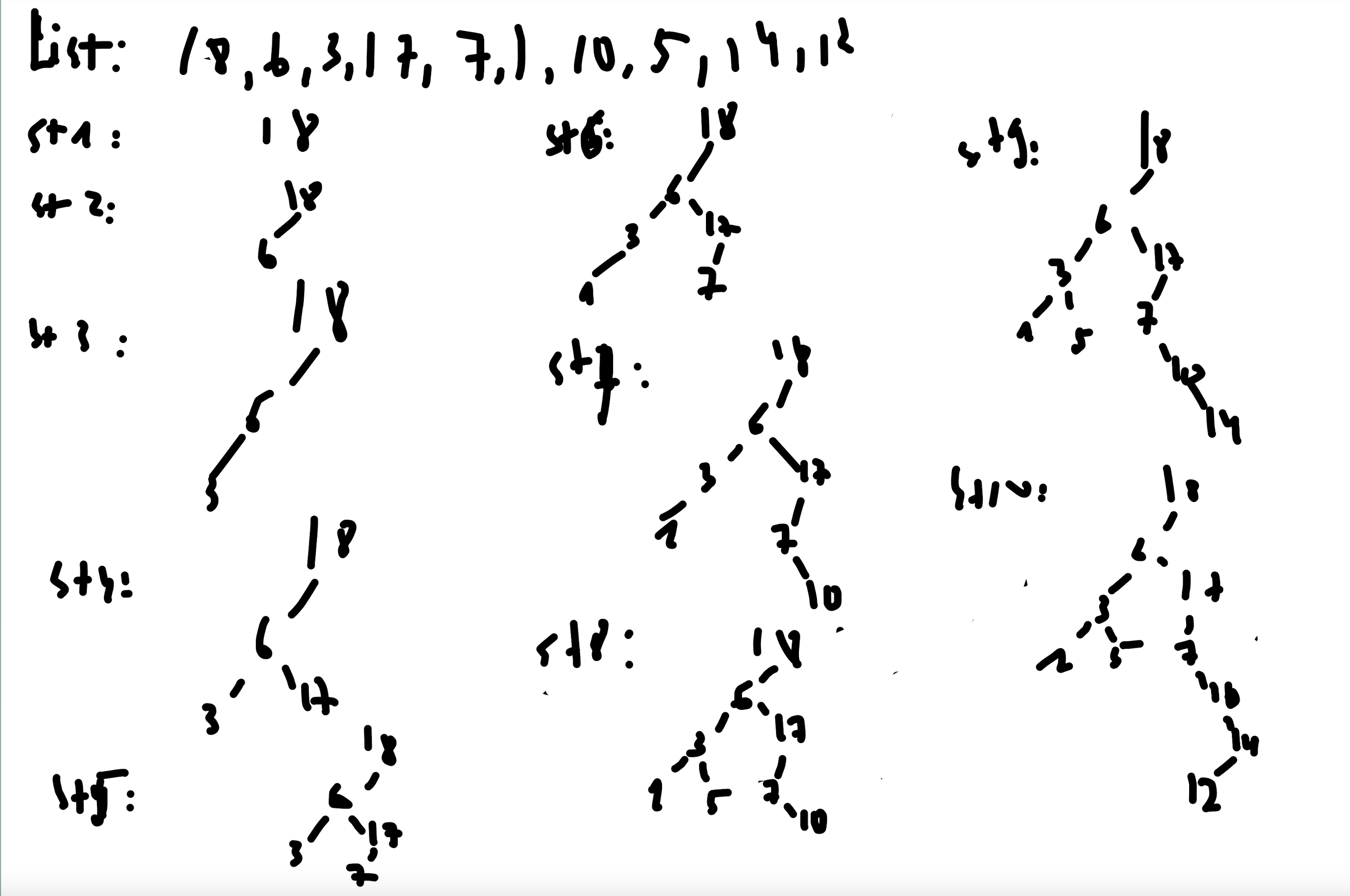


- Del 6, 13, 35



EX4:

List: 18, 6, 3, 17, 7, 1, 10, 5, 14, 12



- Procedure to find the max element and remove it

Procedure searchMaxElement(Node curNode) :

if curNode.rightNode is null : // base case

return curNode;

else searchMaxElement(curNode.rightNode) // recursive

Procedure removeMaxElement() :

Node maxElement = searchMaxElement(Node root)

if maxElement is root:

Node leftTreeMax = searchMaxElement(root.leftNode)

Node tmp = leftTreeMax

leftTreeMax = null

root = tmp

else: maxElement = null

- Procedure to find the min element and remove it

Procedure searchMinElement(Node curNode) :

if curNode.leftNode is null : // base case

return curNode

else searchMinElement(curNode.leftNode) // recursive

Procedure removeMinElement() :

Node minElement = searchMinElement(Node root)

if minElement is root:

Node rightTreeMin = searchMinElement(root.rightNode)

Node tmp = rightTreeMin

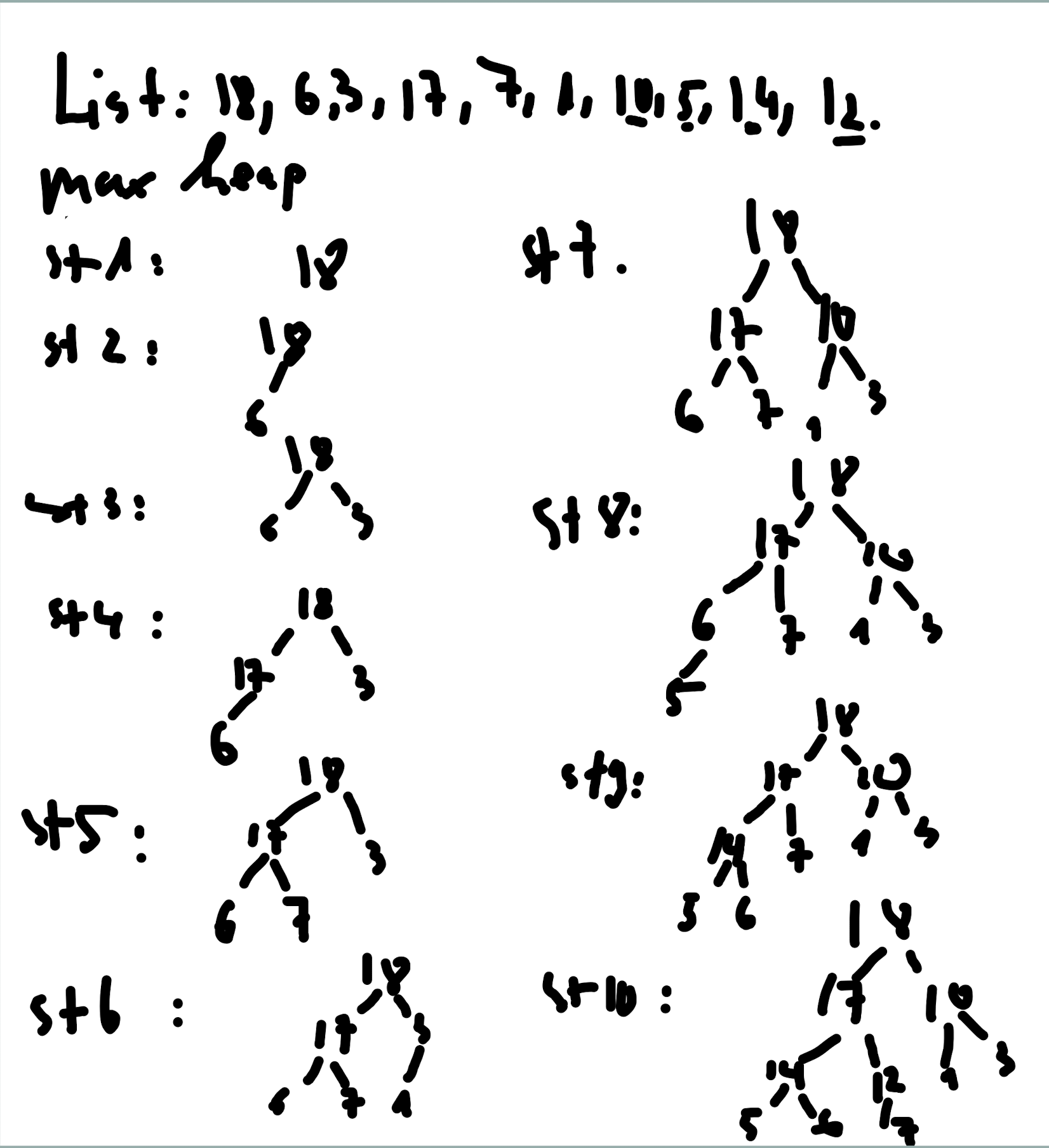
rightTreeMin = null

root = tmp

else: minElement = null

EX5:

List: 18, 6, 3, 17, 7, 1, 10, 5, 14, 12



- Procedure to find and remove the maximum element from max heap tree:

Procedure maxElement() :

if size is 0:

return null

else return heap[0]

Procedure removeMaxElement() :

// replace root of head last element of the heap

heap[0] = heap[size – 1]

size -= 1

heapifyDown(heap,size, 0)

Procedure heapifyDown(heap, size, index) :

leftIndex = 2 \* index + 1

rightIndex = 2 \* index + 2

largest = index

if leftIndex < size and heap[leftIndex] > heap[largest]

largest = leftIndex

if rightIndex < size and heap[rightIndex] > heap[largest]:

largest = rightIndex

if largest is’t index:

heap[index] = heap[largest]

heap[largest] = heap[index]

heapifyDown(heap, size, largest)