Assignment 1: Part 1

1) Suppose you are doing a sequential search of the list [15, 18, 2, 19, 18, 0, 8, 14, 19, 14]. How many comparisons would you need to do in order to find the key 18? Show the steps.

Natural Language
In a sequential search, we iterate through each value in a list and return the index of an item which value matches the search item.

Pseudocode:

for each item in list of values:
    if the item equals the search value, return the items index

if the item was not found return –1

Answer

These are the steps the algorithm would take:

First compare the 0th index with 18, this returns false.
Second compare the 1st index with 18, this returns true.

Two comparisons were required to find the key 18.

2) Suppose you have following list of numbers to sort [19, 1, 9, 7, 3, 10, 13, 15, 8, 12]. Show the partially sorted list after three complete phases of bubble sort.

Natural Language
A phase in the bubble sort algorithm iterates over a list of values, comparing each value with the next and switching the order of the two values if the first value is larger. The bubble sort will continue to initiate new sorting phases until a phase does not switch any items.

Optimization
The minimum required number of comparisons per phase is the length of the list minus the phase count. This is because each phase places the biggest item within the phase's range at the end of the phase's range.

Code
My code for a bubble sort in Python is:

```
def bubbleSort(list):
        switched = True
        passCount = 0
        while switched:
            switched = False
            for i in range(len(list) - passCount -1):
                if list[i]>list[i+1]:
                    temp = list[i]
                    list[i] = list[i+1]
                    list[i+1] = temp
                    switched = True
            passCount += 1
```

```
            print(passCount, list)

list =   [19, 1, 9, 7, 3, 10, 13, 15, 8, 12]
bubbleSort(list)
```

Answer:

These are the first 3 phases:
1 [1, 9, 7, 3, 10, 13, 15, 8, 12, 19]
2 [1, 7, 3, 9, 10, 13, 8, 12, 15, 19]
3 [1, 3, 7, 9, 10, 8, 12, 13, 15, 19]

The result is : [1, 3, 7, 9, 10, 8, 12, 13, 15, 19]

3) Which of these answers is the correct representation of the tree? Show your working out.

Natural Language
In a Binary Search Tree each node has a key and at most two children. One is named the left child and the other the right child. The left child has a key smaller than the nodes key, and all keys in the left subtree are smaller than the nodes key. Similarly, all keys in the right subtree are larger than the nodes key.

Binary Tree Insertion
When inserting a key into a node, the insertion key is compared with the nodes key.

If the insertion value is larger than the node key, then an attempt to insert the key at the right child is made.

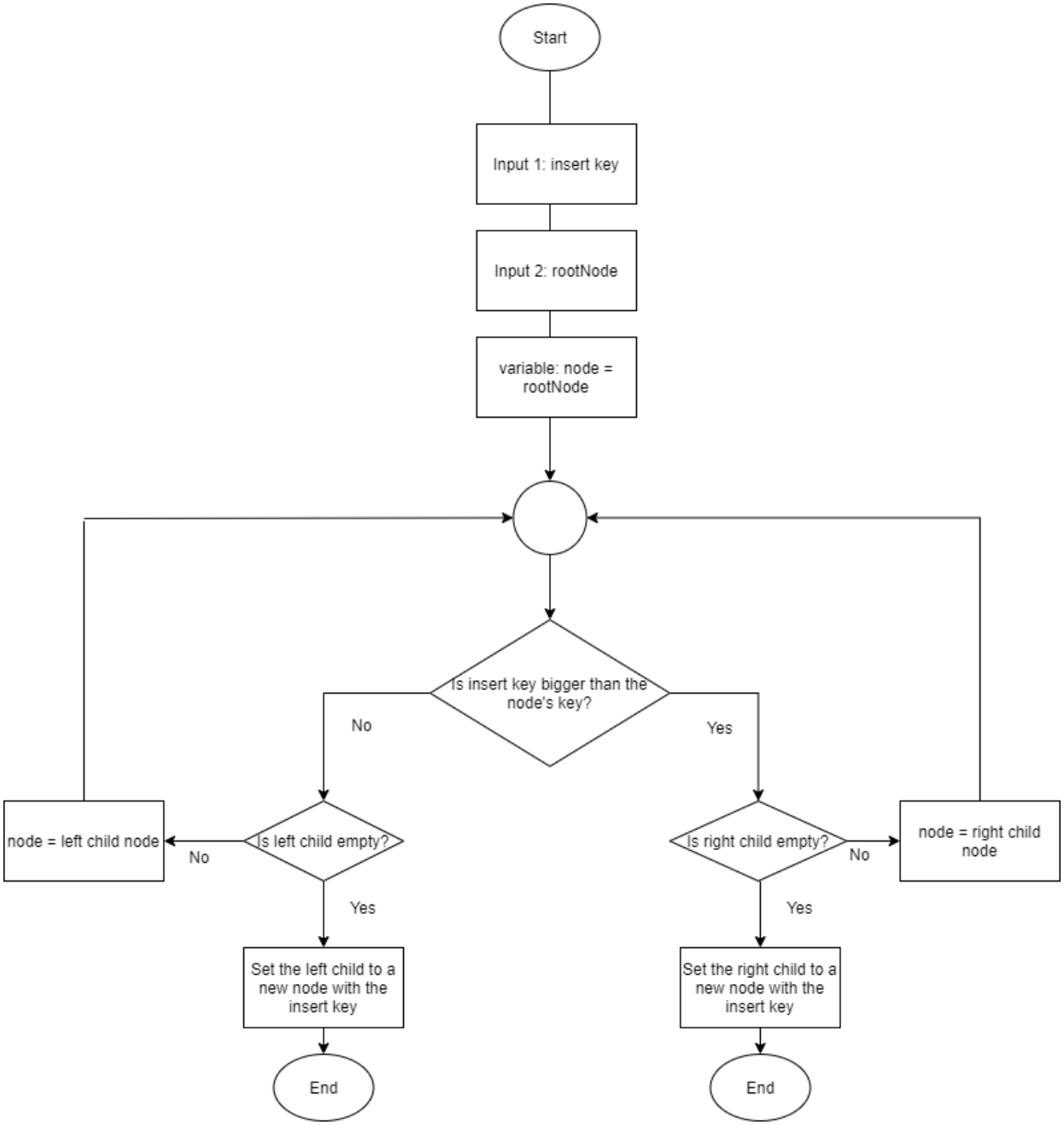
If the insertion key is less than the nodes key then an attempt to insert the key at the left child is made.

If the selected child node is empty then a new node with the key is created and inserted.

Otherwise, we repeat the operation attempting to insert the key into the selected child node.

This process continues until the key is inserted.

Flow Diagram
The flow diagram for insertion into a Binary Tree is:



Answer

Here are the steps per line in the question:

Root node:
[‘a’,[].[]]

After adding ‘b’ and ‘c’ nodes:
[‘a’, ['b', [], []], ['c', [], []]]

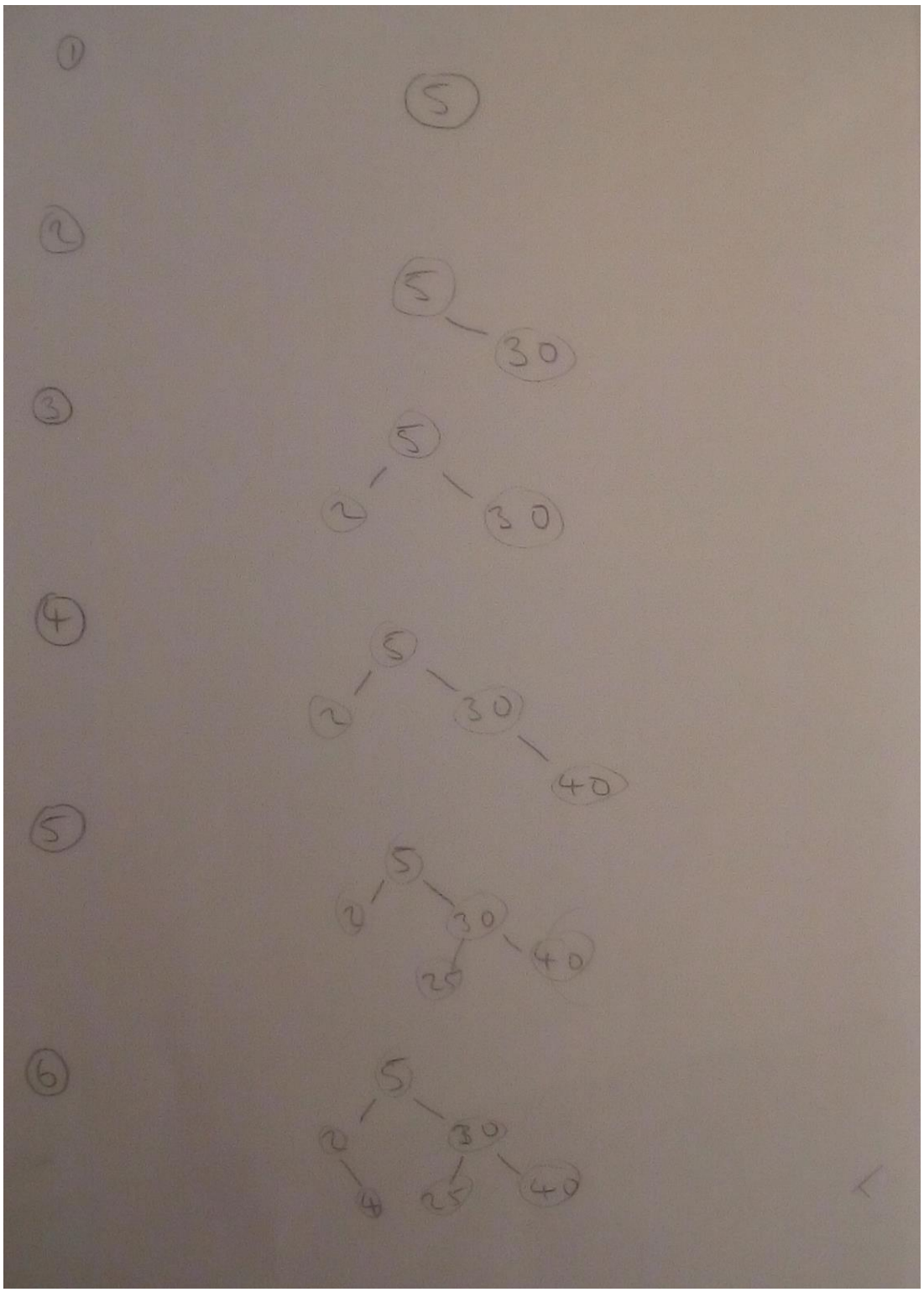
After adding ‘d’ node:
[‘a’, ['b', [], []], ['c', [], [‘d’,[],[]]]

After adding ‘e’ node:
['a', ['b', [], []], ['c', [], ['d', ['e', [], []], []]]]

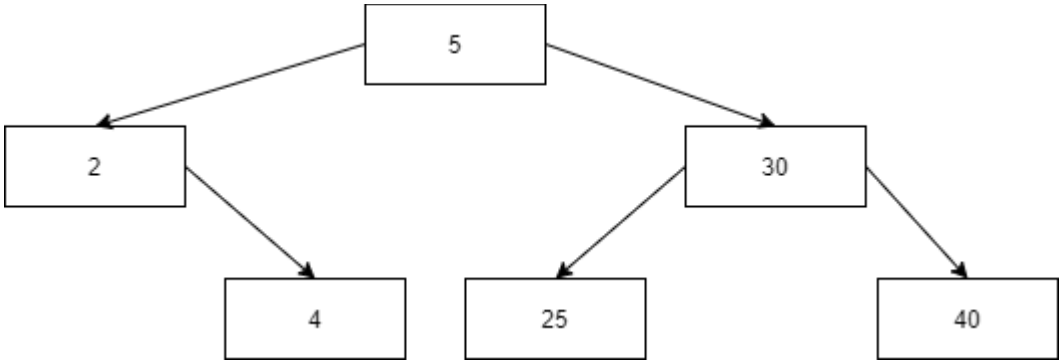
The answer is C

4) Draw a tree showing a correct binary search tree given that the keys were inserted in the following order 5, 30, 2, 40, 25, 4.

Steps



Answer



Code

Here is my Python code for Binary Tree insertion:

```
def createNode(value):
    return [value,[],[]]

def insert(value, tree):
    if( value > tree[0]):
        #addToRight
        if(len(tree[2]) == 0):
            tree[2] = createNode(value)
        else:
            insert(value,tree[2])
    else:
        #addToLeft
        if(len(tree[1]) == 0):
            tree[1] = createNode(value)
        else:
            insert(value,tree[1])

root = createNode(5)

for val in [30, 2, 40, 25, 4]:
    insert(val,root)

print(root)
```