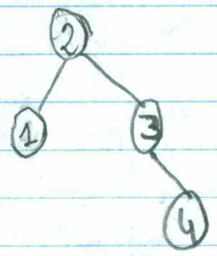
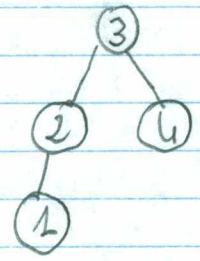
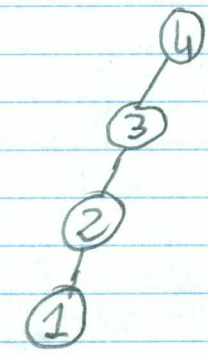
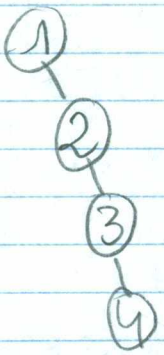
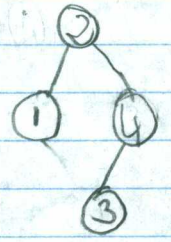
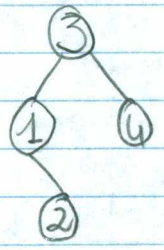
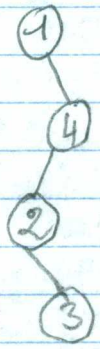
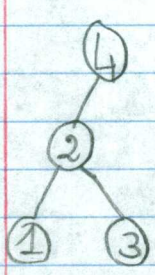
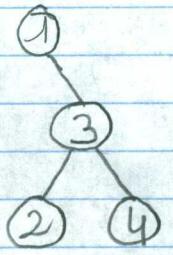
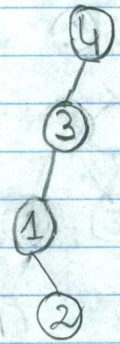
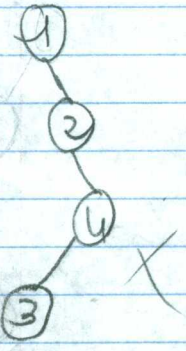
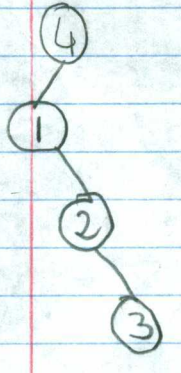
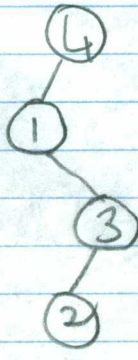
Fidele Donadje
CS 260 002

5.1

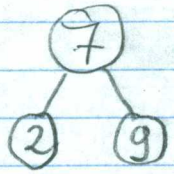







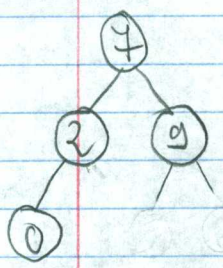








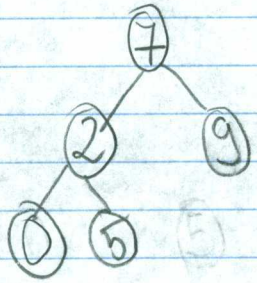
5.2

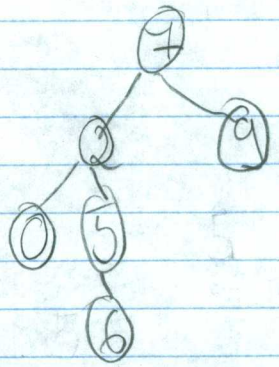
①

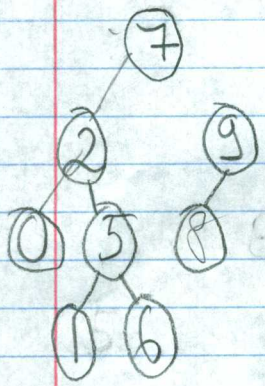
7



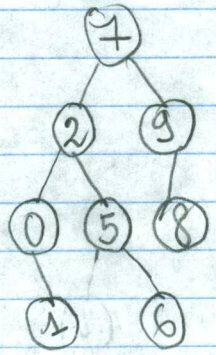




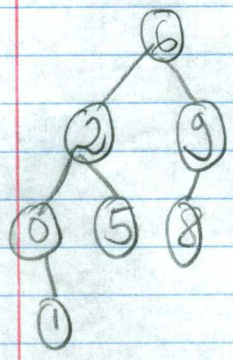




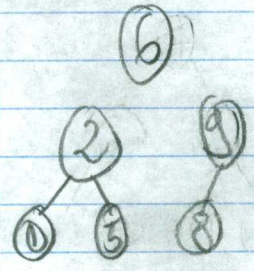
5.3



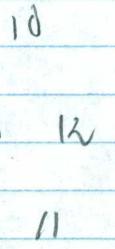
Delete 7



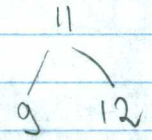
Delete 2



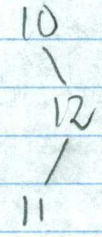
5.4)

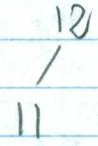


Delete -> 10, 9



Delete -> 9, 10





As we can see, the order matter. Depending on which node we delete, we will get a different tree

#5.5

#5.6 log(n) x n will be the time

n will be the space

5.7

We can use parallel arrays to solve the problem

```
a = first()

Name[], height[], Comp-val[]
Name.add(a -> name)
height.add(a -> height)

While (b = next() != NULL)
{
    Name.add(b -> name)
    height.add(b -> height)
}
comp-val = Impressive_A(height)

// For each index, we get a record by printing Name[i], height[i], compval[i]
```

The runtime will be O(n) and the space complexity 3n (3 tables)

#8:

Space complexity will be $2n$
Insert will be $O(1)$
remove will be $O(n)$ or $O(1)$

#9

We will use a hash table. The hash table will insert and access value in constant time.