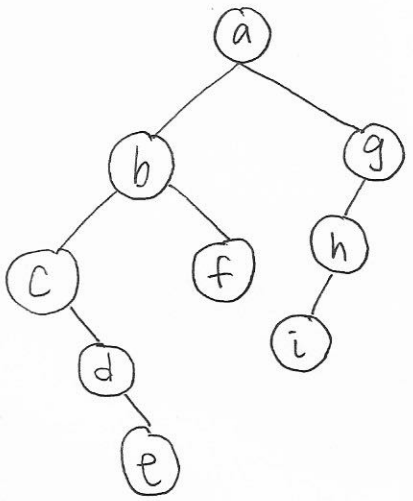
Problem 3.1



Problem 3.2

Fan-out = f, height = h, number of nodes = n

a)

$$n = f^0 + f^1 + f^2 + \cdots + f^h = \sum_{i=0}^{h} f^i = \frac{f^{h+1} - 1}{f - 1}$$

b)

$$n \le \frac{f^{h+1} - 1}{f - 1} \Rightarrow \quad n(f-1) \le f^{h+1} - 1 \Rightarrow \quad n(f-1) + 1 \le f^{h+1}$$

$$\Rightarrow \log(n(f-1)+1) \le \log f^{h+1} \Rightarrow \frac{\log(n(f-1)+1)}{\log f} \le h+1$$

$$\Rightarrow \log_f(n(f-1)+1) - 1 \le h$$

Problem 3.3

a)

$B(1) = 1$

$B(2) = 2$

$B(3) = 5$

$B(4) = 14$

b)

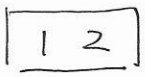
$a_n = a_{n-1} + 3^{n-2}$

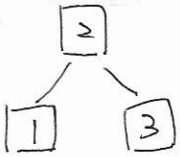
$a_1 = 1$

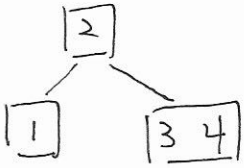
The recurrence start from 1 and each is growing by a factor of 3.

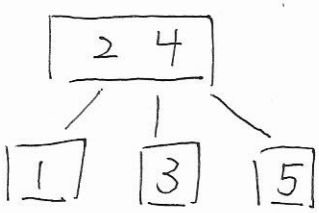
Problem 3.4

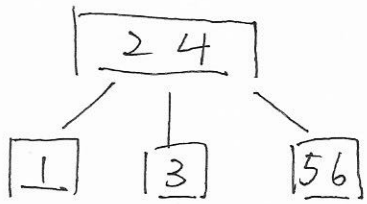
① 

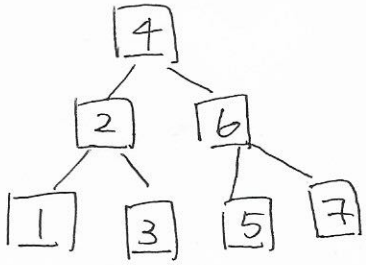
② 

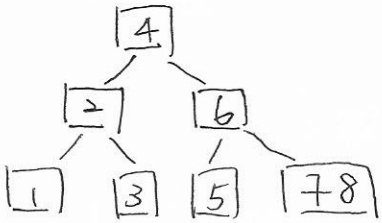
③ 

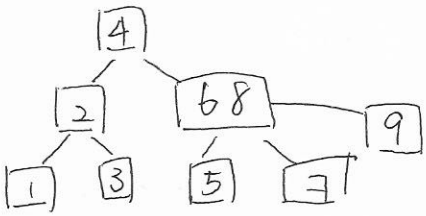
④ 

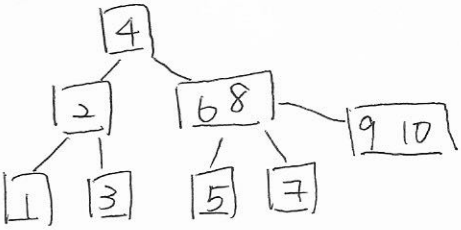
⑤ 

⑥ 

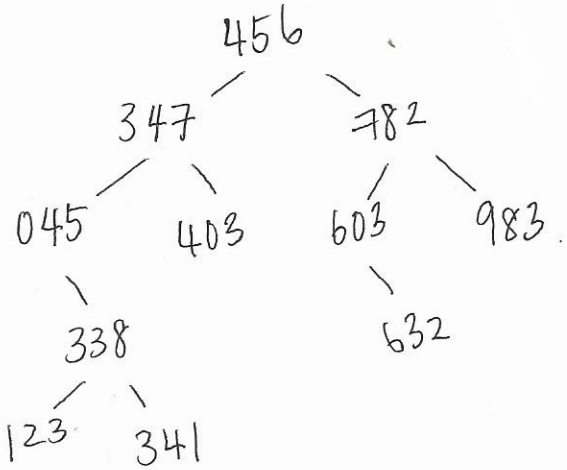
⑦ 

⑧ 

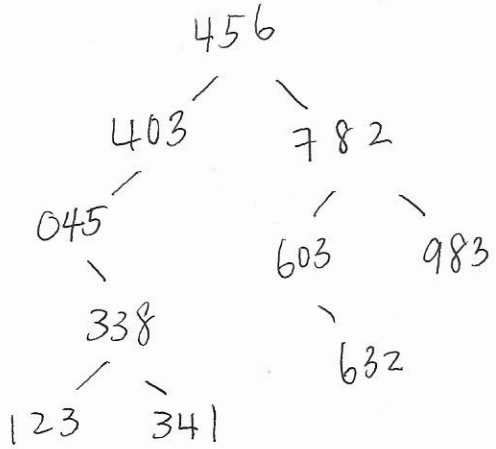
⑨ 

⑩ 

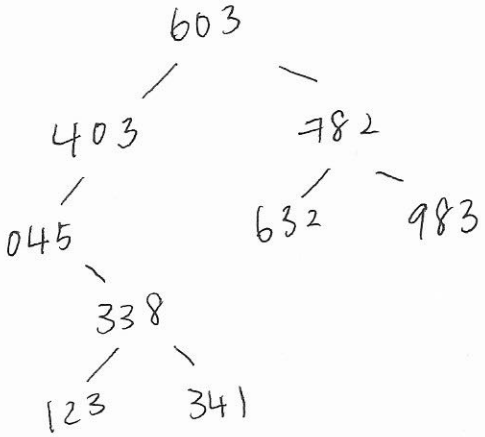
Problem 3.5



delete (347)



delete (456)

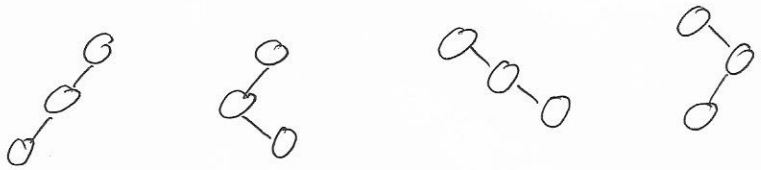


Problem 3.6

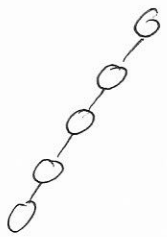
```
fan-out (node p) {
    if (p is null) return 0
    // array to store the fan out of child nodes
    int fan_out_array [order]
    // find the number of children of root
    int rootFanOut = p → numOfChild
    for (int i = 0 ; i<order; i++) {
        fan_out_array [i] = fan_out (p→i)
    }
    int fanOutSub = Max (fan_out_array)
    return Max (fanOutSub, rootFanOut)
}
```

Problem 3.7

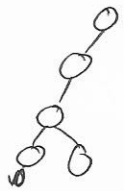
a) Three node tree of loneliness 3



Five node tree of loneliness 5



Five node tree of loneliness 3



b)

```
//root p, c is the counter
loneliness (node p, int c) {
        if (p is null) return 0
        if (left and right child is not null)
                loneliness(left child, c)
                loneliness (right child, c)
        else if ( only right child is not null)
                c++.
                loneliness (right child, c)
        else if (only left child is not null
                c++
                loneliness(left child, c)
}
```

c)

2^{n-1}, since each node could have either left or right child and the root will always be one loneliness.