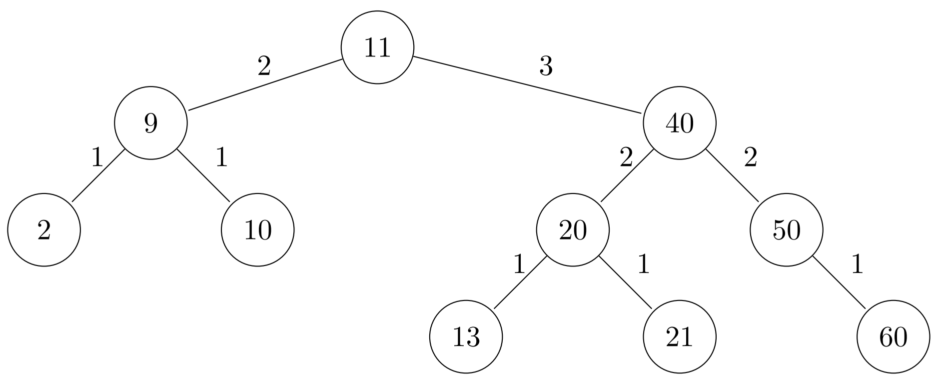
1.

a)



b) the tree is already balanced as all left and right children differ in height by at most 1.

c)

0 $\rightarrow 11 \rightarrow 21$

1 $\rightarrow 9$

2 $\rightarrow 2$

3 $\rightarrow 40 \rightarrow 20 \rightarrow 50 \rightarrow 60 \rightarrow 10$

4 $\rightarrow 13$

2.

a)

since T is perfectly balanced and full we known that any node at height h has $\frac{2^h-2}{2}$ nodes on the left and right side of it. The right child R of the root would then have $h_R = \ell - 1$. The number of, children c on its right side can be calculated by

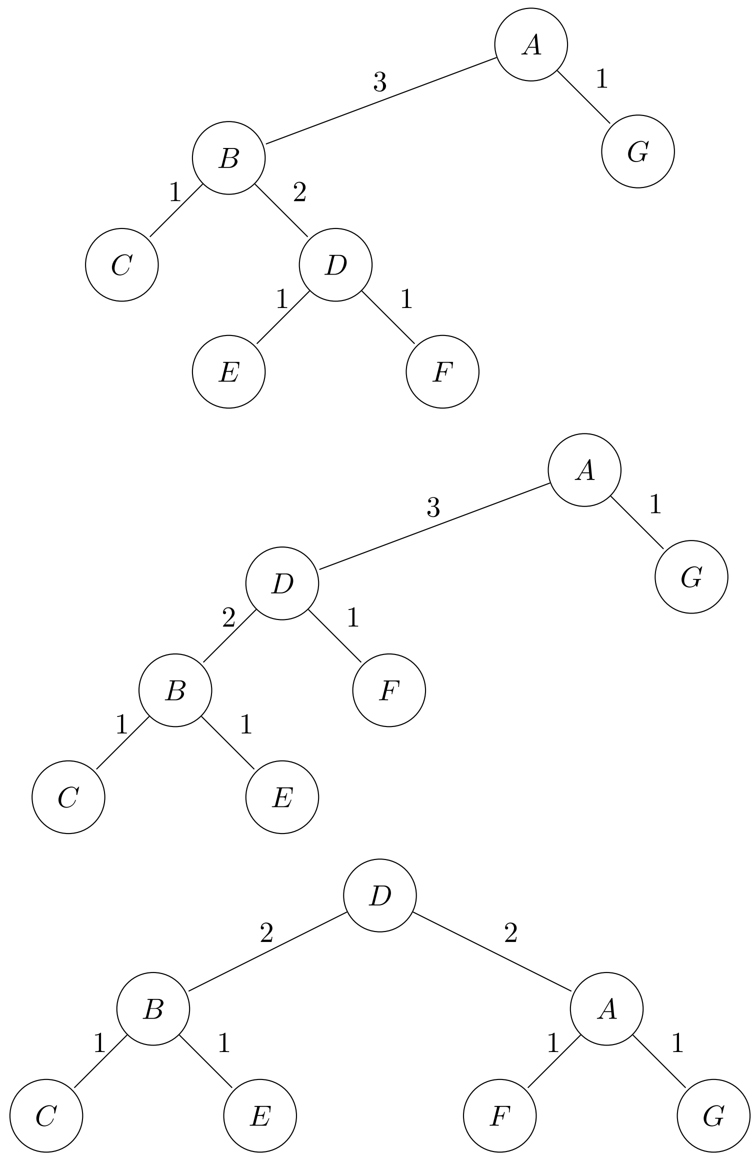
$$\begin{aligned} c &= \frac{2^{\ell-1} - 2}{2} \\ &= \frac{2^{\ell-1}}{2} - \frac{2}{2} \\ &= 2^{\ell-2} - 1 \end{aligned}$$

so the right child of the root is always smaller than $2^{\ell-2} - 1$ elements making the algorithm selecting the right child of the root if one exist, otherwise selecting the root. This has time complexity $O(1)$ because it is not dependant on the size of the tree in any way. We know this is correct because the element we are looking for is smaller than a little less than $\frac{1}{4}$ the elements in T.

$$\begin{aligned} 2^{x-2} - 1 &\approx \frac{2^x - 1}{4} \\ 4(\frac{2^x}{4} - 1) &\approx 2^x - 1 \\ 2^x - 4 &\approx 2^x - 1 \end{aligned}$$

The right child of the root is smaller than its right child which is almost $\frac{1}{4}$ the elements in T.

b)



the balance factors of A,B, and D are 0.

3.

a)

To construct a B Tree from an array where all the elements of the array are leaves in the tree, you would first construct a new array A_1 of size $\lfloor n/2 \rfloor$, where $A_1[i] = \frac{A[i*2]+A[i*2-1]}{2}$. A_1 will be the array of parents to A. you then repeat the proccess until you create an array of size 1 each array being the direct parent of the two elements used to calculate its value. The runtime can be expressed by the function F

$$F(n) = \sum_{i=0}^{log_2 n} \frac{n}{2^i} = n \sum_{i=0}^{log_2 n} \frac{1}{2^i} \implies O(F) = O(n \log n)$$