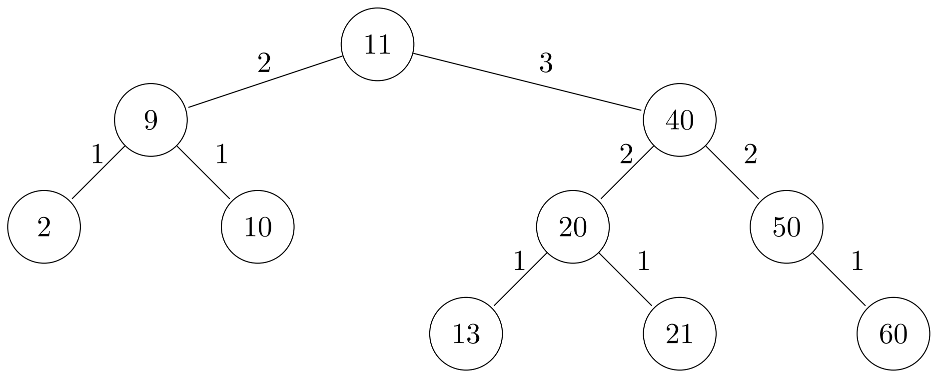
Com S 311 **Homework 3** **John Vincent**

1.

a)



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

c)

0 → 11 → 21
1 → 9
2 → 2
3 → 40 → 20 → 50 → 60 → 10
4 → 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

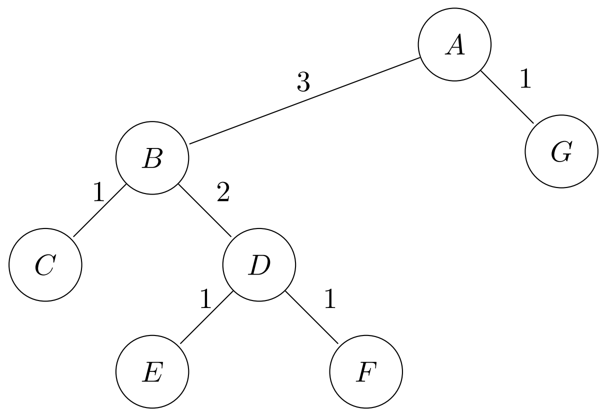
$$
\begin{aligned}
c &= \frac{2^{\ell-1}-2}{2} \\
&= \frac{2^{\ell-1}}{2} - \frac{2}{2} \\
&= 2^{\ell-2} - 1
\end{aligned}
$$

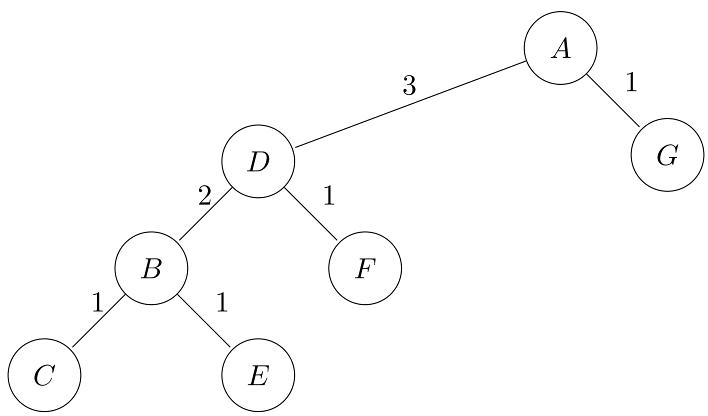
so the right child of the root is always smaller than $2^{\ell-2} - 1$ elements making the algorithm selecting the right child of the root if one exist, otherwise selecting the root. This has time complexity $O(1)$ because it is not dependant on the size of the tree in any way. We know this is correct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

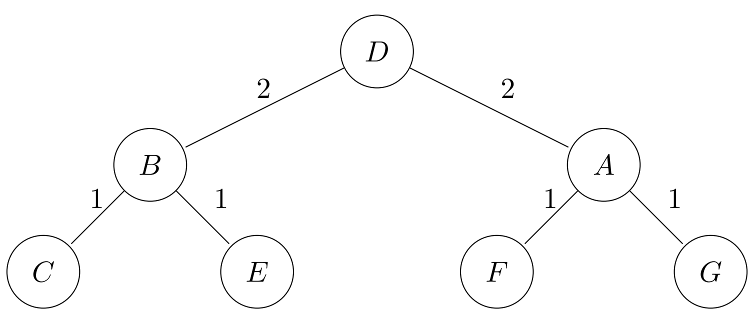
$$2^{x-2} - 1 \approx \frac{2^x - 1}{4}$$
$$4(\frac{2^x}{4} - 1) \approx 2^x - 1$$
$$2^x - 4 \approx 2^x - 1$$

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

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

b)

I would make an altered B-Tree where each node contains 5 values;

1. Sum of left children S_l.
2. Sum of right children S_r.
3. Number of left children C_l.
4. Number of right children C_r.
5. value v.

The increment function would then have an algorithm that does the following. starting at the root $v = v + val$, then if $C_l + C_r = i$ return, else if $C_r > i$ go to right child, else $i = i - C_r$ and go to left child. SS would then have the following algorithm where T is the sum that will be returned. starting at the root node, for all nodes until an end is reached, $T = T + v$ then if $\ell = 0 : T = T + S_l + S_r$ and end, else if $\ell = C_r + C_l$ end, else if $\ell \geq C_r : \ell = \ell - C_r$ go to left child, else go to right child. Both operations reduce the passed in index by the largest factor of 2 possible and run until the index is zero, which means they are both bounded by the function $C + \log n$ where C is some constant.

4.

A standard B-Tree supports all of these operations in $O(\log n)$ time so I would just make a textbook B-Tree