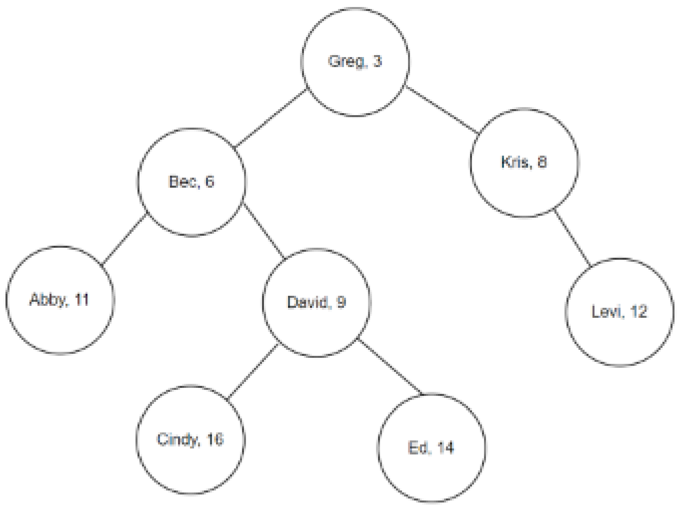
Question 1: Quasi-balanced Search Trees

Part A

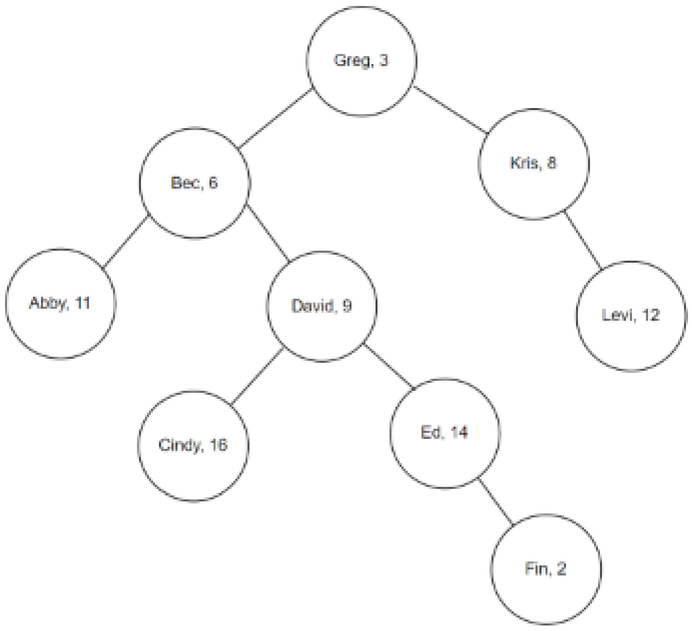
1. we can easily find that for a specific node, all the left children nodes are smaller than the node, all the right children nodes are bigger than the node lexicographically.
2. After removing the names and only consider the student IDs, we can find for a specific node, all the IDs of children nodes of the specific node is bigger than the ID of the specific node.

Part B

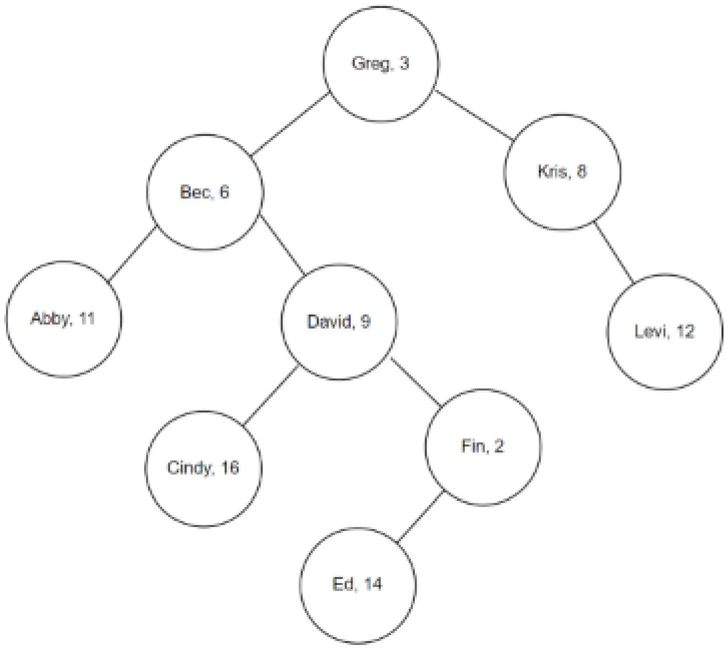
We start with the following QUBSET

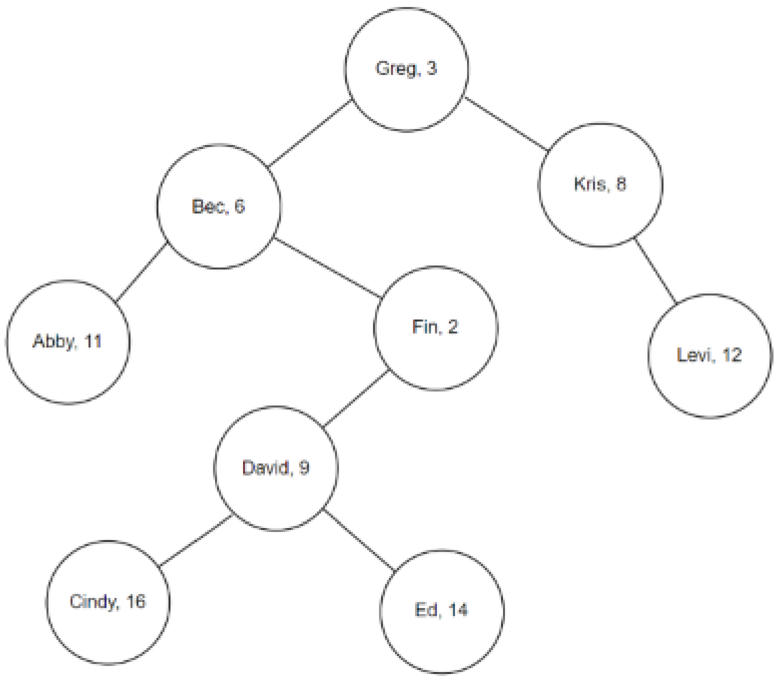


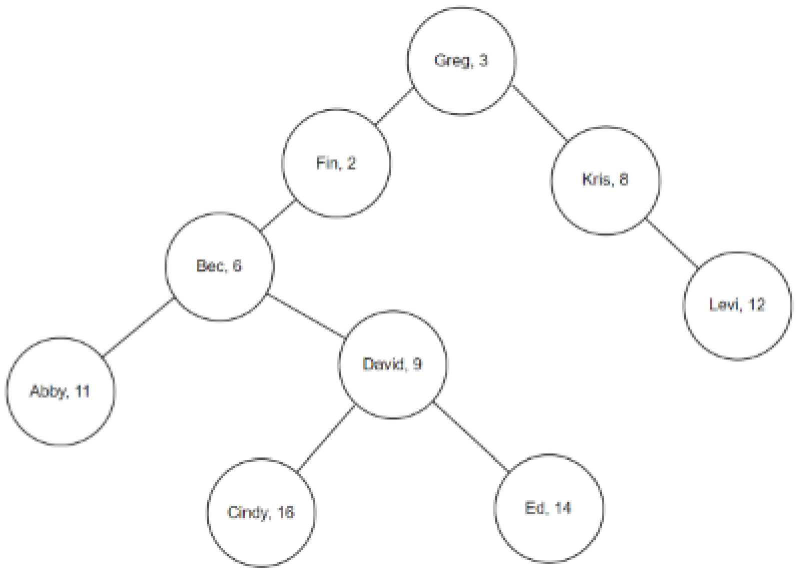
Now suppose we add Fin who has a student ID of 2 to the QUBSET follwing the rule in part a)i.

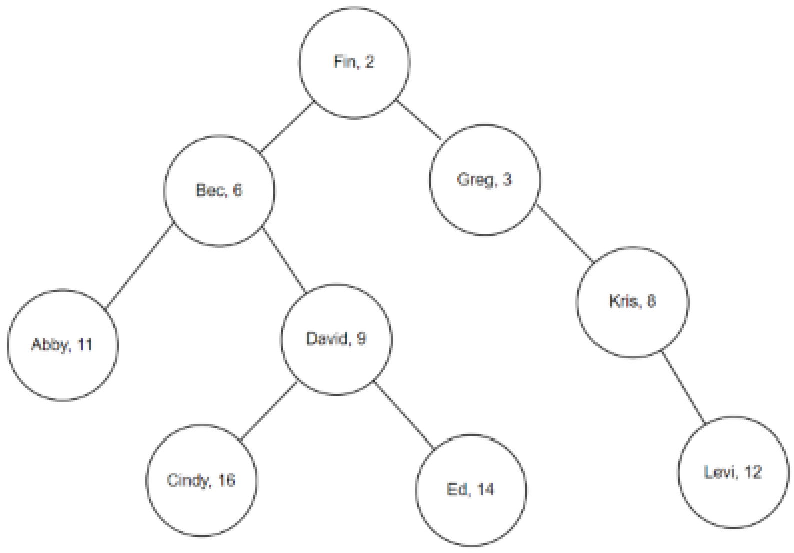


After this, the tree does not follow the rule in part a) ii. We continue to update the diagram until it follows the QUBSET rule.




Greg, 3
Bec, 6
Kris, 8
Abby, 11
Fin, 2
Levi, 12
David, 9
Cindy, 16
Ed, 14


Greg, 3
Fin, 2
Kris, 8
Bec, 6
Levi, 12
Abby, 11
David, 9
Cindy, 16
Ed, 14



Now we satisfy the *QUBSET* rules and stop.

Part C

we can have some assumptions for the function `add_student(T,S)` as follow:

- $T:$ an arbitary QUBSET , S : a new student
- T and S are the same type
- S has no children
- `FATHER(X):` return the parent node of node X
- `LEFTCHILD(X):` return the left child node of node X
- `RIGHTCHILD(X):` return the right child node of node x
- `NAME(X):` return the name of the node X
- `ID(X):` return the student ID of the node X

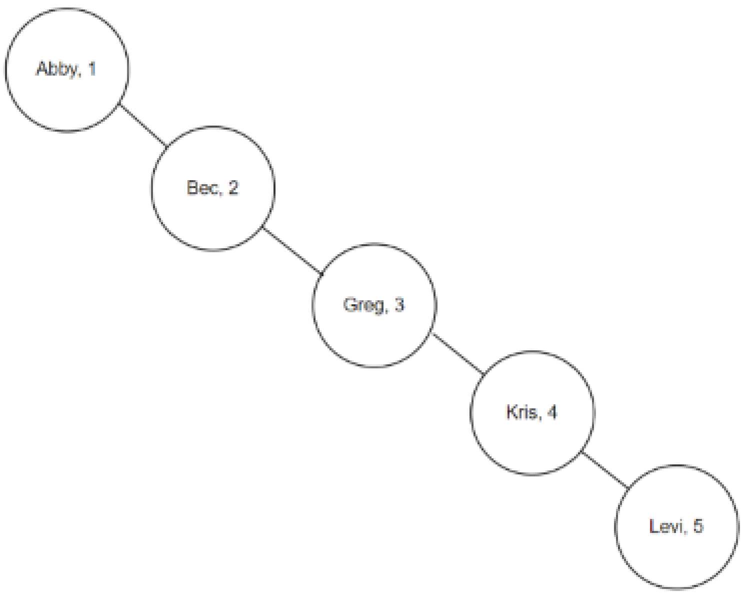
```
function add_student(T, S)
    root <- GETPOS(T, S)
    if LEFTCHILD(root) is empty then
        LEFTCHILD(root) <- S
    else
        RIGHTCHILD(root) <- S
    UPDATETREE(root)

function GETPOS(T, S)
    if NAME(T) < NAME(S) then
        if LEFTCHILD(T) is empty then
            return T
         else
            return GETPOS(LEFTCHILD(T), S)
    else
        if RIGHTCHILD(T) is empty then
            return T
        else
            return GETPOS(RIGHTCHILD(T), S)
```

```
function UPDATETREE(U)
    if ID(U) < ID(FATHER(U)) then
        if FATHER(FATHER(U)) is non-empty then
                if RIGHTCHILD(FATEHR(FATHER(U))) is FATHER(U) then
                    RIGHTCHILD(FATHER(FATHER(U))) <- U
                else
                    LEFTCHILD(FATHER(FATHER(U))) <- U
        if RIGHTCHILD(FATHER(U)) is U then
            tempFa <- FATHER(U)
            RIGHTCHILD(tempFa) <- LEFTCHILD(U)
            FATHER(LEFTCHILD(U)) <- tempFa
            FATHER(U) <- FATHER(tempFa)
            FATHER(tempFa) <- U
            LEFTCHILLD(U) <- tempFa
        else
            tempFa <- FATHER(U)
            LEFTCHILD(tempFa) <- RIGHTCHILD(U)
            FATHER(RIGHTCHILD(U)) <- tempFa
            FATHER(U) <- FATHER(tempFa)
            FATHER(tempFa) <- U
            RIGHTCHILD(U) <- tempFa
        UPDATETREE(U)
    else
        return
```

Part D

the worst case may be like this, the height is 5:



Part E

When we consider a group of n students that are listed in alphabetical order, and add them into the binary search tree, it will represent like the tree in the Part D, a chain whose height is n .Considering add a new node into the binary search tree, the new node will be placed to the bottom right corner, because of the property of binary search tree and the data order.but when we use the *QUBSET* , the student ID will be another rule for ordering. When new nodes are added, the size relationship of the ID causes the left-handed and right-handed adjustment of the tree structure, making the chain have the opportunity to become a tree. considering the limit case, when the values of student ID are totally generated randomly, the height of the tree will tend to $log(n)$, As n increases, the difference in tree height between the two will become larger and larger.

Question 2: Colourful Study Notes

Part C

When we have n words and a total of C colours, the complexity of the iterating over entire sequences of colours is $O(C^n * n^2 * C^2)$ which is an exponential complexity.

Part D

we can let $f(n, c)$ as up to the $n\text{-th}$ word, the largest scores when c is the color of the $n\text{-th}$ word. then we will have the recurrence relaton for $f(n, c)$:

$$f(n, c) = max_{i=0,1,\cdots,m}(f(n-1, c_i) + WC(n, c) + CT(c_i, c), f(n, c))$$

then we will have the score F is :

$$F(n) = max_{i=0,1,\cdots,m}(f(n, c_i))$$

in this way we can solve the score $F(n)$.

Part G

Assume we have N words and a total of C colours. the complexity of the dynamic programming approach in Part D is $O(NC^4 + Nn)$, wich n is the number of the colortables.