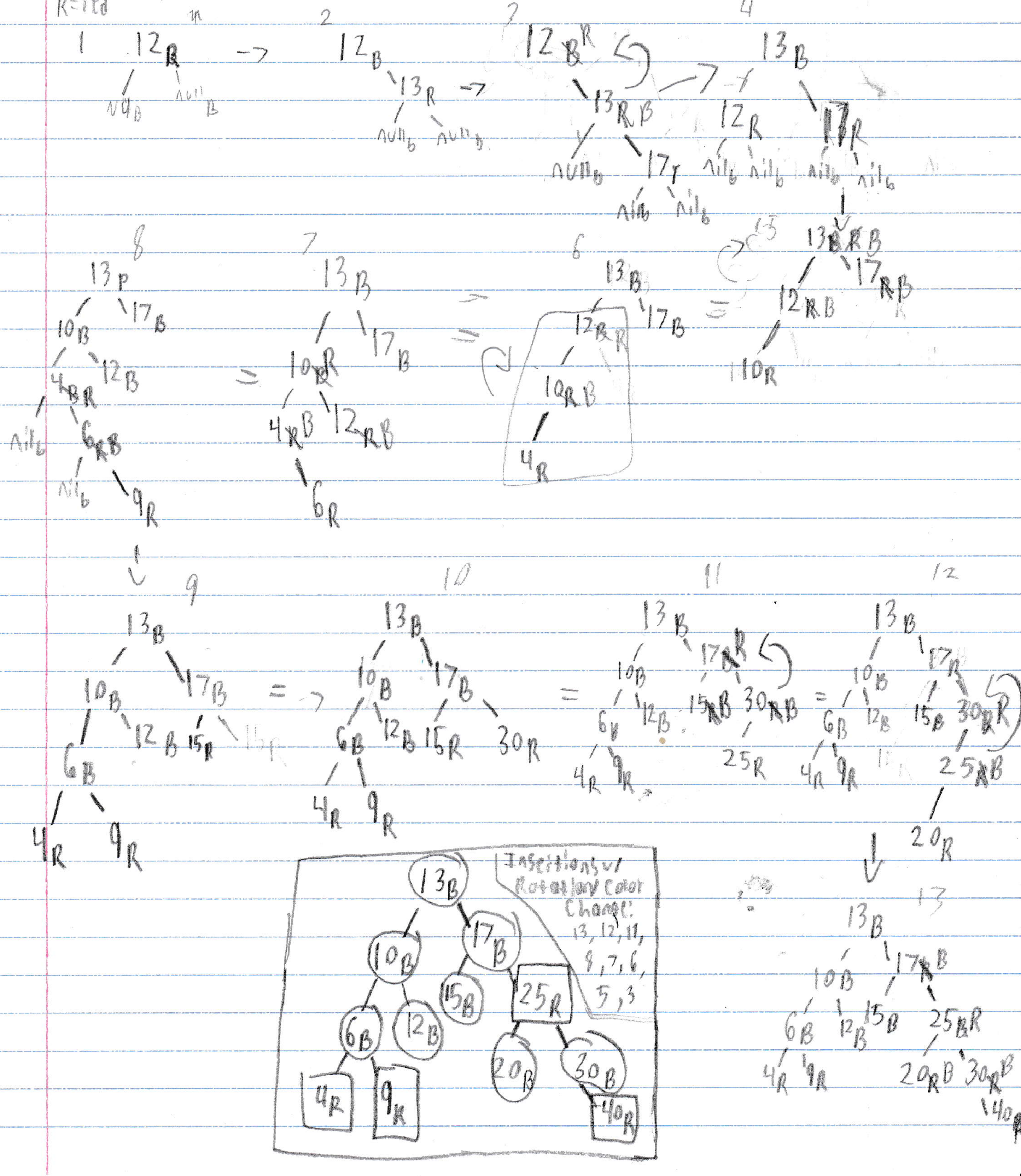
Hunter McMahon

CS 313 Written Assignment 4

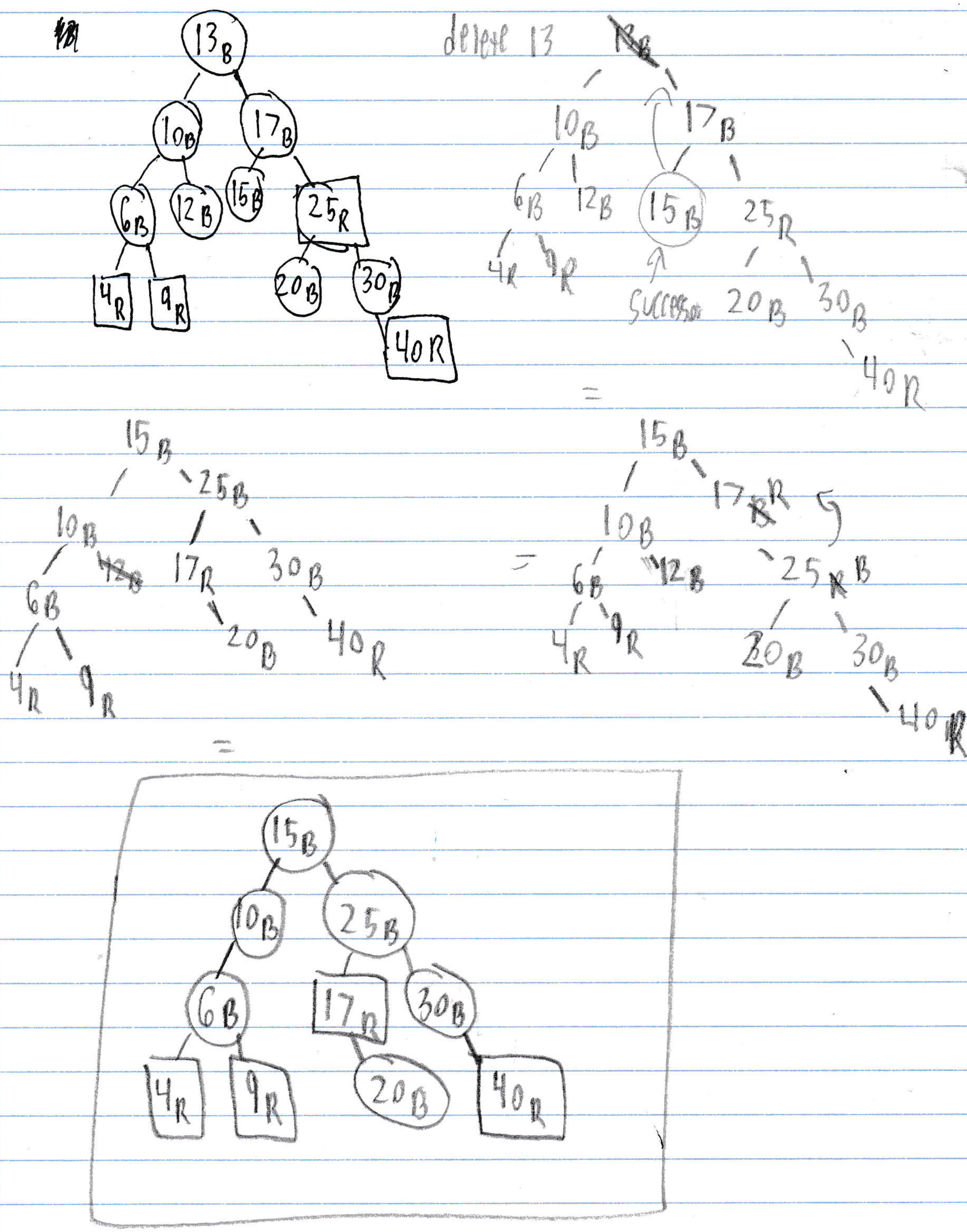
1. insert the following into an initially empty red black-tree, show the tree after each insertion that causes a color-shift or Rotation

B=Black
R=red

12, 13, 17, 10, 4, 6, 9, 15, 30, 25, 20, 40



2. delete 13 and then 12 from the tree in #1.



3. TB 13.3-5 Consider a Red-black tree formed by inserting n-nodes w/ RB-insert
-argue that if $n>1$, the tree has at least one red node

- based on the Logic used for inserting a node, we first insert it as we would in a BST & give it a Red coloring. Then we adjust the tree such that it abides by the Rules of a Red-black tree. in this, we either color shift so that there aren't two reds such that one red is a parent of the other red or we just preform a rotation to keep the tree balanced, Thus the only way an inserted node which is Red by default to become black during the insert Phase is if it becomes the parent of a seperate Red node after Rotation therefore a R/B tree w/ $n>1$ Leaves must have at least 1 Red-node

4. TB 18.1.3 Show all legal b-trees w/ Min. degree 2 that store the keys 1, 2, 3, 4, 5

$t=2$

* each node needs between t & 2t children

Insert order

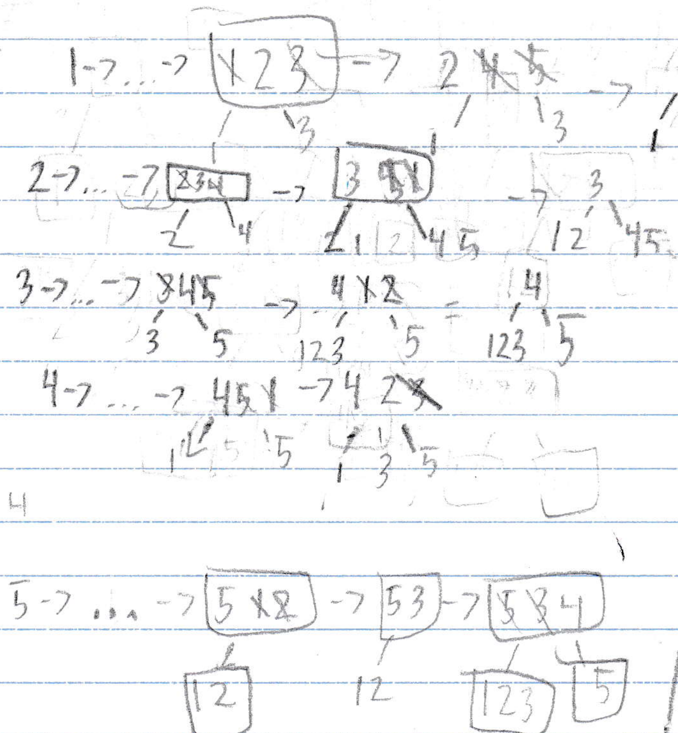
1 | 32 | 34
1 | 45 | 23
1 | 54 | 32
etc. > 1, 2, 3, 4, 5

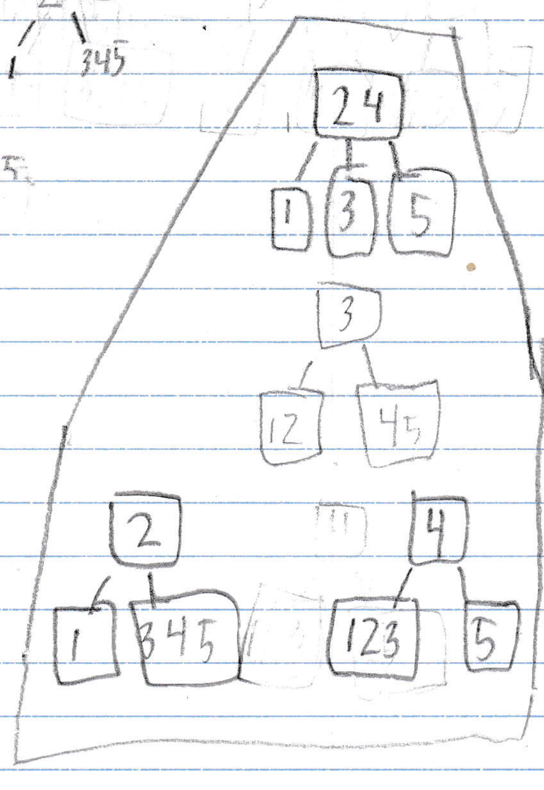
2 | 43 | 13
2 | 51 | 43
2 | 15 | 34 etc. = 2, 3, 4, 5, 1

3 | 54 | 21
3 | 12 | 45
3 | 21 | 54 etc. = 3, 4, 5, 1, 2

4 | 5 | 321
4 | 23 | 51
4 | 32 | 15 | etc. = 4, 5, 1, 2, 3

5 | 34 | 12 = 5, 1, 2, 3, 4
or etc.
5 | 45 | 21
or
5 | 21 | 43
etc.





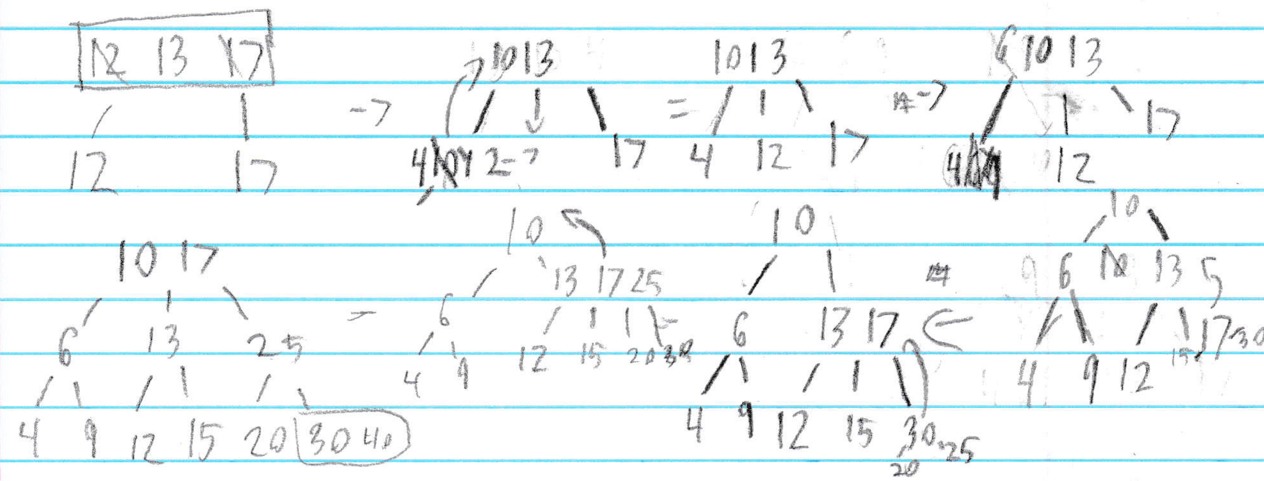
* on insertion order after initial value doesn't matter, resulting b-tree will be the same

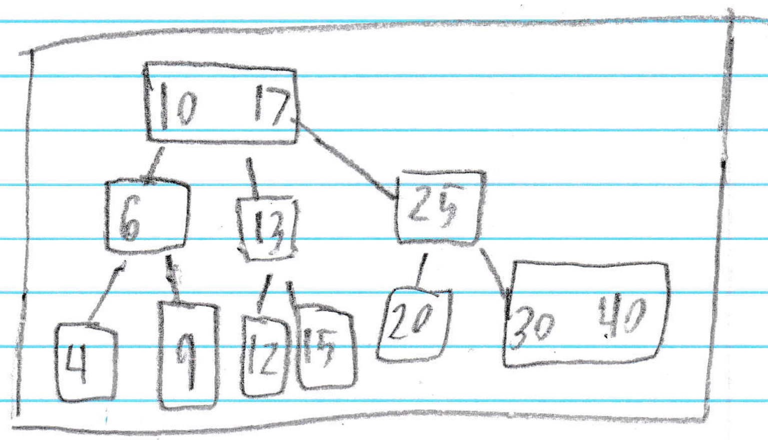
5. insert the values into an initially empty b-tree w/min degree 2 (2-3-4 tree) in the order given:

12 13 17 | 10 4 | 6 9 | 15 30 | 25 20 | 40

Show each intermediate step that causes a split

* between t & 2t children





6. Suppose you have an array ~~of size~~ S that is size n w/ each element in S representing a vote in a class election. Each vote is given as the candidates student I.d ~~number~~ (as an int)

W/O making assumptions about the # of candidates, design an $O(n \lg n)$ Algorithm to determine which candidate recieves the most votes.
- extra credit if $O(n)$

Python dicts use hash tables which have $O(1)$ time complexity, of course this is Psuedo code so we assume that we have a dict object implemented as a hash table

If we have x candidates, want to find the most

```
Vote_Counter(S)
        Candidates = dictionary()
        Max_Votes = 0
        Winner = 0
        for i in S:
            if S[i] in Candidates:
                Candidates[S[i]] += 1
                If Candidates[S[i]] > Max_Votes
                    Max_Votes = Candidates[S[i]]
                    Winner = S[i]
            else:
                Candidates[S[i]] = 0
        return Winner
```

$O(n)$ Runtime

$O(n) < O(n \cdot \lg n)$

$n < n \cdot \lg n$

Still above our Runtime & is thus still an upper bound to it

7. modify the previous problem solution to a situation where we know that # of $K<n$ candidates running, design an $O(n \lg K)$ algorithm to determine the winner candidate w/ the most votes.

my ~~own~~ code from # 6 is a solution to this as well:

```
vote_counter(S)
        candidates = dictionary()
        max_votes = 0
        winner = 0
        for i in S:
                if S[i] in candidates:
                        candidates[S[i]] += 1
                        if candidates[S[i]] > max_votes
                                max_votes = candidates[S[i]]
                                winner = S[i]
                else:
                        candidates[S[i]]
        return winner
```

- Since dictionaries can be implemented w/ hash tables which have $O(1)$ time complexity for search & insert, the only thing to consider in this programs run time is the single loop that stops after going through all n elements thus it runs in $O(n)$ time which is faster than $O(n \lg k)$ thus it still is an upper bound.