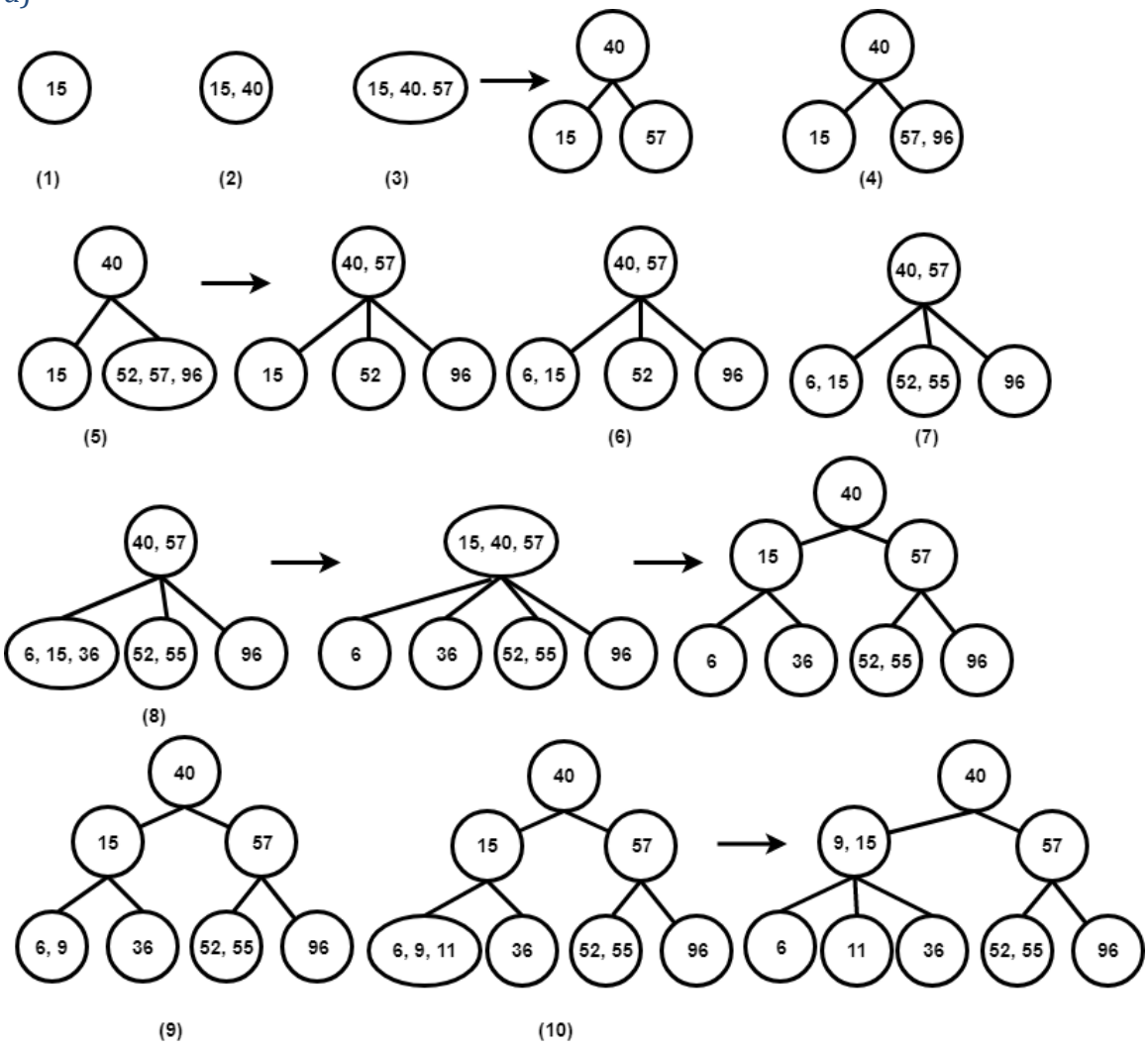
1. (20 Points) Given this array, {15, 40, 57, 96, 52, 6, 55, 36, 9, 11}, draw the sequence of trees by inserting elements in that order into initially empty tree using the following trees. Note a result tree is required after each key insertion.

a) 2-3 Tree.

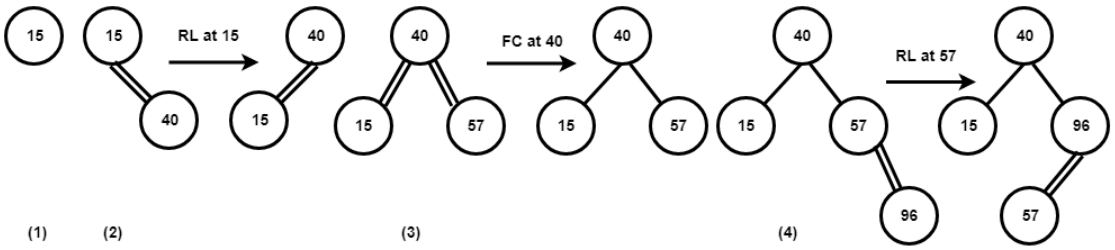
b) Red black BST.

Answer:

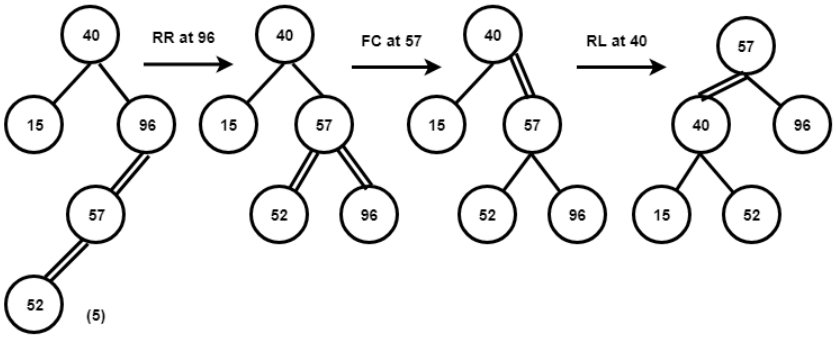
a)



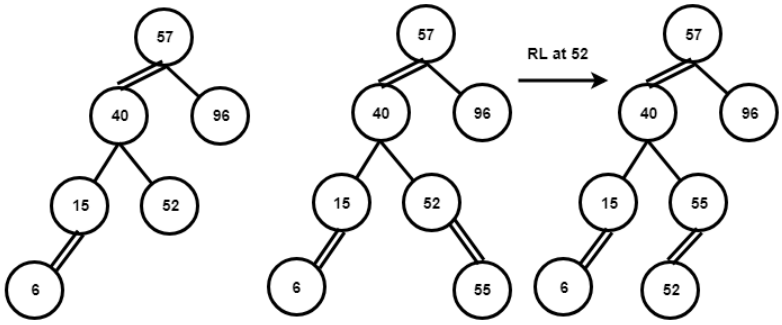
b) next page, double lines represent red link and single line represents black link



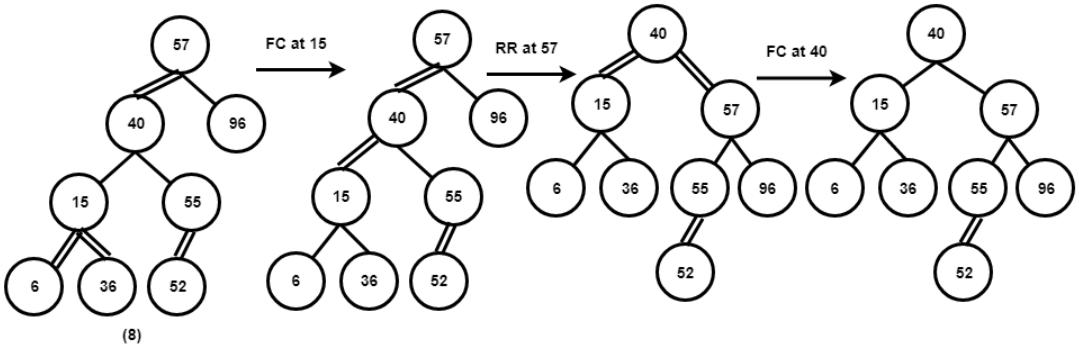
(1) (2) (3) (4)



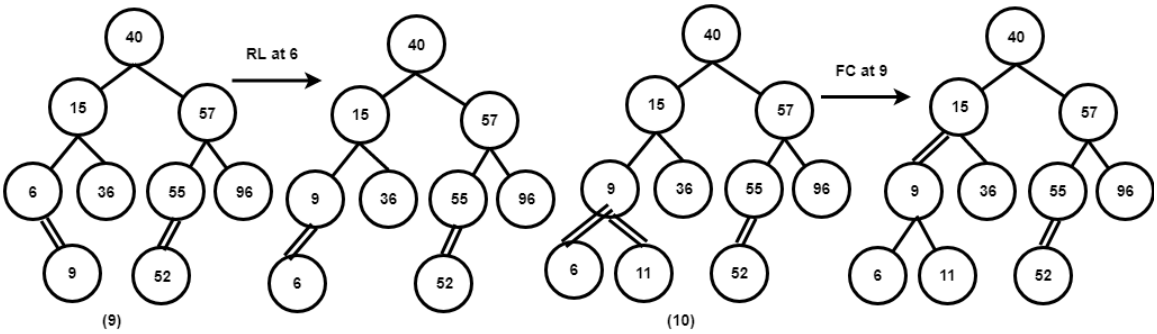
(5)



(6) (7)



(8)



(9) (10)

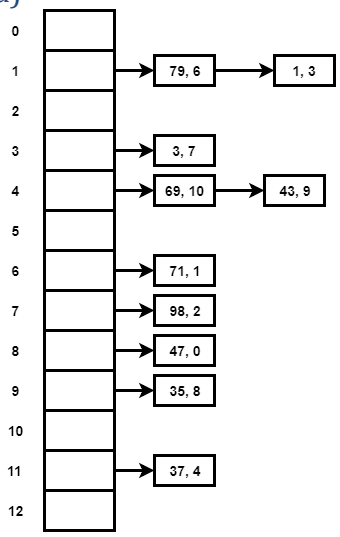
2. (10 Points) Given keys as {47, 71, 98, 1, 37, 43, 79, 3, 35, 43, 69} and their associated values as {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10} respectively. Show the contents of the hash table after inserting those key values pairs into an initially empty hash table with size 13 using the following conflict resolution methods discussed in class:

a) Separate chaining. When there is a search miss, insert new key value pair at the beginning of the linked list.

b) Linear probing **with resizing**. Make sure to follow LinearProbingHashST code presented in class slides.

Answer:

a)



b)

index	0	1	2	3	4	5	6	7	8	9	10	11	12
keys		1			43		71	98	47			37	
vals		3			5		1	2	0			4	

Before Resizing

index	0	1	2	3	4	5	6	7	8	9	10	11	12
keys		1	79	3						35		37	
vals		3	6	7						8		4	
index	13	14	15	16	17	18	19	20	21	22	23	24	25
keys					43	69	71	98	47				
vals					9	10	1	2	0				

After Resizing

3. (10 Points) Give a set with 13 elements, show the final result of executing the following instructions with **UF_WeightedQuickUnion**: union(7, 8), union(9, 10), union(11, 12), union(9, 11), union(0, 1), union(0, 2), union(6, 4), union(3, 5), union(4, 3), union(0, 6). Assuming initially there are 13 components.

a) Show the final contents of id[] array

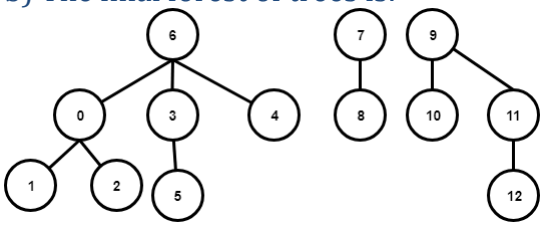
b) Draw the final forest of trees.

Answer:

a) The contents of id array are:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
id	6	0	0	6	6	3	6	7	7	9	9	9	11

b) The final forest of trees is:



4. (5 Points) Mark the following statements true or false.

a) Suppose that keys are t-bit binary integers. For a modular hash function (key % M) with prime M greater than 2, it is always true that if two keys differing only in one bit (such as 1111 and 1101) they would have different hash values.

b) Consider the idea of implementing modular hashing for integer keys with the code (a * key) % M, where a is an arbitrary fixed prime. This change mixes up the bits sufficiently well so nonprime M can be used.

Answer:

a) The statement is correct. Assume it is incorrect, we have n1 and n2 differ just one bit, and n1%M==n2%M, that means (n1-n1)%M==0. Let us assume n1>n2 and has k bits, n1-n2 can have values in this set $\{2^0, 2^1, 2^2, \ldots, 2^{(k-1)}\}$, none of them can have remainder as 0 when is divided by M. That contradicts the assumption.

b) The statement is incorrect. Suppose a is 13 and M is 100, if keys are all divisible by 10, then the hash function can only map keys to these values {0, 10, 20, 30, 40, 50, 60, 70, 80, 90}, 10 out of 100 values.

5. (5 Points) Find 8 strings, each of length 8, that have the same hashCode() value, supposing that the hashCode() implementation for String is the following:

```
public int hashCode()  {
        int hash = 0;
        for (int i = 0; i < length(); i ++)
                hash = (hash * 31) + charAt(i);
        return hash;
}
```

Strong hint: Aa and BB have the same value.

Answer:

There are multiple correct answers, here is one set:

{"BBBBBBBB", "AaBBBBBB", "BAaBBBBB", "BBAaBBBB", "BBBAaBBB", "BBBBAaBB", "BBBBBAaB", "BBBBBBAa"}

6. (10 Points) Modify the following methods in SeparateChainingHashST.java class without changing any of the existing method signatures.

Answer:

```
    // update this method to keep track of n
    public void put(Key key, Value val) {
        if (getLoadFactor() >= 0.75) {
            resize(2*m);
        }
        SequentialSearchST<Key, Value> sst = st[hash(key)];
        int sizeBefore = sst.size();
        st[hash(key)].put(key, val);
        int sizeAfter = sst.size();
        n += (sizeAfter - sizeBefore);
    }

    // update this method to keep track of n
    public void delete(Key key) {
        SequentialSearchST<Key, Value> sst = st[hash(key)];
        int sizeBefore = sst.size();
        st[hash(key)].delete(key);
        int sizeAfter = sst.size();
        n += (sizeAfter - sizeBefore);
    }

    /**
     * @return load factor of the symbol table: (number of key value pairs)/(table size)
     */
    public double getLoadFactor() {
        return (n*1.0/m);
    }

    /**
     * Resize methods doubles table size and rehashes existing key value pair to the new table
     * in the order as they appear in the old table
     */
    public void resize(int capacity) {
        SeparateChainingHashST<Key, Value> newSt = new SeparateChainingHashST<Key, Value>(capacity);
        for (SequentialSearchST<Key, Value> sst: st) {
            for (Key key: sst.keys()) {
                newSt.put(key, sst.get(key));
            }
        }

        this.m = newSt.m;
        this.st = newSt.st;
    }
}
```

Submission Note

1) For written part of the questions:
   a) Write your answers inside a text document (in plain text, MS Word, or PDF format)
   b) Name the file as firstname.lastname.assignment4.txt(doc, docx, or pdf) with proper file extension
2) For programming part of the questions
   a) Use JDK 1.8 and Junit5
   b) Put your full name at the beginning of every source file you created or modified. **2 points will be deducted if your names are not included in the source files.**
   c) Do not change the provided package, class, or method name. You can add extra classes or methods if they are needed.
   d) **If your code does not compile, you will get zero point**.
   e) Use the provided tests to verify your implementation. Extra tests might be used for grading.
   f) Zip all the source files into firstname.lastname.assignment4.zip
3) Submit both of your files (text document and zip file) via Canvas course web site.
4) Due Nov 27th, 11:59 PM