Q2 Jae Hong Lee

Q2.1

Output

1 1 0
2 2 1
2 1 3
5

Q2.2

a)

Str	Ch	X	Y
'reiterate'	't'	9	3
"	"	6	3
'erate'	"	6	3
"	"	3	3
'e'	"	3	-1

b) 3

Q2.3

```
public static int process(String s1, String s2, int n){
    if (s1 == null || s2 == null){
        throw new IllegalArgumentException();
    }else{
        int num = 0;
        String result = "";
        if (s1.length() < n && s2.length() < n){
            result = s1+s2;
        }else if(s1.length() < n){
            result = s1 + s2.substring(s2.length() - n);
        }else if (s2.length() < n){
            result = s1.substring(0, n) + s2;
        }else{
            result = s1.substring(0, n) + s2.substring(s2.length() - n);
        }
```

Q2.3 continue

```
for (int i=0; i <s2.length(); i++){
    if( s1.charAt(i) == s2.charAt(i)){
        num++;
    }
}
System.out.println(result);
return num;
}
```

Q3 Question

Jae Hong Lee

Q3.1

a) b) c)

```
public class price {
    private double dollar;
    private double cent;

    public price (int d, int c ){
        self.dollar = d;
        self.cent = (double) c;
    }
```

Q3.2

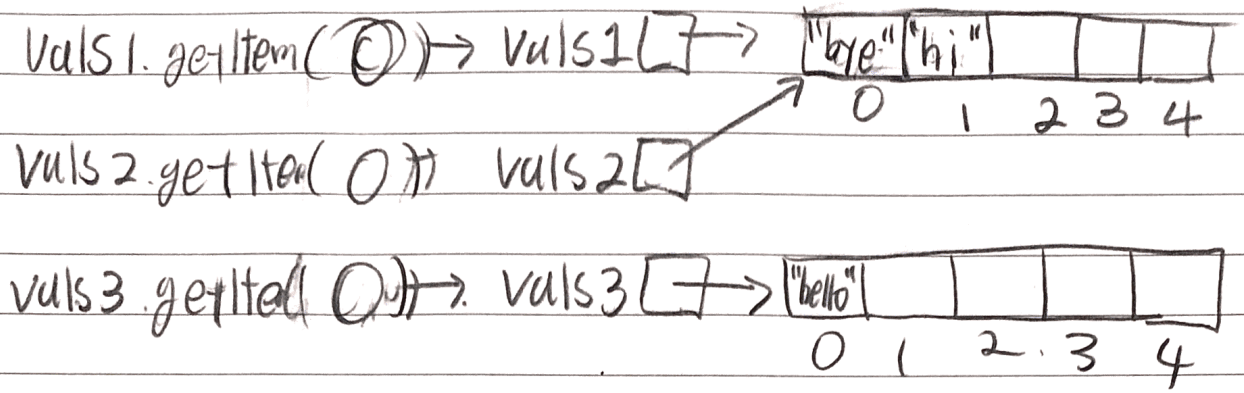
a)
```
public int getCents() {
    return this.cent;
}
```

b)
```
public void increaseBy(int c){
    if (c <0){
        throw new IllegalArgumentException();
    } else{
        self.cent = self.cent + (double)(c/100)
```

Q3.3
```
Price p3 = New price(d, c);
p3 increaseBy(30);
```

Q4 Jue Hong Lee

Q4.1



Q4.2

Output
bye
bye
hello

Since vals1 and vals2 are referencing the same arraylist adding "hi" and "bye" would move "hi" to the next index, so the index of vals1 and vals2 is "bye" and vals3 index 0 is "hello"

Q4.3

a) Arraylist would be better than LLlist when getting item from the list since Arraylist can get the item instantly from the index number.

b) LLlist would be better when the size of the list is unknown and can expand infinitely. So when the adding many arrays in the list

Q5 Jae Hong Lee

Q5.1

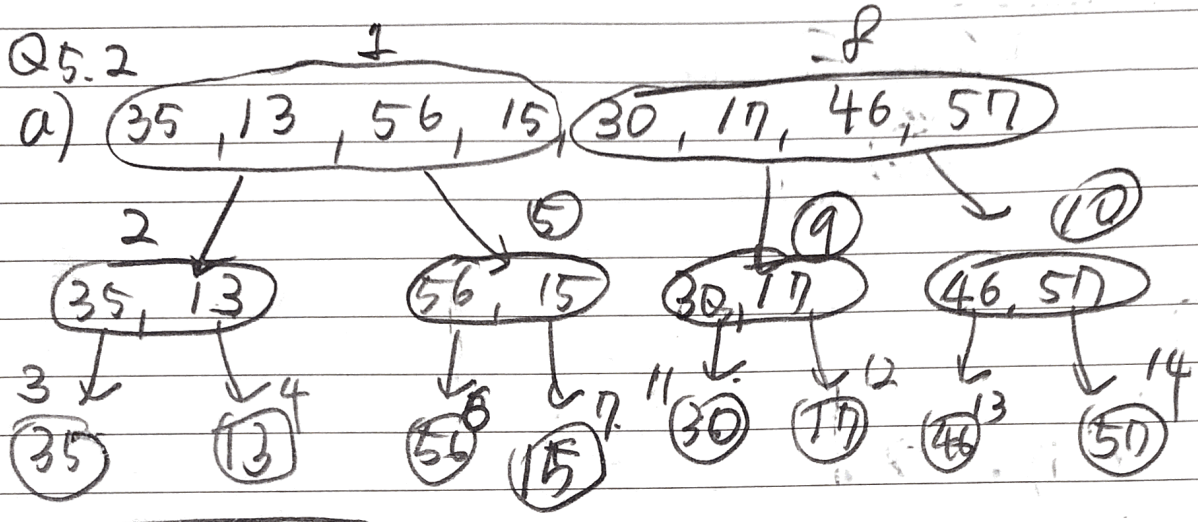
a) 26 20 10 12 42 18 16 48

b) 26 20 10 12 18 16 42 48

c) 20 10 12 18 16 26 42 48

Q5.2

a)

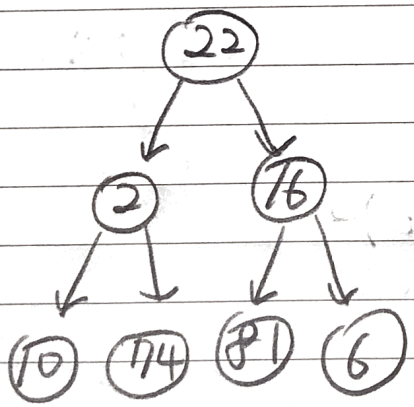


14th called

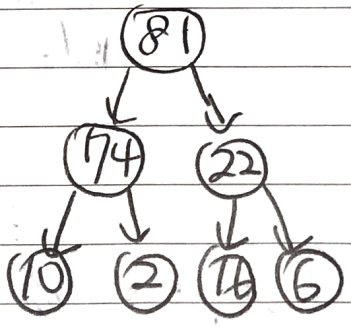
b) 13, 35, 56, 15, 30, 17, 46, 57

Q5.3

a)



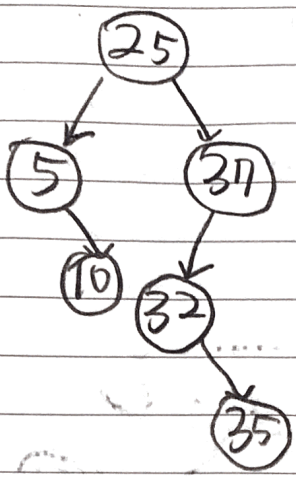
b)



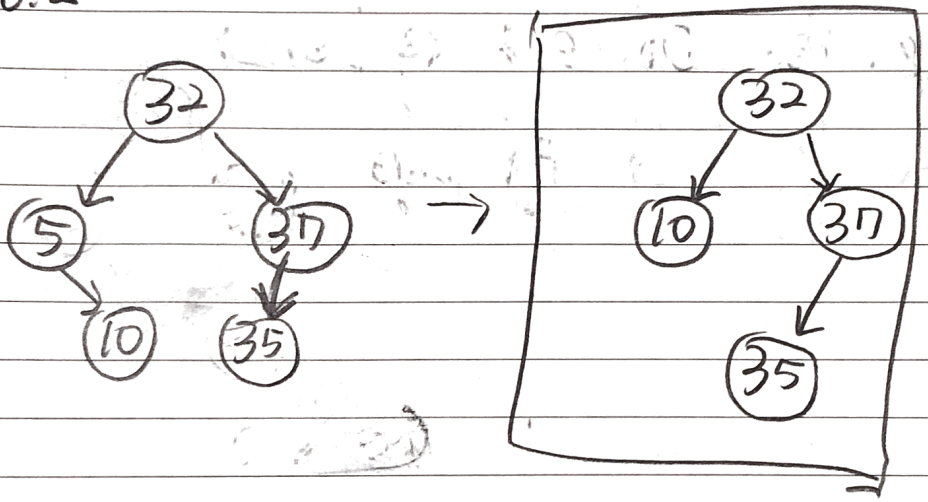
c) [16, 6, 22, 16, 2, 74, 81]

Q6 Jae Hong Lee

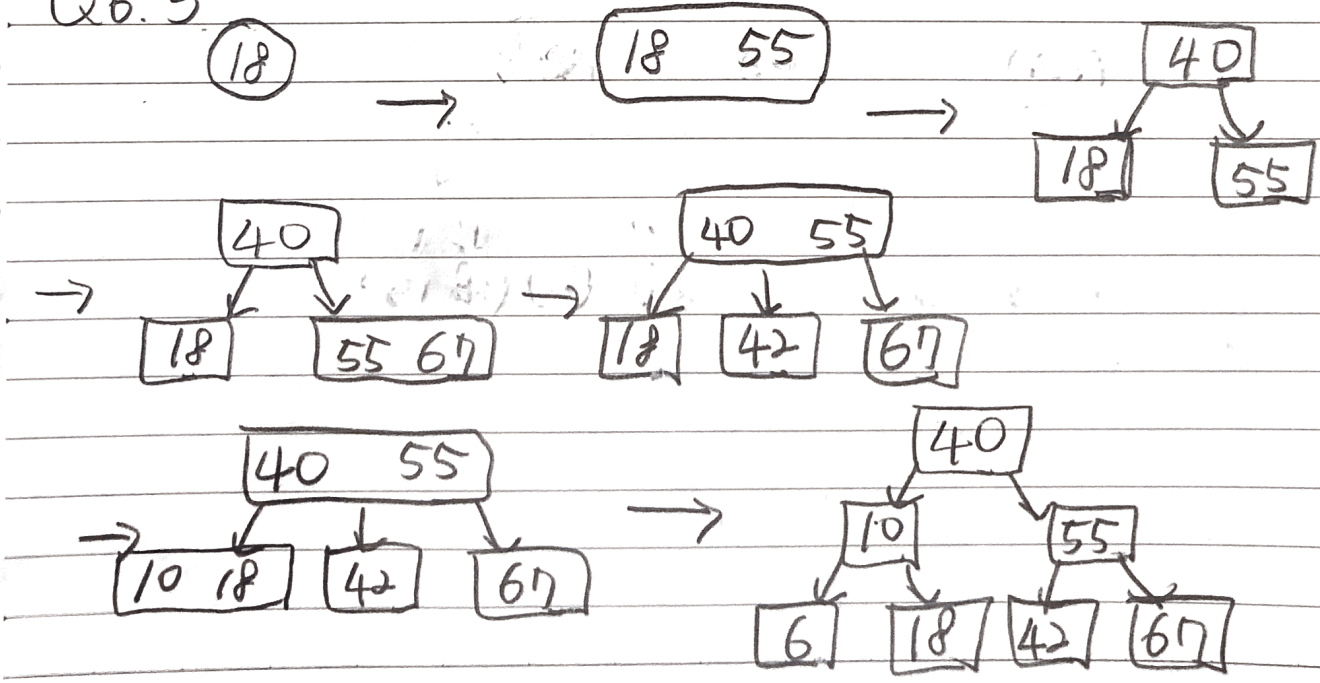
Q6.1



Q6.2

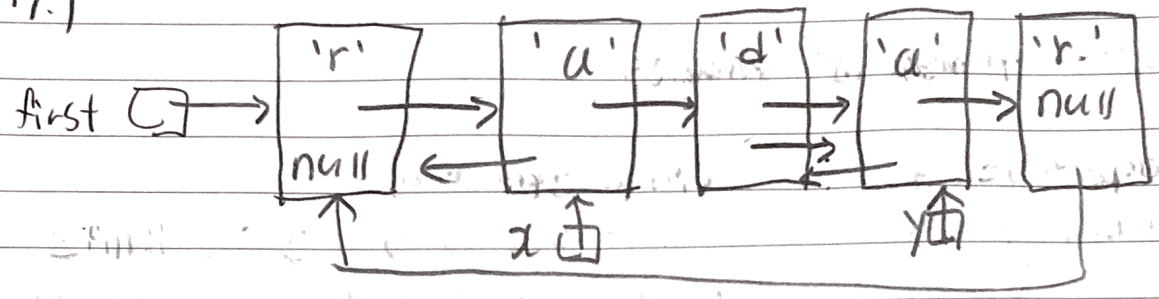


Q6.3



Q 7 Jae Hong Lee

Q7.1



Q7.2

public Dnode process (DNode first, char ch) {

Q8 Jae Hong Lee

Q8.1
h2(key) → number of vowels.

a) h1(elephant) = 8
h2(elephant) = 3

probe sequencing
1. h1 = 8 % 7 = 1 → filled
2. h1+h2 = (8+3) % 7 = 3 → filled
3. h1+2h2 = (8+6) % 7 = 0 → open!

b) yes position 0, from the open addressing probing sequence we reached the position, which is not full and also not deleted.

Q8.2
Quadratic probing

14+3
12 % 7 = (5)

a) h1(elephant) = 8

probe sequencing
1. h1 = 8 % 7 = 1 → filled
2. h1+1 = (9 % 7) = 2 → empty but deleted
3. h1+4 = (8+4 % 7) = 5 → filled
4. h1+9 = (17 % 7) = 3 → filled
5. h1+16 = (24 % 7) = 3 → filled
6. h1+25 = (33 % 7) = 5 → filled
7. h1+36 = (44 % 7) = 2 → empty but deleted

b) Since we couldnot find the empty and not deleted slot for the length times 7 so we place it on the first empty position which is [2]

Q#.3

It is not always possible for quadratic probing as you see from the Q#.2, the quadratic probing can have the same remainder so it wouldn't find the open spot. However using two hash function, double hashing can have a better possibility for finding spot.

Q 9 Jae Hong Lee

Q9.1

Output

{37, 10, 16, 25}

{10, 37}

{10}

{37, 16, 25}

Q9.2

```
public static int process (Stack<Object> S, Object item){
    int num = 0;
    Stack<Object> i = new Stack<Object> ();
    while ( !(S.isEmpty()) ){
        Object k = S.pop();
        if( k.equals(item) ){
            num++;
            i.push(k);
        }
    } while( !(i.isEmpty()) ){
        k.push(i.pop());
    } return num;
}
```

Q9.3

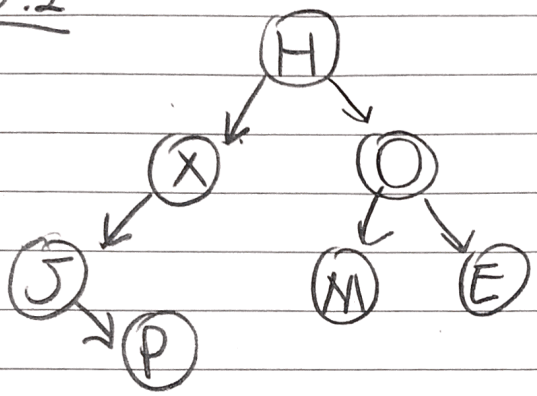
The worst case is $O(n^2)$ since there is two while loop, we need to go over every n for the worst case so the worst case runtime is n^2.

Q 10 Jue Hong Lei

Q10.1

It is Balanced. Since the all subtree's height difference is less or equal to 1. such as the subtree high of 25 is 2 in the left tree and 1 in the right tree so the difference is 1. and also the left subtree of 17's height is 1 the same as right tree's height and the height of subtree 31.

Q10.2



Q10.3

a)
```
public Static int find (Node root){
    Node trav = root;
    int n= 1;
    while( trav. left != null){
        trav= trav. left;
        n += 1;
    } return n;
}
```

b)
```
public Static int find (Node root) {
    Node trav = root;
    if( trav. left == null || trav. right ==null){
        return -1;
    } else {
        int n = 1;
        while (trav. left != null) {
            trav= trav. left; n += 1;
    } return n;
}
```

Q11 Jae Hong Lee

Q11.1

```
public static int process(Node root, int k1, int k2){
    if(root == null || k1 > k2){
        return 0;
    }else{
        int leftTree = processTree(root.left, k1, k2);
        int rightTree = processTree(root.right, k1, k2);
        if(root.key >= k1 && root.key <= k2){
            return leftTree + rightTree + 1;
        }else{
            return leftTree + rightTree;
        }
    }
}
```

Q11.2

best case: O(1): when the two k1 and the k2 values are the root's right next two values then root is the only value of this process method

worst case: O(n): when the k1 and k2 values are first and last values of the tree then processTree method should search down every single node so it will have n runtime.