8. (a) What are the differences between ArrayList and Vector? (10)

Consider the following class:

```
public class Fruit {
    private String name;
    private int quantity;
}
```

Suppose there is an ArrayList of Fruit named listFruits. Write necessary Java code so that when Collections.sort(listFruits) is called, the products are sorted by descending order of quantity and if quantity matches then by lexicographic order of their name.

(b)

8. a)

| ArrayList | Vector |
|---|---|
| Non-synchronized | Synchronized |
| Increases size 50% when capacity excedes | Increases size 100% when capacity exceeds |

8. b)

```
// method 1
class Fruit implements Comparable<Fruit>
{
private String name;
private int quantity;
// constructors, getters, setters
```

```
@Override
public int compareTo (Fruit f)
{
if (this.quantity != f.getQuantity())
  returns  f.getQuantity() - this.quantity;

else return
this.name.compareTo (f.getName());

  }

List <Fruit> listFruits
          = new ArrayList<>();
 // add food;
Collections.sort (listFruits);
```

method 2

```
class cmp implement Comparator<Fruits>
                                    }
 @Override
public void compare(Fruit a, Fruit b)
{ if (a.getQuantity != b.getQuantity()).
  return b.getQuantity() - a.getQuantity
                                    ();
 else
return a.getName().compareTo(b.getNames));
 }
 // Calling sort function
Collection.sort(listFruits, new cmp());
```

method 3 // using Lambda function

```
Comparator<Fruit> cmp
    = (a, b) -> (
        if (a.getQuantity != b.getQuantity())
        return b.getQuantity() - a.getQuantity();
        else
        return a.getName().compareTO(
                b.getName());
    );

Collections.sort(listFruits, cmp);
```

## method 4 (automatic lexicographic compare)

```
Comparator<Fruit> cmp
    = (a, b) -> (b - a);

Collections.sort(listFruits, cmp);
```

**2(b).** Consider the following class. (20)

```
public class Person {
   private int id;
   private String name;
   private int age;

   Person (int id, String name, int age) {
      this.id = id;
      this.name = name;
      this.age = age;
   }

   public int getId () {
      return id;
   }
   public String getName () {
      return name;
   }
   public int getAge () {
      return age;
   }

}
```

iii. Sort the list based on the ascending order of the age. If the age is the same, then sort based on ascending order of the name. You can't use your own sorting technique. However, you can change the Person class if necessary.

```
public Class Person implements Comparable
                                <Person>{
// copy pase constructors, getters, setters ;
@Override
public int compareTo (Person p) {
if (this.age != p.getAge())
   return this.age - p.getAge() ;
else
  return this.name.compareTo. (p.getName())
}
```

**8(c).** Consider the following class: (10)

implements Comparable<Product>

```
class Product {
        private String name;
        private double price;

        Product (String name, double price) {
                this.name = name;
                this.price = price;
        }
        public String getName () {
                return this.name;
        }

        public double getPrice () {
                return this.price;
        }
}
```

Write Java code for the following:

i. Define an ArrayList named myProducts that can store a list of Product.
ii. Generate 4 random Product with names 'A' to 'D' and random price and add them to myProducts.
iii. Sort myProducts based on Product's name in ascending order. You can't use your own sorting techniques. You can change the Product class if necessary.

i) List<Product> myProducts
= new ArrayList<>();

ii) Random r = new Random();

myProduds.add
(new Product("A", r.nextDouble());

myProduds.add
(new Product("B", r.nextDouble());

myProduds.add
(new Product("C", r.nextDouble());

myProduds.add
(new Product("D", r.nextDouble());

iii) @Override

```
public int compareTo(Product p) {
return this.name.compareTo(p.getName());
}
```