Write a recursive method named `clear` that removes all elements from a Queue. Do NOT use any loops. You may assume that the instance of Queue implements the following interface and that the methods are as have been discussed in class.

```
public interface QueueInterface<E> {
    public boolean empty();
    public E offer(E e);
    public E poll();
    public int size();
    public E peek();
}
```

The `clear` method should be generic and should accept an object that implements QueueInterface as it only input argument. (clear is NOT to be written within the class that implements Queue.)For grins, write and iterative (using loops) version and a recursive version of this method.

In case you are struggling with syntax, here method signature that will work

```
public <Z> void clearQueue(QueueInterface<Z> q)
```

```
    public <Z> void clearQueue(QueueInterface<Z> q) {
        if (q.empty())
            return;
        q.poll();
        clearQueue(q);
    }
```

Problem 2: Suppose that you are given a class ShortStack that has only the methods

```
    public class ShortStack<V> {
        public V pop();
        public void push(V v);
   }
```

These two methods follow the documentation of Stack discussed in class.

You are then asked to implement a method with the following documentation and signature.

```
    /**
     * Determine the number of items contained in
     * the provided instance of ShortStack.
     * This method may modify the provided instance while
     * it is running.  However, when the method
     * is complete the provided stack will have exactly the same
     * contents in exactly the same order
     * as prior to the execution of the method.
     * @param sStack the stack whose count is to be determined.
     * @return the number of items in the provided stack
     */
    public int ssSize(ShortStack<E> sStack) {return -1;}
```

Provide the implementation of this method. You may use any additional data structures you would find useful.
While you are at it, implement the class ShortStack using a backing array.

```
    public <W> int ssCounter(ShortStack<W> ss) {
        ShortStack<W> tss = new ShortStack<W>();
        W w = null;
        int count = 0;
        while (true) {
            w = ss.pop();
            if (w == null)
                break;
            count++;
            tss.push(w);
        }
        while (true) {
            w = tss.pop();
            if (w == null)
                break;
            ss.push(w);
        }
        return count;
    }
```

Problem 3: You are given the following array .
Show the sequence of recursive function calls for the searchUtil function discussed in class when searching for 34 and 4182

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

```
searchUtil(34, 0, 19, 0)
                    mid pos=9, val=55
   searchUtil(34, 0, 8, 1)
                    mid pos=4, val=5
       searchUtil(34, 5,8, 2)
                    mid=6  val=13
          searchUtil(34, 7,8,3)
                    mid=7, val=21
               searchUtil(34,8,8,4)
                              FOUND!!!

searchUtil(4182,0,19,0)
                    mid=9, val=55
   searchUtil(4182, 10, 19, 1)
                    mid=14, val=610
      searchUtil(4182, 15, 19, 2)
                    mid=17 val=2584
          searchUtil(4182, 18, 19, 3)
                    mid=18, val=4181
             searchUtil(4182, 19, 19, 4)
                    mid=19 val=6765
                searchUtil(4182, 19, 18, 5)
                    FAIL
```

Problem 4. You are given an array list. Write a recursive method that returns a new array list such that the returned array list is reversed order from the provided array list. Your method should leave the original array list unchanged. Hint, a private recursive function will be very helpful. NO LOOPS

```
private <X> ArrayList<X> reverserUtil(ArrayList<X> source,
ArrayList<X> target, int pos) {
        if (pos<=0)
            return target;
        pos--;
        target.add(source.get(pos));
        return reverserUtil(source, target, pos);
    }

    public <Y> ArrayList<Y> reverser(ArrayList<Y> source) {
        return reverserUtil(source, new ArrayList<Y>(),
source.size());
    }
```

problem 5: You are given an array of items that implement the comparable interface. Write a recursive method that returns the location of smallest element in the array.

```
public <V extends Comparable<V>> int smallest(V[] arr) {
        int ret = 0;
        for (int i = 1; i < arr.length; i++) {
            if (arr[i].compareTo(arr[ret]) < 0) {
                ret = i;
            }
        }
        return ret;
    }
```

problem 6. What does this pair of functions do? Use examples in your explanation.
For these functions, create an example with an array of at least 4 items. Show every function call; describe the result of each function call.

```
    private <T> void sUtil(T[] src, T[] tgt, int srcIdx, int
tgtIdx, int smallIdx) {
        if (srcIdx>= src.length)
            return;
        if (srcIdx == smallIdx) {
            sUtil(src, tgt, srcIdx + 1, tgtIdx, smallIdx);
            return;
        }
        tgt[tgtIdx] = src[srcIdx];
        sUtil(src, tgt, srcIdx + 1, tgtIdx + 1, smallIdx);
    }
    public <U> U[] r(U[] arr, int rIndex) {
        U[] rtn = (U[]) new Comparable[arr.length-1];
        sUtil(arr, rtn, 0, 0, rIndex);
        return rtn;

    }
```

The function r takes an array (of some generic type) and an index within the array (It really should check to confirm the containment, but it does not. It then calls sUtil passing it that array and another array one shorter. sUtil, makes of copy of the source array, with the item at rIndex (the location passed into the function r, removed. sUtil might be better named “shortenUtil” and r might be better name “remove” or “shorten”. For example, if the passed in array was of length 3, and the index to be removed was 0, then the recursive calls would look be

```
sUtil(origArray, shorterArray, 0,0,1)
   sUtil(origArray, shorterArray, 1,1,1)
      sUtil(origArray, shorterArray, 2,1,1)
         sUtil(origArray, shorterArray, 3,2,1)
              STOP
```

problem 7: You are given a 3-d array of integers. Each of the dimensions of the array is the same size. (it is a cube). Write a method to compute the sum of the positive items in the array. What is the computational complexity of this method?

```
int sum3d(int[][][] arr) {
int rtn=0;
for (int i=0; i<arr.length; i++)
   for(int j=0; j<arr[i].length; j++)
       for(int k=0; k<arr[i][j].length; k++)
           rtn += arr[i][j][k];
return rtn;
}
```

The function will run in O(N^3) time