

```

class Arrays
{
    // this class solves some problems P9.5, P9.6, P9.10
    // in the textbook

    // this method accepts two arrays, a and b.
    // it creates a new array and fills it by
    // alternately interleaving the elements of the
    // a and b arrays
    public static int[] merge( int[] a, int[] b )
    {
        int size = a.length + b.length;
        int[] c = new int[size];

        int ai=0, bi=0, ci=0; // indices into a,b,c arrays

        // use a loop to fill the array c[]
        while( ci < c.length )
        {
            // if at least one element remains in a[]
            if( ai < a.length ) {
                c[ci] = a[ai]; // copy it to c[]
                ci++; ai++; // advance ci,ai
            }

            // if at least one element remains in b[]
            if( bi < b.length ) {
                c[ci] = b[bi]; // copy it to c[]
                ci++; bi++; // advance ci,bi
            }
        }

        return c;
    }

    // this method accepts two sorted arrays, a and b.
    // it creates a new array and fills it by
    // interleaving the elements of the a and b
    // arrays, but keeping them in sorted order
    public static int[] mergeSorted( int[] a, int[] b )
    {
        int size = a.length + b.length;
        int[] c = new int[size];

        int ai=0, bi=0, ci=0;
        for( ci=0; ci < c.length; ) {
            // while there are elements in a[ai] < b[bi]
            // or b[] is now empty, copy them into c[]
            while( ai < a.length &&
                (bi >= b.length || a[ai] <= b[bi]) )
            {
                c[ci] = a[ai];
                ci++; ai++;
            }

            // while there are elements in b[bi] < a[ai],
            // or a[] is now empty, copy them into c[]
            while( bi < b.length &&
                (ai >= a.length || b[bi] <= a[ai]) )
            {
                c[ci] = b[bi];
                ci++; bi++;
            }
        }

        return c;
    }
}

```

```

// this method looks for a unique in the array
// from a[0] up to a[lastElem-1]. it compares
// against a[lastElem].
public static boolean isUniqueElem( int[] a, int lastElem )
{
    for( int i=0; i < lastElem; i++ ) {
        if( a[i] == a[lastElem] )
            return false; // found match, not unique
    }
    return true; // didn't find any matches, is unique
}

// this method takes an array and makes a new copy,
// but it removes duplicate entries in the copy.
// the copy may be of smaller length
public static int[] removeDuplicates( int[] a )
{
    int maxsize = a.length;
    int[] b = new int[maxsize];
    int bi = 0;
    for( int ai=0; ai < a.length; ai++ ) {
        // find unique elements, put them in b[]
        if( isUniqueElem( a, ai ) ) {
            b[bi] = a[ai];
            bi++;
        }
    }

    // now bi tells us how many elements in b[]
    // are actually used. make a new array of
    // the proper size, fill it, and return it.
    int[] c = new int[bi];
    for( int i=0; i < bi; i++ )
        c[i] = b[i];
    return c;
}

// this method prints the contents of an array
public static void printArray( int[] a )
{
    for( int i=0; i < a.length; i++ ) {
        System.out.print( a[i] + " " );
    }
    System.out.println(); // go to next line
}

public static void main( String[] args )
{
    // create 2 arrays and test the merge() method
    // then test the removeDuplicates() method on
    // the merged array
    int[] x = { 1, 4, 9, 16 };
    int[] y = { 9, 7, 4, 9, 11 };
    int[] z = merge( x, y );
    System.out.println( "Testing merge()" );
    printArray( x );
    printArray( y );
    printArray( z );
}

```

```
System.out.println( "Testing removeDuplicates()" );
z = removeDuplicates( z );
printArray( z );

// ////////////////////////////////////////
// create 2 sorted arrays and test the mergeSorted() method
// then test the removeDuplicates() method on
// the merged array

int[] p = { 1, 4, 9, 16 };
int[] q = { 4, 7, 9, 9, 11 };
int[] r = mergeSorted( p, q );
System.out.println( "\nTesting mergeSorted()" );
printArray( p );
printArray( q );
printArray( r );

System.out.println( "Testing removeDuplicates()" );
r = removeDuplicates( r );
printArray( r );

}

}
```

```

class Find
{
    // this class solves problem P5.24 in the textbook
    // in two ways: one with recursion, one without.

    // this method looks for the string 't' inside
    // the supposedly larger string 's'
    public static int find( String s, String t )
    {
        int si=0, ti=0;
        while( si <= s.length() - t.length() )
        {
            // look inside String s, at position si
            String partOfS = s.substring( si, si+t.length() );
            // start ^^, ^^^^^^^^^^^^^^^ pastEnd

            if( partOfS.equals( t ) ) // compare strings
                return si; // found at position si
            si++;
        }
        return -1; // not found
    }

    // this method looks for the string 't' inside
    // the supposedly larger string 's'
    public static int findRecursive( String s, String t )
    {
        if( s.length() < t.length() )
            return -1; // base case, s is too small; not found

        // look at first part of S
        String firstPartOfS = s.substring( 0, t.length() );

        // check if String s starts off with String t
        if( firstPartOfS.equals( t ) ) // compare strings
            return 0; // found at position 0 of s
        else {
            // Recurse by removing first character from s,
            // and looking for t in the remainder of s.
            // If we found it, we must find the position in
            // s by counting as the recursion unrolls.
            int found = findRecursive( s.substring(1), t );
            if( found < 0 )
                return -1; // didn't find it, keep -1
            else
                return 1+found; // count the positions
        }
    }

    public static void main( String[] args )
    {
        // use 4 simple test cases, more should be used.
        // first case: returns 1, the first match (multiple matches)
        // second case: returns 0, matching at beginning of s
        // third case: returns 9, matching at end of s
        // fourth case: returns -1, no match

        System.out.println( find( "Mississippi", "is" ) );
        System.out.println( find( "Mississippi", "Miss" ) );
        System.out.println( find( "Mississippi", "pi" ) );
        System.out.println( find( "Mississippi", "hip" ) );

        System.out.println( findRecursive( "Mississippi", "is" ) );
        System.out.println( findRecursive( "Mississippi", "Miss" ) );
        System.out.println( findRecursive( "Mississippi", "pi" ) );
        System.out.println( findRecursive( "Mississippi", "hip" ) );
    }
}

```