A)

1.定义一个方法 getLowerCaseWords()从文本中解析出单词，并转为小写。

```
public  static List<String >  getLowerCaseWords(File file) {
    Scanner scanner = null;
    Pattern pattern = Pattern.compile("[a-zA-Z]+");
    String text = "";
    List<String > words = new ArrayList<>();
    try {
        scanner = new Scanner(file);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    if(scanner!=null){
        while(scanner.hasNextLine()){
            text+= " "+scanner.nextLine();
        }
        scanner.close();
    }

    Matcher matcher = pattern.matcher(text);
    while (matcher.find()){
        words.add(matcher.group().toLowerCase());
    }
    return words;
}
```

2.定义一个方法，从一个 list 里面找到包含另一个 list 的元素

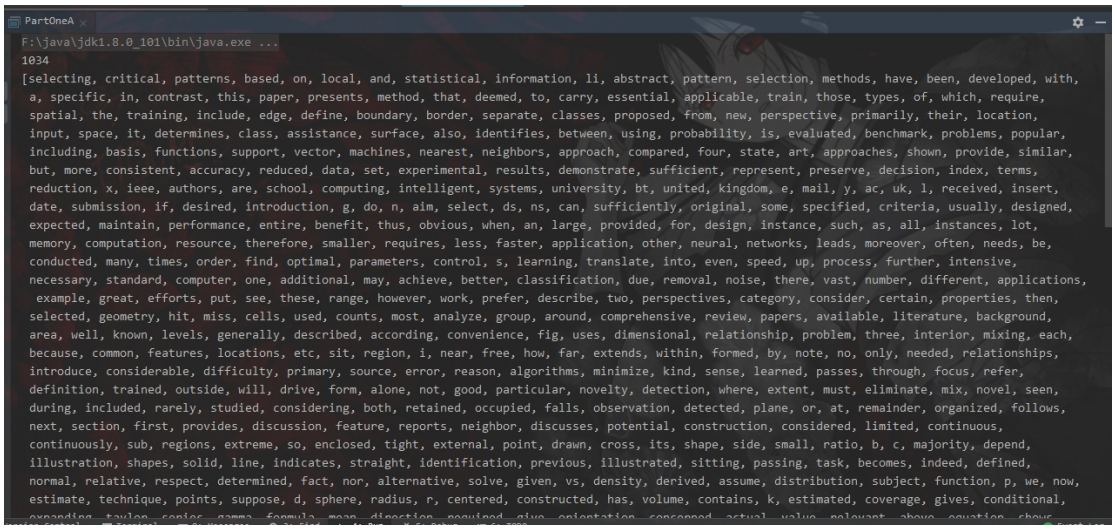
```
public static List<String> getMatchedWords(List<String>
source ,List<String> target){
    return
source.stream().filter(x->target.contains(x)).distinct().collect(Coll
ectors.toList());
}
```

3.写一个测试方法

```
public static void main(String[] args) {
    // read an_article.txt
    File articleFile = new
File("E:\\workspace\\thread\\src\\main\\resources\\an_article.txt");
    // read google-10000-english-no-swears.txt
    File googleFile = new
File("E:\\workspace\\thread\\src\\main\\resources\\google-10000-
```

```
english-no-swears.txt");
    // get words from an_article.txt
    List<String> articleWords=getLowerCaseWords(articleFile);
    // get words from google-10000-english-no-swears.txt
    List<String> googleWords=getLowerCaseWords(googleFile);
    // match with google-10000-english-no-swears, get valid words
    List<String> result = getMatchedWords(articleWords, googleWords);
    // print result
    System.out.println(result.size());
    System.out.println(result.toString());
}
```

运行结果如图所示：



B).

1.定义一个 count，用来统计移动次数

2.定义一个 merge 方法，将最小分组的组内进行比较，并移动。

```
public static void merge(String[] strs, int low, int mid, int high) {
    count++;
    String[] temp = new String[high - low + 1];
    int i = low;// left point
    int j = mid + 1;// right point
    int k = 0;

    // Move the smaller element into the new array first
    while (i <= mid && j <= high) {
        if (strs[i].compareTo(strs[j]) <= 0) {
            temp[k++] = strs[i++];
        } else {
            temp[k++] = strs[j++];
        }
```

```
    }

    // Move the rest of the elements on the left into the array
    while (i <= mid) {
        temp[k++] = strs[i++];
    }

    // Move the rest of the elements on the right into the array
    while (j <= high) {
        temp[k++] = strs[j++];
    }

    // Overwrite the elements of the new array over the strs array
    for (int k2 = 0; k2 < temp.length; k2++) {
        strs[k2 + low] = temp[k2];
    }
}
```

3. 定义一个 mergeSort 方法，使用递归，将待排序的数组分隔成最小的组，再调用 merge 方法，返回排好序的数组

```
public static String[] mergeSort(String[] strs, int low, int high) {

    int mid = (low + high) / 2;
    if (low < high) {
        // left
        mergeSort(strs, low, mid);
        // right
        mergeSort(strs, mid + 1, high);
        // merge left and right
        merge(strs, low, mid, high);
    }

    return strs;
}
```

最后写驱动方法运行截图如下