一、实验目的

1. 了解分治法思想；
2. 掌握递归算法的思想与程序编写；
3. 熟练使用二分查找法实现代码编写。
4. 熟练使用合并排序、快速排序实现代码编写

二、实验内容

1. n个数的全排列问题。
2. 给定由 n 个整数（可能为负整数）组成的序列，求解其连续的最大字段和。当所有数都是负整数时，最大字段和是 0 .
   如：a[] = {-2, 11, -4, 13, -5, -2}时， max = 11 + (-4) + 13 = 20. 。
3. 给定一个随机数数组，求取这个数组中的逆序对总个数。要求时间效率尽可能高。
4. 设b[0:n-1]为数组，数组中含有n个数，参照课本2.7，试设计一个消去递归的合并排序算法。

三、实验总结（写出本次实验的收获，遇到的问题等）

全排列

```java
public class Permutation {
    static int k = 0;

    public static void main(String[] args) {
        int a[] = { 1, 2, 3, 4, 5 };
        permutations(a, 0, 4);
    }

    public static void permutations(int[] a, int m, int n) {
        if (m == n) {
            k++;
            System.out.print(k + "个:");
            for (int i = 0; i <= n; i++) {
                System.out.print(a[i]);
            }
            System.out.println();
        } else {
            for (int i = m; i <= n; i++) {
                int temp = a[m];
                a[m] = a[i];
                a[i] = temp;
                permutations(a, m + 1, n);
            }
        }
    }
}
```

最大字段和

```
//最大子段
public class Maxsize {
    public static void main(String[] args) {
        int arr[] = { -20, 11, -4, 13, -5, -2 };
        System.out.println(maxsize(arr, 0, arr.length - 1));

    }

    public static int maxsize(int[] arr, int left, int right) {
        int sum = 0, midSum = 0, leftSum = 0, rightSum = 0;
        int center, s1, s2, lefts, rights;
        // 如果序列长度为1时
        if (left == right) {
            sum = arr[left];
        } else {
            // 划分
            center = (left + right) / 2;
            // 左递归
            leftSum = maxsize(arr, left, center);
            // 又递归
            rightSum = maxsize(arr, center + 1, right);

            s1 = 0;
            lefts = 0;
            for (int i = center; i >= left; i--) {
                lefts += arr[i];
                if (lefts > s1) {
                    s1 = lefts;
                }
            }

            s2 = 0;
            rights = 0;
            for (int j = center + 1; j <= right; j++) {
                rights += arr[j];
                if (rights > s2) {
                    s2 = rights;
                }
            }

            midSum = s1 + s2;
            if (midSum < leftSum) {
                sum = leftSum;
            } else {
                sum = midSum;
            }
            if (sum < rightSum) {
                sum = rightSum;
            }

        }
        return sum;
    }
}
```

逆序对

```
public class InversePairs {
    public static void main(String[] args) {
        System.out.println(kInversePairs(3, 1));

    }

    public static int kInversePairs(int n, int k) {
        if (k > n * (n - 1) / 2) // n numbers can generate at most n * (n - 1) / 2 inverse
pairs
            return 0;

        if (k == n * (n - 1) / 2 || k == 0)
            return 1;

        int mod = 1000000007;
        int[][] dp = new int[n + 1][k + 1];

        for (int i = 1; i < n + 1; i++) {
            dp[i][0] = 1; // deal with j = 0
            for (int j = 1; j < Math.min(k, i * (i - 1) / 2) + 1; j++) {
                dp[i][j] = (dp[i][j - 1] + dp[i - 1][j] - (j >= i ? dp[i - 1][j - i] : 0)) %
mod;
                // all dp[i][j] modulo 10^9 + 7
                // so dp[i - 1][j - 1] might bigger than dp[i][j - 1] + dp[i - 1][j]
                if (dp[i][j] < 0)
```

```
                    dp[i][j] += mod;
                }
            }
        
        return dp[n][k];
    }
}
```

归并排序

```
public class MergeSort {
    private int[] numbers;
    private int[] helper;

    private int number;

    public void sort(int[] values) {
        this.numbers = values;
        number = values.length;
        this.helper = new int[number];
        mergesort(0, number - 1);
    }

    private void mergesort(int low, int high) {
        // check if low is smaller than high, if not then the array is sorted
        if (low < high) {
            // Get the index of the element which is in the middle
            int middle = low + (high - low) / 2;
            // Sort the left side of the array
            mergesort(low, middle);
            // Sort the right side of the array
            mergesort(middle + 1, high);
            // Combine them both
            merge(low, middle, high);
        }
    }

    private void merge(int low, int middle, int high) {

        // Copy both parts into the helper array
        for (int i = low; i <= high; i++) {
            helper[i] = numbers[i];
        }

        int i = low;
        int j = middle + 1;
        int k = low;
        // Copy the smallest values from either the left or the right side back
        // to the original array
        while (i <= middle && j <= high) {
            if (helper[i] <= helper[j]) {
                numbers[k] = helper[i];
                i++;
            } else {
                numbers[k] = helper[j];
                j++;
            }
            k++;
        }
        // Copy the rest of the left side of the array into the target array
        while (i <= middle) {
            numbers[k] = helper[i];
            k++;
            i++;
        }

    }

    public static void main(String[] args) {
        int arr[] = { 78, 9, 45, 7, 2, 90 };
        new MergeSort().sort(arr);
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + "\t");
        }
    }
}
```