一、查找算法：

1、遍历查找

```
public class Main {
    public static void ooo(int[] n, int k, int target) {
            for (int i = 0; i < k; i++) {
                if (n[i] == target) {
                    return n[i];
                }
            }
     }
```

2、二分查找

```
public static int ooo(int[] n, int start, int end, int target) {
      if (start > end) {
            return -1; }//-1 代表不存在该值
      int mid = start + (end - start) / 2;
      if (n[mid] == target) {
            return n[mid];
      } else if (n[mid] > target) {
            return ooo(n, start, mid - 1, target);
      } else {
            return ooo(n, mid + 1, end, target);
      }
}
```

3、Horspool

```
public class Main {
      private static final int SIZE = 500；//这里是规定字符串的长度
        public int horspoolMatching(String text, String pattern) {
            int[] shiftTable = new int[SIZE];//设置移动表
            int m = pattern.length();
            for (int i = 0; i < SIZE; i++) {
                  shiftTable[i] = m;
            }
            for (int i = 0; i < m - 1; i++) {
                  shiftTable[pattern.charAt(i)] = m - 1 - i;//规定移动距离
            }

// 从模式的末尾开始逐个字符比较，如果匹配则继续，否则根      据移动表移动指针
            int n = text.length();
            int i = m - 1;
            while (i < n) {
                  int k = 0;
```

```
            while (k < m && pattern.charAt(m - 1 - k) == text.charAt(i - k)) {
                k++;
            }
            if (k == m) {
                return i - m + 1;//如果匹配就返回初始位置，不匹配就继续移动
            } else {
                i += shiftTable[text.charAt(i)];
            }
        }
        return 0;//0 代表完全不匹配
    }
```

4、最长公共子序列

```
public static String LCS(String X, String Y) {
    int m = X.length();
    int n = Y.length();
    int[][] dp = new int[m + 1][n + 1];
    String[][] solution = new String[m + 1][n + 1];
    // 构建 dp 和 solution 矩阵
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (X.charAt(i - 1) == Y.charAt(j - 1)) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                solution[i][j] = solution[i - 1][j - 1] + X.charAt(i - 1);
            } else {
                dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                // 选择长度更长的子序列
                if (dp[i - 1][j] > dp[i][j - 1]) {
                    solution[i][j] = solution[i - 1][j];
                } else {
                    solution[i][j] = solution[i][j - 1];
                }
        }
    }
}
    return solution[m][n];//这里我图省事儿直接返回序列了，长度我们可以在主方法里面搞
}
```