

```

    if (arr == null || arr.length == 0) {
        return 0;
    }
    int[][] f = new int[arr.length][arr.length];
    int[][] s = new int[arr.length][arr.length];
    for (int j = 0; j < arr.length; j++) {
        f[j][j] = arr[j];
        for (int i = j - 1; i >= 0; i--) {
            f[i][j] = Math.max(arr[i] + s[i + 1][j], arr[j] + s[i][j - 1]);
            s[i][j] = Math.min(f[i + 1][j], f[i][j - 1]);
        }
    }
    return Math.max(f[0][arr.length - 1], s[0][arr.length - 1]);
}

```

如上的 win2 方法中，矩阵  $f$  和  $s$  一共有  $O(N^2)$  个位置，每个位置计算的过程都是  $O(1)$  的比较过程，所以 win2 方法的时间复杂度为  $O(N^2)$ ，额外空间复杂度为  $O(N^2)$ 。

## 跳跃游戏

### 【题目】

给定数组  $arr$ ， $arr[i]=k$  代表可以从位置  $i$  向右跳  $1 \sim k$  个距离。比如， $arr[2]=3$ ，代表从位置 2 可以跳到位置 3、位置 4 或位置 5。如果从位置 0 出发，返回最少跳几次能跳到  $arr$  最后的位置上。

### 【举例】

$arr=[3,2,3,1,1,4]$ 。

$arr[0]=3$ ，选择跳到位置 2； $arr[2]=3$ ，可以跳到最后的位置。所以返回 2。

### 【要求】

如果  $arr$  长度为  $N$ ，要求实现时间复杂度为  $O(N)$ 、额外空间复杂度为  $O(1)$  的方法。

### 【难度】

士 ★☆☆☆

## 【解答】

具体过程如下：

1. 整型变量 `jump`，代表目前跳了多少步。整型变量 `cur`，代表如果只能跳 `jump` 步，最远能够达到的位置。整型变量 `next`，代表如果再多跳一步，最远能够达到的位置。初始时，`jump=0`，`cur=0`，`next=0`。

2. 从左到右遍历 `arr`，假设遍历到位置 `i`。

1) 如果 `cur>=i`，说明跳 `jump` 步可以到达位置 `i`，此时什么也不做。

2) 如果 `cur<i`，说明只跳 `jump` 步不能到达位置 `i`，需要多跳一步才行。此时令 `jump++`，`cur=next`。表示多跳了一步，`cur` 更新成跳 `jump+1` 步能够达到的位置，即 `next`。

3) 将 `next` 更新成 `math.max(next, i+arr[i])`，表示下一次多跳一步到达的最远位置。

3. 最终返回 `jump` 即可。

具体过程请参看如下代码中的 `jump` 方法。

```
public int jump(int[] arr) {
    if (arr == null || arr.length == 0) {
        return 0;
    }
    int jump = 0;
    int cur = 0;
    int next = 0;
    for (int i = 0; i < arr.length; i++) {
        if (cur < i) {
            jump++;
            cur = next;
        }
        next = Math.max(next, i + arr[i]);
    }
    return jump;
}
```

## 数组中的最长连续序列

### 【题目】

给定无序数组 `arr`，返回其中最长的连续序列的长度。

### 【举例】

`arr=[100,4,200,1,3,2]`，最长的连续序列为`[1,2,3,4]`，所以返回 4。