

1.

设 n 个不同的整数排序后存于 `T[0:n-1]` 中。例如，存在一个下标 $i(0 \leq i < n)$ ，使得 `T[i]=i`，设计一个有效算法找到这个下标。要求算法在最坏情况下的计算时间为 $O(\log n)$ 。

因为是有序的整数数组，需要查找一个 `T[i]==i` 的数，因此考虑使用 二分查找 方法。

```
def binary_search(arr):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2

        if arr[mid] == mid:
            return mid      # 找到 T[i] = i
        elif arr[mid] > mid:
            high = mid - 1
        else:
            low = mid + 1

    return -1
```

使用三组符合条件的数据进行测试

```
print(binary_search([-10, -5, 0, 3, 7]))
print(binary_search([0, 2, 5, 8, 17]))
print(binary_search([-10, -5, 3, 4, 7]))
```

得到输出如下：

```
> python binary_search.py
3
0
-1
```

算法的时间复杂度是 $O(\log n)$

2.

设计一个 $O(n^2)$ 时间的算法，找出 n 个数组成的序列的最长单调递增子序列。

最长单调递增子序列问题，使用动态规划解决。

定义数组 `dp[i]`，表示以第 i 个元素结尾的最长递增子序列的长度。

得出状态转移方程： $dp[i] = \max(dp[j] + 1)$ 对于所有 $j < i$ 且 $arr[j] < arr[i]$

初始化 `dp` 数组为 `1`，因为每个字母都是自己的单调递增子序列

```
def LIS(arr):
    n = len(arr)
    if (n == 0):
        return 0    # 如果数组为空，直接返回0

    dp = [1] * n    # 初始化dp数组：长度为n，每个元素都为1

    for i in range (n):
        for j in range (i):
            if arr[j] < arr[i]:
                dp[i] = max(dp[i], dp[j] + 1)

    return max(dp)    # 返回dp数组中的最大值
```

测试数据如下：

```
print(LIS([10, 9, 2, 5, 3, 7, 101, 18]))
print(LIS([0, 1, 0, 3, 2, 3]))
print(LIS([7, 7, 7, 7, 7]))
```

得到输出如下：

```
> python LIS.py
4
4
1
```

时间复杂度为 $O(n^2)$, 空间复杂度为 $O(n)$