数据结构第一次上机实验报告

林子开

2023 年 9 月 17 日

目录

1 插入排序代码

Listing 1: insertion sort 源代码

```
def insertiong_sort(seq):
    """插入排序，返回从小到大的升序数组"""
    n =  len(seq)
    for j in  range(1,n):
        key = seq[j]
        i = j - 1
        while (i >= 0) and (seq[i] > key):
            seq[i+1] = seq[i]
            i = i - 1
        seq[i+1] = key
    return seq
```

2 merge 排序代码

Listing 2: merge sort 源代码

```
from math import *
def merge_sort(seq):
    """merge排序，返回从小到大的升序数组"""
    n =  len(seq)
    if n == 1:
        return seq
```

```
    if n//2 == 0:
        A1 = merge_sort(seq[0: int(n/2)])
        A2 = merge_sort(seq[ int(n/2):])
    else: # 如果是奇数
        A1 = merge_sort(seq[0:ceil(n/2)]) # 元素升序排列
        A2 = merge_sort(seq[ceil(n/2):]) # 元素升序排列
    # 以下是merge过程，使用双指针思想
    i = 0
    j = 0
    sorted_seq = []
    while i <  len(A1) and j <  len(A2):
        if A1[i] <= A2[j]:
            sorted_seq.append(A1[i])
            i += 1
        else: # if A1[i] > A2[j]
            sorted_seq.append(A2[j])
            j += 1
    sorted_seq.extend(A1[i:])
    sorted_seq.extend(A2[j:])
    return sorted_seq
```

3 寻找最优的 k

3.1 f(n,k) 表达式的推导

在本节中，为方便讨论，如无特别说明，均考虑最差情况，即输入为降序排列的列表，输出为升序排列的列表。

对于 merge sort，设 $T(n) = 2T(\frac{n}{2}) + dn$，则在递归的每一层中，merge 过程的复杂度都为 dn。

对于 insertion sort，由于我们已假设只考虑最差情况，则其时间复杂度为 $g(n) = an^2 + bn + c$，其中 a, b, c 均为常数。

当列表长度小于或等于 k 时，就不再递归，直接调用插入排序。设递归开始时处在 level 0，并设假设在 level $(m-1)$ 时 $\frac{n}{2^{m-1}} > k$，在 level m 时 $\frac{n}{2^m} \leq k$，那么在 level m 时调用 insertion sort。同时，可以知道 k 与 m 满足的关系为：

$$k = \lceil \frac{n}{2^m} \rceil \tag{1}$$

$$m = \lceil \log_2(\frac{n}{k}) \rceil \tag{2}$$

画出对应的递归树如下：

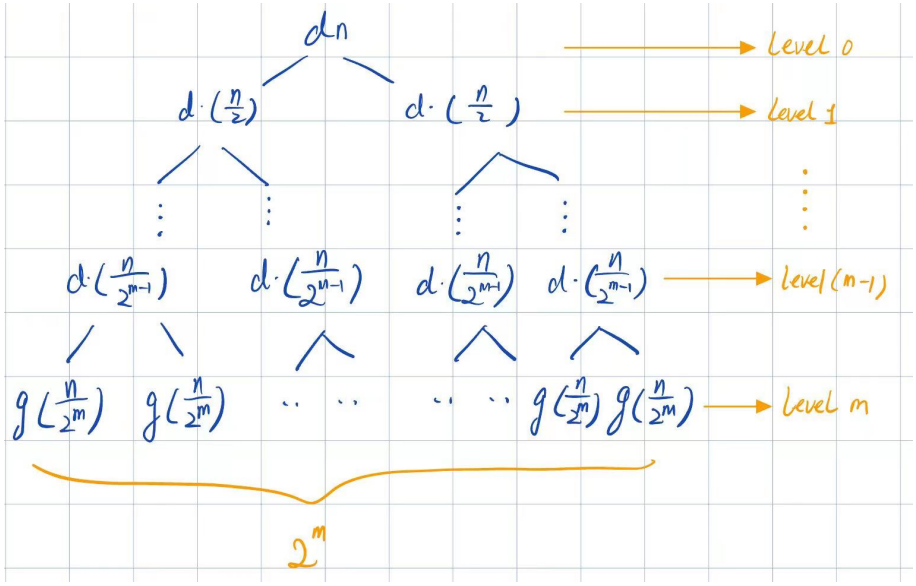


图 1: combined sort 递归树

由递归树可以得到总的时间复杂度为

$$\begin{aligned} f(n,m) &= dnm + 2^m g(\frac{n}{2^m}) \\ &= dnm + 2^m \left[a(\frac{n}{2^m})^2 + b(\frac{n}{2^m}) + c\right] \end{aligned} \tag{3}$$

将式(1) (2)代入，在考虑渐进情况下的复杂度时，可忽略取整，于是得到：

$$\begin{aligned} f(n,k) &= dn(\log_2 n - \log_2 k) + (\frac{n}{k})(ak^2 + bk + c) \\ &= dn\log_2 n - dn\log_2 k + ank + nb + \frac{nc}{k} \end{aligned} \tag{4}$$

3.2 理论推导最优 k

下面从理论上寻找最优的 k。当 n 既定时，将 $f(n,k)$ 对 k 求偏导得到：

$$\begin{aligned} \frac{\partial f}{\partial k} &= -dn\frac{1}{k}\log_2 \mathrm{e} + an - \frac{nc}{k^2} \\ &= \frac{1}{k^2}(ank^2 - dn\log_2 \mathrm{e}k - nc) \end{aligned} \tag{5}$$

其中 e 是自然对数的底。

由式(5)，结合二次函数性质可知，在 $k \geq 0$ 时只有一个极值点，而且是极小值点，该极值点为：

$$k^* = \frac{dn\log_2 \mathrm{e} + \sqrt{(dn\log_2 \mathrm{e})^2 - 4acn^2}}{2an} \tag{6}$$

则理论上当 $k = k^*$ 时，在渐进情况下，算法会有最快的速度。

3.3 实验寻找最优 k

在实际情况下，由于常数 a, b, c, d 并不知道，所以需要通过实验来确定最优的 k^*。在实验中，还需要测试平均情况（多组随机输入求平均值），以及最坏情况（倒序输入）。我们把可行的实验途径罗列如下，共四种：

1. **求常数 + 最坏情况**。对 insert sort 和 merge sort 取不同的 n，分别输入长度为 n 的倒序数组并记录时间，再通过线性回归的方式估计最坏常数 a, b, c, d，带回表达式(6)，得到最优 k。

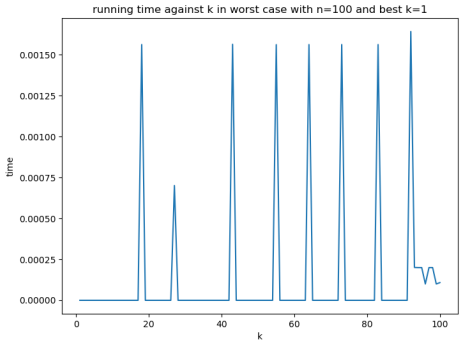
2. **求常数 + 随机情况**。对 insert sort 和 merge sort 取不同的 n，分别多次输入长度为 n 的随机数组并求时间的平均值，再通过线性回归的方式估计随机情况下的常数 a, b, c, d，带回表达式(6)，得到最优 k。但此方法不够稳定，可能会受伪随机数生成规律的影响而不准确。

3. **固定 n+ 最坏情况**。固定数组大小 n，输入长度为 n 的倒序数组，尝试不同的 k 值并记录时间，找到时间最短的 k。

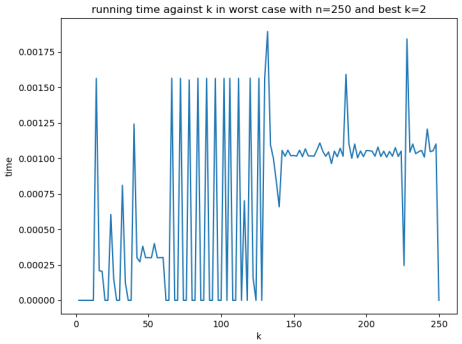
4. **固定 n+ 随机情况**。固定数组大小 n，多次输入长度为 n 的随机数组，尝试不同的 k 值并记录平均时间，找到平均时间最短的 k。同样，此方法不够稳定，可能受伪随机数生成规律的影响。

在本次报告中，我们只进行第 3 和第 4 类实验。

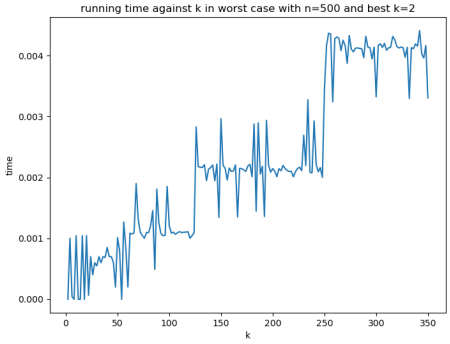
我们考虑 $n = 100, 250, 500, 1000$ 四种情况，输入最坏情况的倒序数组寻找使运行时间最短的 k，以及输入多个随机排序数组并求平均时间最短的 k，共八个实验，结果如下：



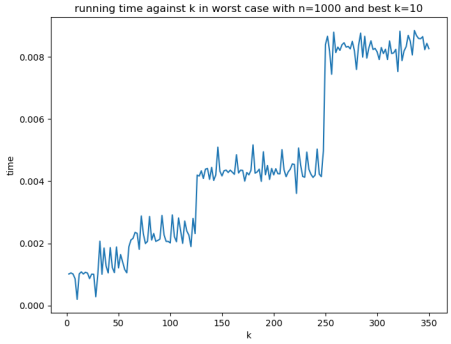
(a) n=100, best k=1



(b) n=250, best k=2

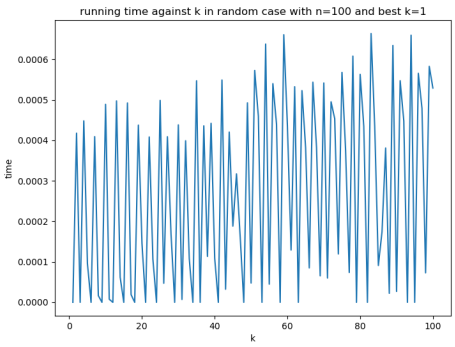


(c) n=500, best k=2

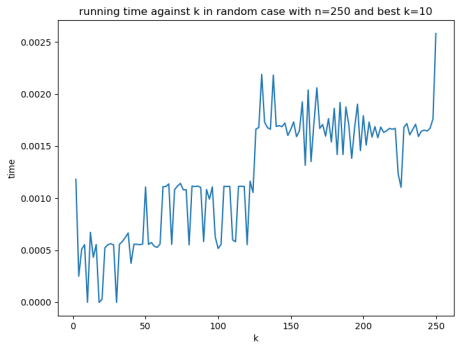


(d) n=1000, best k=10

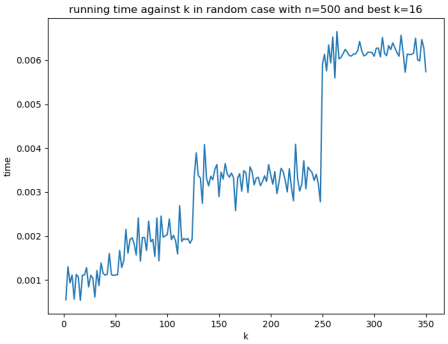
图 2: 在最坏倒序数组的情况下寻找最优 k



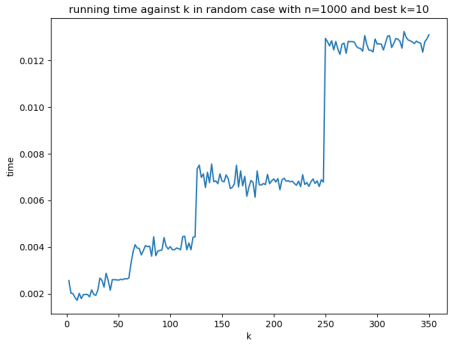
(a) n=100, best k=1



(b) n=250, best k=10



(c) n=500, best k=16



(d) n=1000, best k=10

图 3: 在多个随机排序数组的情况下寻找使平均时间最短的 k

可以发现，在随机情况下，k 并没有明显随着 n 的增大而增大。

4 融合后的排序代码

Listing 3: combined sort 源代码

```
def insertion_sort(seq):
    """插入排序，返回从小到大的升序数组"""
    n =  len(seq)
    if n == 1:
        return seq
    for j in  range(1,n):
        key = seq[j]
        i = j - 1
        while (i >= 0) and (seq[i] > key):
            seq[i+1] = seq[i]
            i = i - 1
        seq[i+1] = key
    return seq


def combined_sort(seq,k):
    """主体为merge排序，当数组长度小于k时，不再递归，直接使用插入排序。
    输入：seq为待排序的数组，k为两种排序方式的临界值
```

```
    输出：该函数返回从小到大的升序数组。"""
    n =  len(seq)
    # 如果数组太短，不再递归，直接调用插入排序的函数
    if n <= k:
        return insertion_sort(seq)
    if n//2 == 0:
        A1 = combined_sort(seq[0: int(n/2)],k)
        A2 = combined_sort(seq[ int(n/2):],k)
    else: # 如果是奇数
        A1 = combined_sort(seq[0:ceil(n/2)],k) # 元素升序排列
        A2 = combined_sort(seq[ceil(n/2):],k)  # 元素升序排列
    # 以下是merge过程，使用双指针思想
    i = 0
    j = 0
    sorted_seq = []
    while i <  len(A1) and j <  len(A2):
        if A1[i] <= A2[j]:
            sorted_seq.append(A1[i])
            i += 1
        else: # A1[i] > A2[j]
            sorted_seq.append(A2[j])
            j += 1
    sorted_seq.extend(A1[i:])
    sorted_seq.extend(A2[j:])
    return sorted_seq
```