算法设计与分析 (2017 年春季学期)

第一次作业参考答案

1 对下面每一对表达式 (A, B)，请判断 A 和 B 之间的关系是 O, Ω 还是 Θ。注意他们之间可能满足多种关系。**(每小题 4 分，共 20 分)**

1. $A = n^3 - 100n, B = n^2$;
2. $A = \log n, B = \log_{1.1} n$;
3. $A = 2^{2n}, B = 2^{3n}$;
4. $A = 2^{\log n}, B = n$;
5. $A = \log\log n, B = 10^{100}$.

解：

1. $A = \Omega(B)$;
2. $A = O(B), A = \Omega(B), A = \Theta(B)$;
3. $A = O(B)$;
4. $A = O(B), A = \Omega(B), A = \Theta(B)$;
5. $A = \Omega(B)$.

2 请给出 $T(n)$ 尽可能紧凑的渐进上界并予以说明，可以假定 n 是 2 的幂次。**(每小题 3 分，共 21 分)**

1.

$$T(1) = T(2) = 1$$
$$T(n) = T(n-2) + 1 \quad if \quad n > 2$$

2.

$$T(1) = 1$$
$$T(n) = T(n/2) + 1 \quad if \quad n > 1$$

3.

$$T(1) = 1$$
$$T(n) = T(n/2) + n \quad if \quad n > 1$$

4.

$$T(1) = 1$$

$$T(n) = 2T(n/2) + 1 \quad if \quad n > 1$$

5.

$$T(1) = 1$$

$$T(n) = 4T(n/2) + 1 \quad if \quad n > 1$$

6.

$$T(1) = 1$$

$$T(n) = 3T(n/2) + n^2 \quad if \quad n > 1$$

7.

$$T(1) = 1$$

$$T(n) = T(n/2) + \log n \quad if \quad n > 1$$

解：

1. $T(n) = O(n)$
2. $T(n) = O(\log n)$
3. $T(n) = O(n)$
4. $T(n) = O(n)$
5. $T(n) = O(n^2)$
6. $T(n) = O(n^2)$
7. $T(n) = O(\log^2 n)$

   原式展开为：

   $$\log n + \log(n/2) + \log(n/4) + \cdots + \log 2 + 1 = 1 + (1 + 2 + \cdots + \log n) \leq \sum_{i=1}^{\log n} i = O(\log^2 n)$$

3 k 路归并问题 (19 分)

现有 k 个有序数组 (从小到大排序)，每个数组中包含 n 个元素。您的任务是将它们合并成 1 个包含 kn 个元素的有序数组。首先来回忆一下课上讲的归并排序算法，它提供了一种合并有序数组的算法 $Merge$。如果我们有两个有序数组大小分别为 x 和 y，$Merge$ 算法可以用 $O(x+y)$ 的时间来合并这两个数组。

1. 如果我们应用 $Merge$ 算法先合并第一个和第二个数组，然后由前后两个数组合并后的数组与第三个合并，再与第四个合并，直到合并完 k 个数组。请分析这种合并策略的时间复杂度 (请用关于 k 和 n 的函数表示)。
2. 针对本题的任务，请给出一个更高效的算法，并分析它的时间复杂度。(提示：此题若取得满分，所设计算法的时间复杂度应为 $O(nk\log k)$。)

解：

1. 题目中给出的 $Merge$ 算法时间复杂度是线性的，根据题目中的策略对数组进行和并，每次和并的复杂度分别为 $n+n, 2n+n, \ldots, (k-1)n+n$。总的复杂度为：

$$\left(n\sum_{i=1}^{k-1} i\right) + (k-1)n = n\frac{k(k-1)}{2} + (k-1)n = n\frac{k^2-k}{2} + k - 1 = O(nk^2)$$

2. 一种更高效的做法是把 k 个有序数组平均分为两份递归进行合并得到两个数组，然后再合并这两个数组。这种方法的复杂度递归式为 $T(k) = 2T(k/2) + O(nk), T(1) = O(n)$，解出时间复杂度为 $O(nk\log k)$。算法实现可以参考 **Algorithm** 1。

Algorithm 1 $k\_Merge(A, l, r)$

Input:
k 个包含 n 个元素的有序数组, $A[1..k][1..n]$;
递归区间左端点, l;
递归区间右端点, r;
Output:
归并后的包含 $(r-l+1)n$ 个元素的有序数组;

```
if l = r then
    return A[l][1..n];
end if
m ← ⌊(l+r)/2⌋;
return Merge(k_Merge(A, l, m), k_Merge(A, m+1, r));
```

4 局部最小值问题 (20 分)

给定一个由 $n(n \geq 3)$ 个互不相同的整数组成的数组 $A[1..n]$，其满足 $A[1] > A[2]$ 并且 $A[n-1] < A[n]$。我们定义数组的**局部最小值**为比它的两个相邻元素 (如果存在) 都小的整数。换言之，$A[x]$ 是局部最小值当且仅当它满足 $A[x] < A[x-1]$ 并且 $A[x] < A[x+1](1 < x < n)$。例如，下图所示数组中包含两个局部最小值，分别为 3 和 1。

9	3	7	2	1	4	5

求局部最小值显然有一个 $O(n)$ 的做法，仅需要扫描一遍整个数组就可以找到所有的局部最小值。请你给出一个算法可以在 $O(\log n)$ 的时间复杂度内找出一个数组的局部最小值。如果局部最小值有多个，仅需要找出任意一个局部最小值即可。(提示：我们给出的限制条件保证数组至少有一个局部最小值。)

解：
如果 n 等于 3，做法是显然的。考虑 n 大于 3 的情况, 此时令 $m = \lfloor \frac{n}{2} \rfloor$，来检查 $A[m-1], A[m], A[m+1]$ 之间的大小关系：

1. 如果 $A[m-1] > A[m]$ 并且 $A[m] < A[m+1]$，那么 A[m] 就是局部最小值，算法结束。
2. 如果 $A[m-1] < A[m] < A[m+1]$，那么 $A[1..m]$ 中一定存在局部最小值，我们递归处理数组 $A[1..m]$ 即可。
3. 如果 $A[m-1] > A[m] > A[m+1]$，那么 $A[m..n]$ 中一定存在局部最小值，我们递归处理数组 $A[m..n]$ 即可。
4. 如果 $A[m-1] > A[m]$ 并且 $A[m] < A[m+1]$，那么左右两个区间中都存在局部最小值，我们任选一个区间递归处理即可。

任何一种情况每次递归都缩减了一半的规模，因此可以得出递归式 $T(n) \leq T(n/2) + O(1)$，解出算法的时间复杂度为 $T(n) = O(\log n)$

5 字符串等价关系判定问题 (20 分)

给定两个长度为 n 的字符串 A 和 B，若称 A 与 B 是**等价**的，当且仅当它们满足如下关系之一：

1. A 和 B 完全相同;
2. 若将把 A 分成长度相等的两段 A_1 和 A_2，也将 B 分成长度相等的两段 B_1 和 B_2。且他们之间满足如下两种关系之一：
   a. A_1 和 B_1 等价且 A_2 和 B_2 等价;
   b. A_1 和 B_2 等价且 A_2 和 B_1 等价;

请你设计一个高效的算法来判断两个字符串是否等价并分析你的算法的时间复杂度。

解法 1：
通过观察可以发现，题目中所描述的**等价**事实上是一种等价关系。它满足自反性、对称性和传递性，这意味着我们可以把这些字符串划分为不同的等价类。如果对每个等价类找出它们中字典序最小的一个字符串作为其代表元，这样仅需判断它们的代表元是否相等就可以判断两个字符串是否等价了。
给出一个字符串，找和它属于同一等价类的字典序最小元的方法也很简单，就是递归处理字符串的左右两个子串，把字典序较小的子串放在前面。算法实现请参考 **Algorithm** 2。

Algorithm 2 $Smallest(S[1..n])$

Input:
一个长度为 n 的字符串, $S[1..n]$;
Output:
字符串 S 所在等价类的最小元;

```
1: if n%2 = 1 then
2:     return S;
3: end if
4: m ← n/2;
5: S1 ← Smallest(S[1..m]);
6: S2 ← Smallest(S[m+1..n]);
7: if S1 < S2 then
8:     return S1 + S2;
9: else
10:    return S2 + S1;
11: end if
```

注：伪代码中的 $<$ 运算为字典序的比较，$+$ 运算为两个字符串直接拼接。例如 $ab < ac, bd < cd, ab + cd = abcd, cd + ab = cdab$。
每次递归都将问题分解为两个规模缩减一半的问题，之后合并由于涉及到字符串的比较以及拼接，时间复杂度为 $O(n)$，可以写出递归式 $T(n) = 2T(n/2) + O(n)$，解出算法的时间复杂度为 $O(n \log n)$。

解法 2（答出本解法可得 16 分）：
直接从等价关系入手，根据定义直接进行四次递归判断，可以写出一个递归算法 **Algorithm** 3。

Algorithm 3 $Trivial\_equivalence(A, B)$

Input:
两个长度为 n 的字符串, $A[1..n], B[1..n]$;
Output:
两个字符串是否等价;

```
1: if A = B then
2:     return true;
3: end if
4: if n%2=1 then
5:     return false;
6: end if
7: m ← n/2;
8: A1 ← A[1..m];
9: A2 ← A[m + 1..n];
10: B1 ← B[1..m];
11: B2 ← B[m + 1..n];
12: return Trivial_equivalence(A1, B1) and Trivial_equivalence(A1, B1) or
    Trivial_equivalence(A1, B2) and Trivial_equivalence(A2, B1);
```

上述算法在最坏的情况下需要进行四次递归计算，递归式为 $T(n) \leq 4T(n/2) + O(n)$，从而得到算法的时间复杂度为 $O(n^2)$，然而，通过如下分析，可以发现该算法仍有可以改进的地方。

1. 如果 A1 和 B1 等价且 A2 和 B2 等价，那么可以直接返回 A 和 B 等价。
2. 在 A1 和 B1 等价且 A2 和 B2 不等价的时候，我们没有必要再判断 A2 和 B1 以及 A1 和 B2 是否等价了，因为如果 A1 和 B2 等价，又因为之前已经判断过了 A1 和 B1 是等价的，因此 B1 和 B2 一定是等价的，又因为之前已经知道 A2 和 B2 不等价，因此 A2 和 B1 一定是不等价的。
3. 在 A1 和 B1 不等价的时候，我们显然也不需要判断 A2 和 B2 是否等价了，仅需要判断 A2 和 B1 以及 A1 和 B2 是否等价。

因此，该算法最多仅需进行 3 次递归计算。递归式为 $T(n) \leq 3T(n/2) + O(n)$，解出其时间复杂度为 $O(n^{\log_2 3})$。算法实现请参考 **Algorithm** 4。

Algorithm 4 $Equivalence(A, B)$

Input:

两个长度为 n 的字符串, $A[1..n], B[1..n]$;

Output:

两个字符串是否等价;

```
1: if A = B then
2:    return true;
3: end if
4: if n%2=1 then
5:    return false;
6: end if
7: m ← n/2;
8: A1 ← A[1..m];
9: A2 ← A[m + 1..n];
10: B1 ← B[1..m];
11: B2 ← B[m + 1..n];
12: if Equivalence(A1, B1) then
13:    if Equivalence(A2, B2) then
14:       return true;
15:    else
16:       return false;
17:    end if
18: else
19:    return Equivalence(A1, B2) and Equivalence(A2, B1);
20: end if
```