Homework 1

Name: 方嘉聪 ID: 2200017849

Problem 1 (Textbook 1.18). 对以下函数, 按照他们的阶从高到低排序; 如果两个函数的阶相同, 表示为 $f(n) = \Theta(g(n))$.

$$2^{\sqrt{2\log n}},\ n\log n,\ \sum_{k=1}^{n}\frac{1}{k},\ n2^n,\ (\log n)^{\log n},\ 2^{2n},\ 2^{\log\sqrt{n}}$$

$$n^3,\ \log n!,\ \log n,\ \log\log n,\ n^{\log\log n},\ n!,\ n,\ \log 10^n$$

◀

Answer. 我们用 $>$ 表示阶的大小关系:

$$n! > 2^{2n} > n2^n > (\log n)^{\log n} = n^{\log\log n} > n^3 > n\log n = \Theta(\log n!) >$$

$$n = \Theta(\log 10^n) > 2^{\log\sqrt{n}} > 2^{\sqrt{2\log n}} > \log n = \Theta(\sum_{k=1}^{n}\frac{1}{k}) > \log\log n$$

◁

Problem 2 (Textbook 1.19). 求解下列递推方程:

(1) $\begin{cases} T(n) = T(n-1) + n^2 \\ T(1) = 1 \end{cases}$

(2) $\begin{cases} T(n) = T(n/2) + T(n/4) + cn \\ T(1) = 1 \end{cases}$

(3) $\begin{cases} T(n) = 5T(n/2) + (n\log n)^2 \\ T(1) = 1 \end{cases}$

(4) $\begin{cases} T(n) = T(n-1) + \frac{1}{n} \\ T(1) = 1 \end{cases}$

◀

Answer. (1) 用迭代法:

$$T(n) = n^2 + (n-1)^2 + \cdots + 4 + 1 = \frac{n(n+1)(2n+1)}{6}$$

(2) 使用递归树法: 设数深为 k, 那么 $n/2^k \geq 1 \implies k \leq \log n$:

$$T(n) = \sum_{k=0}^{\log n}\left(\frac{3}{4}\right)^k cn = 4cn\left(1 - \left(\frac{3}{4}\right)^{\log n}\right) = \Theta(n)$$

(3) 由主定理, $f(n) = (n\log n)^2 = O(n^{\log_2 5-\epsilon})$, $T(n) = \Theta(n^{\log_2 5})$

(4) 用迭代法:

$$T(n) = \sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$$

$\triangleleft$

Problem 3 (Textbook 1.21). 设原问题规模为 n, 从下述算法中选择一个最坏情况下时间复杂度最低的算法, 简述理由:

1. 算法 A: 将原问题划分为规模减半的 5 个子问题, 在线性时间内合并结果.
2. 算法 B: 将原问题划分为 2 个规模为 $n-1$ 的子问题, 在常量时间内合并结果.
3. 算法 C: 将原问题划分规模为 $n/3$ 的 9 个子问题, 在 $O(n^3)$ 内合并结果.

$\blacktriangleleft$

Answer. 选择算法 **A**, 理由如下:

1. 算法 A: $T(n) = 5T(n/2) + O(n) \implies T(n) = \Theta(n^{\log_2 5})$.
2. 算法 B: $T(n) = 2T(n-1) + O(1) \implies T(n) = \Theta(2^n)$.
3. 算法 C: $T(n) = 9T(n/3) + O(n^3) \implies T(n) = \Theta(n^3)$.

算法 A 在最坏情况下复杂度最低. $\triangleleft$

Problem 4 (Complexity Bounds). For each blank, indicate whether A_i is in O, Ω,or Θ of B_i. More than one space per row can be valid.

A	**B**	O	Ω	Θ
$10n$	n	$\surd$	$\surd$	$\surd$
10	n	$\surd$		
n^2	$2n$		$\surd$	
n^{2021}	2^n	$\surd$		
$n^{\log 9}$	$9^{\log n}$	$\surd$	$\surd$	$\surd$
$\log(n!)$	$\log(n^n)$	$\surd$	$\surd$	$\surd$
$(3/2)^n$	$(2/3)^n$		$\surd$	
3^n	2^n		$\surd$	
$n^{1/\log n}$	1	$\surd$	$\surd$	$\surd$
$\log^5 n$	$n^{0.5}$	$\surd$		
n^2	$4^{\log n}$	$\surd$	$\surd$	$\surd$
$n^{0.2}$	$(0.2)^n$		$\surd$	
$\log\log n$	$\sqrt{\log n}$	$\surd$		
$\log(\sqrt{n})$	$\sqrt{\log n}$		$\surd$	

$\blacktriangleleft$

Problem 5 (Gauging Complexity from Code). Refer to the algorithms below for solving these problems. Give the **worst-case run-time complexity** in Big-Oh notation for each of the algorithms below.

Algorithm 1: Search an element in an array of length n

Input: An array A of size n and an element e
Output: Index of the element e

```
i ← 0;
while i < n do
    if A[i] = e then
        return i;
    end
    i ← i + 1;
end
return −1                                   // If element e is not found
```

Algorithm 2: Replace an element in an array of length n with a given element

Input: An array A of size n, an element e present in the array, and a new element E
Output: Updated array

```
i ← 0;
while i < n do
    currElement ← A[i];
    if currElement = e then
        A[i] ← E;
        return A;
    end
    else
        A[i] ← currElement;
    end
    i ← i + 1;
end
return A            // If element e is not found, return the original array
```

Algorithm 3: Sort an array of length n using bubble sort

Input: An array A of size n
Output: Sorted array

```
for i ∈ [0, n − 1) do
    /* Declare i as the index having the current smallest element       */
    for j ∈ [i + 1, n) do
        /* Find the smallest element in the range [i, n)                */
        if A[j] > A[j + 1] then
            smallest ← j
        end
    end
    swap (A[i], A[smallest]);
end
return A;
```

◀

Answer. (1) **Algorithm 1**: 在最坏情况下, 需要完整遍历一遍数组, 时间复杂度为 $\Theta(n)$.

(2) **Algorithm 2**: 在最坏情况下, 替换的值在数组的最后一位, 需要完整遍历一遍数组, 时间复杂度为 $\Theta(n)$.

(3) **Algorithm 3**: 设最坏情况下所需操作数为 $T(n)$, 则 (c_1, c_2 为常数):

$$T(n) = \sum_{i=0}^{n-2} (c_1(n-i-1) + c_2) = \Theta(n^2)$$

故时间复杂度为 $\Theta(n^2)$.

◁

Problem 6 (Partial Sum of a 1D Array). Given an array A consisting of n integers $A[1], A[2], \cdots, A[n]$. You want to obtain a 2D $n \times n$ array B where $B[i][j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$, i.e., $A[i] + A[i+1] + \cdots + A[j]$. The value of array entry $B[i][j]$ is left unspecified whenever $i > j$.

Here's a basic algorithm to solve this problem:

Algorithm 4: Basic Algorithm

for $i = 1$ *to* n **do**
  for $j = i + 1$ *to* n **do**
    Add up array entries $A[i]$ through $A[j]$;
    store the result in $B[i][j]$
  end
end

(1) For some function f you may choose, give a bound of $O(f(n))$ on the runtime of this algorithm.

(2) For this same function f, prove that the runtime of the algorithm is also $\Omega(f(n))$ (This shows an asymptotically tight bound of $\Theta(f(n))$ on the runtime).

(3) Although the algorithm you analysis in (1)(2) is the most intuitive way to solve the problem, after all, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve the problem, with an asymptotically better runtime. In other words, you should develop an algorithm with runtime $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.

◀

Answer. (1) 设 $T_1(n)$ 为 *Basic Algorithm* 的时间复杂度, 那么:

$$T_1(n) = \sum_{i=1}^{n} \left(1 + \sum_{j=i+1}^{n} (j - i + 2)\right) = \sum_{i=1}^{n} \left(\frac{n^2}{2} + \frac{i^2}{2} + \frac{5n}{2} - in - \frac{5i}{2} + 1\right) = \Theta(n^3)$$

故取 $f(n) = cn^3$(c 为常数), 则 $T_1(n) = O(n^3)$.

(2) 由于 $T_1(n) = \Theta(n^3)$, 故 $T_1(n) = \Omega(n^3)$.

(3) 注意到对于 $B[i][j](j > i)$, $\sum_{k=i}^{j-1} A[k]$ 已经在 $B[i][j-1]$ 中计算过, 无需重复计算, 可以用动态规划的思想优化算法:

Algorithm 5: Optimized Algorithm

```
for i = 1 to n do
    B[i][i] = A[i];
end
for i = 1 to n do
    for j = i + 1 to n do
        B[i][j] = B[i][j − 1] + A[j];
    end
end
```

设 $T_2(n)$ 为 *Optimized Algorithm* 的时间复杂度, 那么:

$$T_2(n) = \sum_{i=1}^{n}\left(1 + \sum_{j=i+1}^{n} 1\right) = \sum_{i=1}^{n}(n - i + 1) = \Theta(n^2)$$

令 $g(n) = cn^2$(c 为常数), 则 $T_2(n) = O(n^2)$.

$\triangleleft$