# 算法设计与应用基础 作业1

## 18340052 何泽

## 1.

> Show that log(n!) = Θ(nlogn)
>
> (Hint: To show an upper bound, compare n! with n n . To show a lower bound, compare it with (n/2)n/2.)

$$log(n!) = log(1) + log(2) + \ldots + log(n-1) + log(n)$$

上界：

$$log(1) + log(2) + \ldots + log(n-1) + log(n) \leq log(n) + log(n) + \ldots + log(n) = n \times log(n)$$

下界：

$$\begin{aligned} & log(1) + log(2) + \ldots + log(\frac{n}{2}) + \ldots + log(n-1) + log(n) \\ \geq\ & log(\frac{n}{2}) + \ldots + log(n-1) + log(n) \\ =\ & log(\frac{n}{2}) + log(\frac{n}{2}+1) + log(\frac{n}{2}+2) + \ldots + log(n-1) + log(n) \\ \geq\ & log(\frac{n}{2}) + log(\frac{n}{2}) + \ldots + log\frac{n}{2}) + log(\frac{n}{2}) \\ =\ & \frac{n}{2} \times log(\frac{n}{2}) \end{aligned}$$

故有：$log(n!) = \Theta(nlogn)$

## 2.

> Compute gcd(210, 588) two different ways: by finding the factorization of each number, and by using Euclid's algorithm.

① $210 = 2 \times 3 \times 5 \times 7$ ， $588 = 2 \times 2 \times 3 \times 7 \times 7$

故有$gcd(210, 588) = 2 \times 3 \times 7 = 42$

② $588 \div 210 = 2\ mod\ 168$　　$210 \div 168 = 1\ mod\ 42$　　$168 \div 42 = 4$

故$gcd(210, 588) = gcd(168, 210) = gcd(42, 168) = 42$

## 3.

> In the RSA cryptosystem, Alice's public key (N, e) is available to everyone. Suppose that her private key d is compromised and becomes known to Eve. Show that if e = 3 (a common choice) then Eve can efficiently factor N.

由于已知e=3和e对于$\phi(n)$的模反元素d，故有：$3d \equiv 1(mod\phi(n))$

由于$d < \phi(n)$，故倍数只能是1或2，也就是 $3d = phi(n) + 1$或$3d = 2\phi(n) + 1$

$\phi(n) = 3d - 1$或 $\frac{3d-1}{2}$ ，于是$\phi(n)$已知，于是有：

$$\begin{cases} n = p \times q \\ \phi(n) = (p-1)(q-1) = pq - p - q + 1 \end{cases}$$

只要解此解此二元二次方程组，便可以得到p和q的值。

## 4. Length of Longest Fibonacci Subsequence（题号873)

> A sequence X1, X2, ..., Xn is fibonacci-like if:
>
> - $n \geq 3$
> - $X_i + X_{i+1} = X_{i+2}, \ for \ all \ i + 2 \leq n$
>
> Given a strictly increasing array A of positive integers forming a sequence, find the length of the longest fibonacci-like subsequence of A. If one does not exist, return 0. (Recall that a subsequence is derived from another sequence A by deleting any number of elements (including none) from A, without changing the order of the remaining elements. For example, [3, 5, 8] is a subsequence of [3, 4, 5, 6, 7, 8].)
>
> Example:
>
> Input : [1, 2, 3, 4, 5, 6, 7, 8]
>
> Output : 5
>
> Explanation: The longest subsequence that is fibonacci-like: [1, 2, 3, 5, 8].

- 算法分析

  一开始我的想法很简单，就是暴力算，类似冒泡排序那样，从头到尾两个循环遍历，每找到两个数，就看之后有没有数值等于这两个数之和，代码是这样的

```
int lenLongestFibSubseq(vector<int>& A) {
    int cal = 2;
    int n=A.size();
    for(int i = 0; i < n; i++){
        for(int j = i + 1; j < n; j++){
            int tmpI = A[i];
            int tmpJ = A[j];
            int sum = tmpI + tmpJ;
            int cur = 2;
            while(find(A.begin()+j,A.end(),sum)!=A.end()){
                tmpI = tmpJ;
                tmpJ = sum;
                sum = tmpI + tmpJ;
                cur++;
            }
            cal = max(cal, cur);
        }
    }
    return cal < 3 ? 0 : cal;
}
```

过程没什么问题，只不过一提交，结果是这样的：

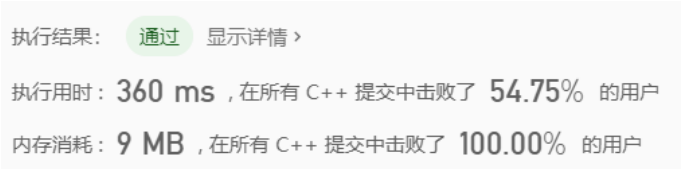
| 提交时间 | 提交结果 | 执行用时 | 内存消耗 | 语言 |
| --- | --- | --- | --- | --- |
| 几秒前 | 超出时间限制 | N/A | N/A | Cpp |

超时了，因为复杂度太高，下一步就是优化

首先我想到的是上面查找sum值的时候可以优化，因为是递增序列，所以改成二分查找可以快一些，但是改完发现还是超时，那么那么接下来我就考虑将数组存到hashmap里，整体思路跟前面是一样的， 遍历每一组组合，查找之后有没有数字等于它们的和，没有的话遍历下一对组合，有的话计算之后有没有数等于之前计算的和与第二个数之和：

```
int lenLongestFibSubseq(vector<int>& A) {
    int ans = 0;
    int n = A.size();
    unordered_map<int,bool> hash;
    for (int a: A)
        hash[a] = true;
    for (int i = 0; i < n; i++){
        for (int j = i + 1; j < n; j++){
            int cal = 0;
            if (hash.count(A[i] + A[j])){
                int u = A[i], v = A[j];
                while (hash.count(u + v)) {
                     cal++;
                     int temp = v;
                     v = u + v;
                     u = temp;
                }
                ans = max(ans, cal + 2);
            }
        }
    }
    return ans;
}
```

- 复杂度分析
  - 空间复杂度：$O(n)$
  - 时间复杂度：$O(n^2 log\, m)$，其中n为元素个数，m为A中最大的数
- 截图




## 5.Insertion Sort List（题号147）

> Sort a linked list using insertion sort.
>
> **Algorithm of Insertion Sort:**

> (a) Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.
>
> (b) At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.
>
> (c) It repeats until no input elements remain
>
> Example:
>
> Input: 4– > 2– > 1– > 3
>
> Output: 1– > 2– > 3– > 4

- 算法分析

  首先定义一个指针a的next指向头节点，再定义两个指针，分别指向当前节点和前一个结点，并比较，若当前值比前一个大，则继续遍历；若小于，则从指针a开始从前往后比较，找到该放的位置改变节点的next指向，从而实现插入排序。

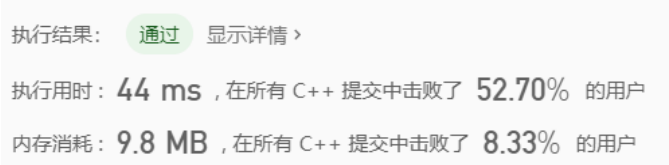
- 代码

```
ListNode* insertionSortList(ListNode* head) {
    ListNode* dummy = new ListNode(-1);
    a->next = head;
    ListNode* preNode = a;
    while (head != nullptr) {
        ListNode* curNode = head;
        head = head->next;
        if (preNode->val <= curNode->val) {
            preNode = curNode;
            continue;
        }
        preNode->next = curNode->next;
        ListNode* p = a;
        for (ListNode* p = a; p != head; p = p->next) {
            if (p->next->val < curNode->val)
                continue;
            curNode->next = p->next;
            p->next = curNode;
            break;
        }
    }
    head = a->next;
    delete a;
    return head;
}
```

- 复杂度分析

  时间复杂度：$O(n^2)$

- 提交截图



## 6. Merge k Sorted Lists（题号23）

> Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.
>
> Example:
>
> Input:
>
> [
>
> 1– > 4– > 5,
>
> 1– > 3– > 4,
>
> 2– > 6
>
> ]
>
> Output: 1– > 1– > 2– > 3– > 4– > 4– > 5– > 6

- 算法分析

  看到这道题的时候我首先想到的是如何合并两个有序链表，我的思路是如下采用递归：

```
ListNode* merge(ListNode* p1, ListNode* p2){
    if(!p1)
        return p2;
    if(!p2)
        return p1;
    if(p1->val <= p2->val){
        p1->next = merge(p1->next, p2);
        return p1;
    }
    else{
        p2->next = merge(p1, p2->next);
        return p2;
    }
}
```

  定义两个指针指向2个链表的头部，比较第一个元素，小的next指向大的，然后递归比较小的next和大的那个

  采用这种方式合并两个链表，那么合并k个链表只需要两两合并即可

- 代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* merge(ListNode* p1, ListNode* p2){
        if(!p1)
            return p2;
        if(!p2)
```

```
            return p1;
        if(p1->val <= p2->val){
            p1->next = merge(p1->next, p2);
            return p1;
        }
        else{
            p2->next = merge(p1, p2->next);
            return p2;
        }
    }

     ListNode* mergeKLists(vector<ListNode*>& lists) {
        if(lists.size() == 0)
            return NULL;
        ListNode* head = lists[0];
        for(int i = 1; i<lists.size(); ++i){
            if(lists[i]) head = merge(head, lists[i]);
        }
        return head;
    }
};
```

- 复杂度分析

  时间复杂度$O(kn)$， 空间复杂度$O(1)$

- 提交截图