算法设计与应用 作业1

姓名：TRY

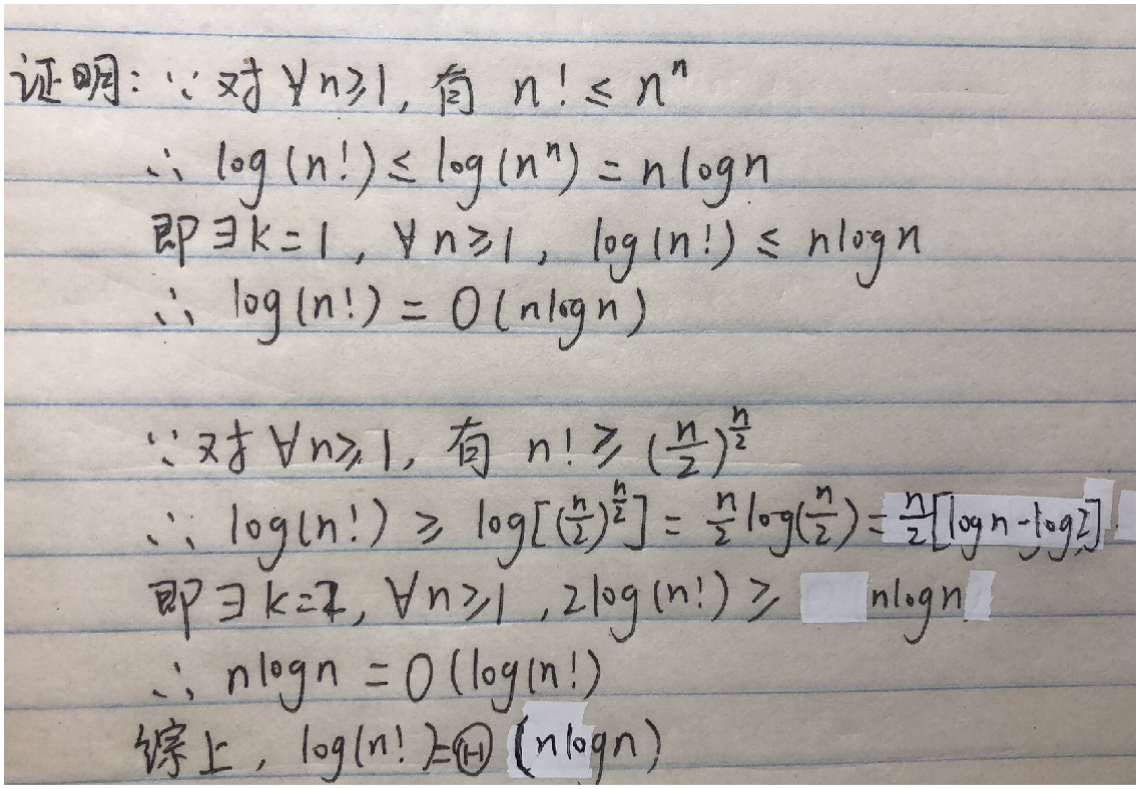
学号：

专业：计算机科学与技术

1.证明

$$log(n!) = \Theta(nlogn)$$

解：



2.用两种方法计算gcd(210,588)：因式分解法&Euclid's算法

方法一：因式分解法

解：

$$210 = 2 * 3 * 5 * 7,\ 588 = 2 * 2 * 3 * 7 * 7$$

所以

$$gcd(210, 588) = 2 * 3 * 7 = 42$$

方法二：Euclid's算法

解：由Euclid's rule可知：gcd(x, y) = gcd (x mod y, y)。（x\y都是正整数）

所以，

$$gcd(588,210)=gcd(588mod210,210)=gcd(168,210)=gcd(210,168)=gcd(210mod168,168)$$

$$=gcd(42,168)=gcd(168,42)=gcd(0,42)=42$$

3.证明题：

Q：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.

解：由欧拉函数可知， 对于一个正整数n，小于n且和n互质的正整数（包括1）的个数，记作φ(n)：

$$\phi(n)=n*(1-\frac{1}{p_1})*(1-\frac{1}{p_2})*...*(1-\frac{1}{p_n})=n*\prod_{i=1}^{n}(1-\frac{1}{p_i})$$

其中， p1, p2..... pn为n的所有质因数，n是不为0的整数。

而在RSA算法中，N=p*q，p和q都是两个大素数（故N只可以分解成这2个素数）。

所以由欧拉公式和 N=p*q 可得，

$$\phi(N)=N*(1-\frac{1}{p})*(1-\frac{1}{q})=N*(\frac{p-1}{p})*(\frac{q-1}{q})=(p-1)*(q-1)$$

在RSA算法中，根据d的定义 “d为e模 (p-1)(q-1) 操作的逆”，有

$$(e*d)\quad mod\quad [(p-1)(q-1)]=1$$

即

$$(e*d)\quad mod\quad \phi(N)=1$$

当e=3时，该等式化为

$$3d\quad mod\quad \phi(N)=1$$

不妨设

$$d<\phi(N)$$

（因为当d>=phi(N)时，一定存在d'=d mod phi(N) < phi(N)，使得d'也是e模phi(N)操作的逆。即一定存在d<phi(N)）

则有

$$3d<3\phi(N)<3\phi(N)+1$$

所以

$$3d=2\phi(N)+1\quad 或\quad 3d=\phi(N)+1$$

由题目条件可知，d 已知。所以由以上的两个情况可以解得

$$\phi(N)=(p-1)*(q-1)=\frac{3d-1}{2}\quad 或\quad 3d-1$$

再加上（由密匙公开，有N已知）

$$N = p * q$$

共有两条关于p、q未知变量的方程，可以解出p、q的值。

所以，Eve可以破解出N，即找到p、q的值。证毕。

4.【Leetcode 873】最长的斐波那契子序列的长度

本道题是计算斐波那契子系列的最大长度。可以使用两种方法：暴力法和动态规划法。

以下详细呈现"暴力法"的思路。

算法思路

- 先利用`vector`构建`unordered_set`集合。
  - **注意1**：如果在`vector`的基础上去做，会超时**time limit**！
  - **注意2**：`unordered_set`是一个类似于`set`的数据结构。（类似于`unordered_map`和`map`的关系）只不过是没有排序的集合，本质通过哈希实现。优点在于使用`unordered_set`的查找更快！
- 数组A进行二层循环遍历。对于每一个数对`A[i],A[j]`，令`y=A[i]+A[j]`，`x=A[j]`，然后利用`count`函数在集合里面查找`y`元素，如果集合中存在y，则更新`(x,y)`为`(y,x+y)`，然后继续查找。
  - **TIPS**：使用`count`函数查找元素（560ms），比使用`find`函数查找元素更快（604ms）。
- 当查找结束时，更新`result`的值。更新`j`的值，开始下次内层循环。
- 当j遍历完后，更新`i`的值，开始下一次的外层循环。

复杂度分析

时间复杂度：其中`n`是数组`A`的长度，最坏时间复杂度是：

$$[(n-2)+(n-3)+(n-4)+\ldots+1]+[(n-3)+(n-4)+\ldots+1]+\ldots+[2+1]+1$$

$$=\sum_{k=2}^{n-1}\sum_{t=1}^{n-k}t=\sum_{k=2}^{n-1}(n-k+1)*(n-k)/2=(n-2)*O(n^2)=O(n^3)$$

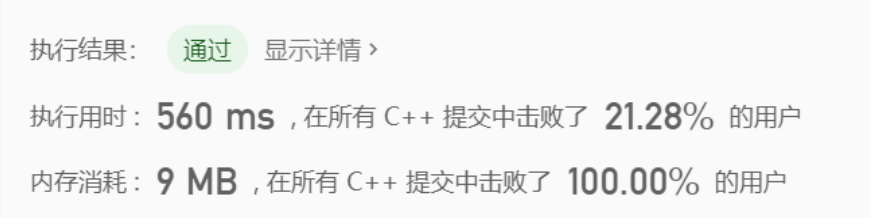
空间复杂度：O(N)，集合（set）`myset`使用的空间。

代码

```
int lenLongestFibSubseq(vector<int>& A) {
    unordered_set<int> myset(A.begin(),A.end());//用unordered_set查找效率更高
    int result = 0;
    for (int i = 0; i < A.size(); i++)
    {
        for (int j = i + 1; j < A.size(); j++)
        {
            int x = A[j], y = A[i] + A[j];//用加法的话：不用检查差和A[i]的相对顺序
```

```
            int length = 2;
            while (myset.count(y))//count和find函数都可以查找，而find返回迭代器，
count返回0/1
            {
                int temp = y + x;
                x = y;
                y = temp;
                result = max(result, ++length);
            }
        }
    }
    return result;
}
```

截图



5.【Leetcode 147】对链表进行插入排序

（由于本道题我在本寒假刷题时，已经完成并写过题解，所以以下内容部分来自我的题解）

算法思路

- 首先，第一反应是像数组的插入排序一样，找到相应插入位置后将元素逐个后移，但发现这样复杂度过高。
- 然后，为了降低复杂度，则要减少移动的次数：**改变指针的指向**！即遍历找到要插入的位置（不大于当前元素值的最后一个位置），通过指针指向的改变，来调整元素之间的顺序，在此位置后插入元素，实现插入排序。
- 此题利用了 **dummyhead伪头指针** ，即构造一个指向head的指针，使得可以从head开始遍历。
    - **注意：**由于指针在插入排序的过程中，所指向的元素在链表中的位置会发生改变，而 `dummyhead` 始终指向第一个元素不变，所以只有返回 `dummyhead->next` 才正确。
        - 在传参的时候，head指向的就是未排序中的第一个元素，且排序后head的指向仍未发生改变即始终指向排序前的第一个元素。但排序过程中，第一个元素的值可能发生改变。而dummyhead->next才会永远指向排序后的第一个元素。
- **tail指针** ：指向已排好元素的最后一个的指针，即此处我所用的prev指针。且由于测试样例中，很多样例都是已经按顺序排好的，故将node和prev进行比较，可**用于提速**！

4 分钟前	通过	16 ms	11.1 MB	Cpp
4 分钟前	通过	28 ms	11.2 MB	Cpp

  上图的两次提交分别都是使用了改变指针指向的方式。其中第一次（用时较长的）是while循环判断中取了=，（其实是更稳定的算法），而第二次（用时较短的）是while循环没取=的做法。（实际不够稳定）

复杂度分析

时间复杂度：最坏情况下，每一个下标为k的元素都要进行k次比较。

$$1+2+3+...+n=(n+1)*n/2=O(n^2)$$

空间复杂度：O(1)

代码

```
ListNode* insertionSortList(ListNode* head) {
    if (!head || !head->next)
        return head;
    ListNode *dummyhead = new ListNode(-1);//伪头指针
    dummyhead->next = head;
    ListNode *prev = head;
    ListNode *node = head->next;
    while (node)
    {
        if (node->val < prev->val)
        {
            ListNode* temp = dummyhead;//! ! temp要等于dummyhead，这样才可以比较第一个元素
            while (temp->next->val < node->val)//! ! ! 这里是temp->next：因为要修改前面的temp的指向
            {
                temp = temp->next;//指针后移
            }
            prev->next = node->next;
            node->next = temp->next;
            temp->next = node;
            node = prev->next;//此时不用改变prev指向！因为prev没有变，只是待排序元素变了位置。
        }
        else
        {
            prev = prev->next;
            node = node->next;
        }
    }
    return dummyhead->next;//!!!不能返回head! ! 因为后面会改变head所指向内存的位置!
}
```

截图

提交时间	提交结果	执行用时	内存消耗	语言
3 个月前	通过	16 ms	11.1 MB	Cpp

6.【Leetcode 23】合并K个排序列表

算法思路

这道题可以使用多种方法求解：如顺序合并、分治合并、使用优先队列合并。

此处详细分析“分治合并”方法。（另外两种方法可看力扣官方题解）步骤如下：

- 将k个已经排序好的链表不断“分治”，直到剩下1个链表
- 由于然后不断两两“合并”，通过使用两个临时指针，分别指向当前访问的链表中的元素，然后比较两者的大小，将较小的元素的节点接到已合并好的链表的后面，并且将对应的临时指针后移。
    - 如第一轮合并后，k个链表合并成了k/2个链表，平均长度为2n/k。第二轮合并后是k/4个链表。第三轮是k/8个链表等等。
- 重复这一过程 `log k` 轮，知道k个链表合并成为了1个链表。

复杂度分析

时间复杂度：

`n`是列表的最大长度，`k`为`lists`数组的大小。

第一轮递归合并链表数组成为 k/2个链表，每一个的时间为O(2n) ；第二轮递归合并链表数组成为k/4个链表，每一个的时间为O(4n)..... 以此类推，总时间为

$$O(\sum_{i=1}^{log_2^k} \frac{k}{2^i} * 2^i n) = O(kn * log_2^k)$$

也就是时间复杂度为

$$O(kn * logk)$$

空间复杂度：递归使用到O(log k) 空间代价的栈空间。所以复杂度为O(log k)。

代码

```
ListNode* mergeTwoLists(ListNode *a, ListNode *b)//合并两个列表
{
    if (!a || !b)
        return a ? a : b;
    ListNode* result = new ListNode(0);
    ListNode* cur = result;
    ListNode* temp1 = a;
    ListNode* temp2 = b;
    while (temp1 != NULL && temp2 != NULL)
    {
```

```
        if (temp1->val < temp2->val)
        {
            cur->next = temp1;
            temp1 = temp1->next;
            cur = cur->next;
        }
        else
        {
            cur->next = temp2;
            temp2 = temp2->next;
            cur = cur->next;
        }
    }
    cur->next = temp1 ? temp1 : temp2;
    ListNode* result1 = result->next;
    delete result;                          //删除掉前面new的内存，降低复杂度
    return result1;
}

ListNode* merge(vector <ListNode*> &lists, int l, int r)//合并列表
{
    if (l == r)
        return lists[l];
    else if (l > r)
        return NULL;
    else
    {
        int mid = (l + r) / 2;
        return mergeTwoLists(merge(lists, l, mid), merge(lists, mid + 1, r));
    }
}
ListNode* mergeKLists(vector<ListNode*>& lists) {
    if (lists.size() == 0)
        return NULL;
    else if (lists.size() == 1)
        return lists[0];
    else
        return merge(lists, 0, lists.size() - 1);
}
```

截图