Amazon OA 经典题目汇总

1.25匹马，找出3匹跑的最快的马，每次只能赛5匹马，而且，没有秒表等计时的工具

首先，将马分为a,b,c,d,e5组，每组分别比赛，分出排名，不妨假设排名如下

a1,a2,a3,a4,a5
b1,b2,b3,b4,b5
c1,c2,c3,c4,c5
d1,d2,d3,d4,d5
e1,e2,e3,e4,e5

然后将a1,b1,c1,d1,e1拿出来比赛，不妨假设排名a1>b1>c1>d1>e1

可以确定a1肯定是第一名，并且d组，e组肯定不会有能进入前三的马，那么接下想，哪些马可能会进入前三

对于a组，a2,a3都是有可能的，而a4,a5没有可能
对于b组，b1,b2都有可能，而剩下的都没可能
对于c组，只有c1有可能

所以有可能的马只有a2,a3,b1,b2,c1，刚好5匹，赛一次，取前二就可以

加强版：81匹马，找出6匹跑的最快的马，每次只能赛9匹马，问最少需要赛多少次

2.给出数组[a1,a2,a3...an,b1,b2,b3...bn]，要求不花额外的空间，变为[a1,b1,a2,b2,a3,b3...an,bn]

第一步，交换中间的一对
第二步，交换中间的两对
第三步，交换中间的三对
迭代N - 1次

举个例子：
a1 a2 a3 a4 b1 b2 b3 b4
a1 a2 a3 b1 a4 b2 b3 b4
a1 a2 b1 a3 b2 a4 b3 b4
a1 b1 a2 b2 a3 b3 a4 b4

3.求最长回文子串，所谓的回文串，就是将字符串反转后，与原字符串相同

首先，能想到暴力法，枚举所有的子串，判断该子串是否是回文，复杂度O(n ^ 3)

其次，可以想到中心扩展法，找到一个中点，然后向两边扩展，注意，中心有可能是一个字符，也有可能是两个字符，复杂度O(n ^ 2)

最后，还有一种O(n)的方法，被称为Manacher算法，由于篇幅及难度的原因，有兴趣的同学可以去学习一下

4.给出一个无序的数列a，求i,j满足i < j，并且a[j] - a[i]最大，要求在线性的时间内求出

统计每个数之前的最小值，然后用当前点的值减去最小值，与已知的最好结果比较，如果比结果大，那么就更新，时间复杂度O(n)

5.给出一个链表和一个数字k，要求交换正数第k个数和倒数第k个数(要求只遍历一次)

正数第k个数比较好找，直接计数就好，倒数第k个怎么办?
答案是，用两个指针，第一个指针先走，当走到第k个的时候，第二个出发开始走，这样，两个之间就相差了k-1，当第一个指针到达尾部的时候，第二个指针也就到达了倒数第k个数，然后交换这两个数就可以了

6.给定一个字符串，请找出其中无重复字符的最长子字符串。

地址 http://www.lintcode.com/zh-cn/problem/longest-substring-without-repeating-characters/

7.给出两个链表，表示两个数的每一位数字，要求这两个链表做加法，并返回一个链表，这个链表代表的是另一个数字

```
int FindLength(Node* n) { // find the length of a given linked list.
    int ret = 0;
    while(n) {
        ret++;
        n = n->next;
    }
    return ret;
}
Node* Add(Node* list1, Node* list2) { // this function is called first
    int state = FindLength(list1) - FindLength(list2);
      // if state > 0, list1 is longer
        // if state < 0, list2 is longer
        // if state == 0, list1 and list2 is of same length
    int carry = 0;
    Node* ret = Add2(list1, list2, carry, state); // add the two lists
    if ( carry > 0 ) { // handle carry for the leftmost digit
        Node* temp = new Node(carry);
        temp->next = ret;
        ret = temp;
    }
    return ret;
```

```
}
Node* Add2(Node* p1, Node* p2, int& carry, int state) { // helper function
    if ( p1 == NULL && p2 == NULL ) // if both are NULL, we are at the end
        return NULL;
    Node* ret = new Node(0); // create new node to return
    if ( state > 0 ) { // p1 is still longer than p2
                // only advance p1's pointer and decrease state
        ret->next = Add2(p1->next, p2, carry, state-1);
        ret->data = carry + p1->data; // just sum carry and p1's data
    }
    else if ( state < 0 ) { // p2 is still longer than p1
                // only advance p2's pointer and increase state
        ret->next = Add2(p1, p2->next, carry, state+1);
        ret->data = carry + p2->data; // just sum carry and p2's data.
    }
    else { // p1 and p2 are of same length
                // advance both pointers, state should stay untouched from now on(0).
        ret->next = Add2(p1->next, p2->next, carry, 0);
        ret->data = carry + p1->data + p2->data; // sum carry and both digits
    }
    carry = ret->data / 10; // calculate new carry
    ret->data %= 10;        // update the current data to be smaller than 10
    return ret;             // return the new node
}
```

8.现有8支队伍，每支队伍与其他队伍各进行2场比赛，只有胜负没有平局，而只有4支队伍会进入下一轮，问，至少赢多少场比赛，才能确保肯定能进下一轮

可以先看共有多少场比赛，8 * 7 * 2 / 2 = 56

共56场比赛，我们可以给每场比赛的胜者加一分，前4高的分数进入下一轮

可以采取构造的方法，构造出最坏的情况，什么是最坏的情况呢？后四名取的分尽可能的少，而前四名取得的分尽可能平均，这样，就将第四名的分数最大化。

首先，后四名的分数尽可能少，第8名，一场没胜，0分，第7名最少胜2场（至少得胜第8名2场），2分，同理第6名4分，第5名6分
那么现在剩下56 - 2 - 4 - 6 = 44分，而前四名尽可能平均，也就是每队11分。

再来想想每队11分能不能成立，因为前4名肯定都胜了后面四队，所以每队都已经有了8分，再看每队能不能只得11 - 8 = 3分，然后发现，如果每队和其他三队，刚好都是一胜一败时，刚好就是3分，因此，每队11分成立。

也就是说，至少得11分，才能确保肯定能进下一轮。

9.一个数组，数组里面的数先升序，再降序，并且最大值只有一个，求最大值的下标

典型的三分，二分也是可以的。

10.给出两个无环的单向链表，判断这两个链表是否有交点，如果有交点，求出交点，要求O(1)的空间，O(n)的时间

判断两个链表是否有交点比较容易，先找到第一个链表的尾节点，再找到第二个链表的尾节点，看这两个交点是否相同就可以
怎么判断交点呢？可以想到，交点以后两个链表就完全相同了，所以我们可以把两个链表尾部对齐，然后再依次遍历，比如说链表1的长度是len1，链表2的长度是len2，如果len1 > len2，那么链表1就先遍历len1 - len2个，这样两个链表就对齐了，然后再依次遍历后面的结点，直到找到相同的结点。

11.给出一个n个数的数组（无序数组），数的范围是0~n-1，有的数字出现了多次，有的数字没有出现，找出所有没出现过的数字，要求O(1)的空间，O(n)的时间

1.也许可以想到xor，如果一个数列只有1个数没有，并且其他数字只出现过1次，那么可以通过两个次异或得到，但这个题里面有的数字可能出现多次，所以不能用这种做法

2.还可以想到map，为每个数建立一个key，然后put进去，但这种做法的空间复杂度

是O(n)，所以也不能用这种做法
直接上代码

```
void repetitions(int A[],int n){
    int i = 0;
    for(i = 0; i < n; i++)
        A[A[i] % n] + = n;       //注意，这里可能有溢出的危险
    for(i = 0; i < n; i++) {
        frequency = A[i] / n;
        element = i;
        print("Element= %d , frequency = %d", element, frequency );
    }
}
```

12.不用乘法符号，计算两个数相乘

```
int Mul(int a,int b){
    int ans = 0;
    int sign = ((a^b)>>31);    //判断两个数的符号是否相同
    a = abs(a);
    b = abs(b);
    while(b){
        if(b & 0x01)    //看b的最后一位是否是1
            ans += a;
        b >>= 1;
        a <<= 1;
    }
    return sign ? (~ans+1) : ans;   //负数的补码是反码+1
}
```