1.（2697）字典序最小回文串

```
char * makeSmallestPalindrome(char * s){
int i=strlen(s)-1;
for(int j=0,k=i;j<=i/2;j++,k--){
if (s[j]!=s[i]);
s[j]=s[k]=fmin(s[j],s[k]);
}
return s;
}
```

从头和尾两个方向同时出发相互比较，进行排序比较；

2.（1935）可以输入的最大单词数

```
int canBeTypedWords(char *text, char *brokenLetters)
{
    int cnt = 0;

    const char s[2] = " "; // 用于拆分的空格

    char *token;             // 每次拆分后的指针


    /* 获取第一个子字符串 */

    token = strtok(text, s);

    /* 判断拆分是否结束 */

    while (token != NULL) {
        // printf("%s\n", token);

        int flag = 1;      // 初始标记可以输入

        for (int i = 0; i < strlen(token); i++) {
            for (int j = 0; j < strlen(brokenLetters); j++) {
                if (token[i] == brokenLetters[j]) {

                    flag = 0;  // 标记不可输入，退出循环

                    break;
                }
            }
            if (flag == 0) {
                break;
            }
        }

        if (flag == 1) {        // 如果标记为 1，则拆分后的子字符串可以输入，计数+1
```

```
            cnt++;
        }
        token = strtok(NULL, s); //  继续获取其他的子字符串
    }

    return cnt;
}
```

3.(14)最长公共前缀

```
char * longestCommonPrefix(char ** strs, int strsSize){
    int len=0;
    int i=0;
    int j=0;
    len=strlen(strs[0]);
    for(i=1;i<strsSize;++i)
    {
        if(strlen(strs[i])<len)
        {
            len=strlen(strs[i]);
        }
    }
    for(j=0;j<len;++j)
    {
        for(i=1;i<strsSize;++i)
        {
            if(strs[i][j]!=strs[i-1][j])
            {
                break;
            }
        }
        if(i!=strsSize)
        {
            break;
        }
    }
    strs[0][j]='\0';
    return strs[0];
}
```

4.(9)回文数

```
bool isPalindrome(int x){
    long int r = 0;
    int y = x;
if(x == 0)
        return true;
    if(x < 0 || x%10 == 0)
        return false;
else
{
while(x)
{
r = 10 * r + x % 10;
x = x / 10;
}
if(r == y )
return true;
else
return false;
    }
}
```

直接反转整个数；

5.（13)罗马数字转整数

```
int count = 0;
while (*s){
if (*s == 'V')          count += 5;
else if (*s == 'L')     count += 50;
else if (*s == 'D')     count += 500;
else if (*s == 'M')     count += 1000;
else if (*s == 'I')
count = (*(s + 1) == 'V' || *(s + 1) == 'X') ? count - 1 : count + 1;
else if (*s == 'X')
count = (*(s + 1) == 'L' || *(s + 1) == 'C') ? count - 10 : count + 10;
else
count = (*(s + 1) == 'D' || *(s + 1) == 'M') ? count - 100 : count + 100;
s++;
}
return count;
```

6.（20）有效的括号

```
char pairs(char a) {
    if (a == '}') return '{';
    if (a == ']') return '[';
    if (a == ')') return '(';
    return 0;
}
```

```
bool isValid(char* s) {
    int n = strlen(s);
    if (n % 2 == 1) {
        return false;
    }
    int stk[n + 1], top = 0;
    for (int i = 0; i < n; i++) {
        char ch = pairs(s[i]);
        if (ch) {
            if (top == 0 || stk[top - 1] != ch) {
                return false;
            }
            top--;
        } else {
            stk[top++] = s[i];
        }
    }
    return top == 0;
}
```

7.（21）合并两个有序列表

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if (l1 == nullptr) {
            return l2;
        } else if (l2 == nullptr) {
            return l1;
        } else if (l1->val < l2->val) {
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        } else {
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
        }
    }
};
```

8.（26）删除有序数组中的重复项

```
int removeDuplicates(int* nums, int numsSize) {
    if (numsSize == 0) {
        return 0;
    }
    int fast = 1, slow = 1;
    while (fast < numsSize) {
        if (nums[fast] != nums[fast - 1]) {
            nums[slow] = nums[fast];
            ++slow;
        }
        ++fast;
    }
    return slow;
}
```

9.（35）搜索插入位置

```
int searchInsert(int* nums, int numsSize, int target) {
    int left = 0, right = numsSize - 1, ans = numsSize;
    while (left <= right) {
        int mid = ((right - left) >> 1) + left;
        if (target <= nums[mid]) {
            ans = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return ans;
}
```

10.（58）最后一个单词的长度

```
int lengthOfLastWord(char * s){
    int len = strlen(s), lastWordLen = 0;
    if (len == 0) return 0;

    for (int i = len - 1; i >= 0; i--) {
        if (s[i] != ' ') lastWordLen++;
        if (s[i] == ' ' && lastWordLen > 0) break;
    }

    return lastWordLen;
}
```

11.（66）加一

```
int* plusOne(int* digits, int digitsSize, int* returnSize)
{
    int jw = 1;
    int i;
    for (i = digitsSize - 1; i >= 0; i--) {
        digits[i] = digits[i] + jw;
        jw = digits[i] / 10;
        digits[i] = digits[i] % 10;
    }
    *returnSize = digitsSize + jw;
    int* sum = (int*)malloc(sizeof(int) * *returnSize);
    memset(sum, 0, sizeof(int) * *returnSize);
    for (i = digitsSize - 1; i >= 0; i--) {
        sum[i + jw] = digits[i];
    }
    sum[0] += jw;

    return sum;
}
```

12.（67）二进制求和

```
void reserve(char* s) {
    int len = strlen(s);
    for (int i = 0; i < len / 2; i++) {
        char t = s[i];
        s[i] = s[len - i - 1], s[len - i - 1] = t;
    }
}

char* addBinary(char* a, char* b) {
    reserve(a);
    reserve(b);

    int len_a = strlen(a), len_b = strlen(b);
    int n = fmax(len_a, len_b), carry = 0, len = 0;
    char* ans = (char*)malloc(sizeof(char) * (n + 2));
    for (int i = 0; i < n; ++i) {
        carry += i < len_a ? (a[i] == '1') : 0;
        carry += i < len_b ? (b[i] == '1') : 0;
        ans[len++] = carry % 2 + '0';
        carry /= 2;
    }

    if (carry) {
        ans[len++] = '1';
    }
    ans[len] = '\0';
    reserve(ans);
    return ans;
}
```

中等题

1.（2）两数相加

```
struct ListNode* addTwoNumbers(struct ListNode* l1, struct ListNode* l2){
    struct ListNode* dummy = malloc(sizeof(struct ListNode));
    struct ListNode* cur = dummy;
    int t = 0;
    while(l1 || l2 || t){
        if(l1) t += l1->val,l1=l1->next;
        if(l2) t += l2->val,l2=l2->next;
        cur->next = malloc(sizeof(struct ListNode));
        cur->next->val = t%10;
        cur->next->next = NULL;
        cur = cur->next;
        t /= 10;
    }
    return dummy->next;
}
```

2.（7）整数反转

```
#define isOverLength 0

int reverse(int x){
    long lRet = 0;
    while(0 != x)
    {
        lRet = lRet * 10 + x % 10;
        x = x / 10;
    }

    if((int)lRet != lRet)
    {
        return isOverLength;
    }

    return (int)lRet;
}
```

3.（29)两数相除

```
#define INT_MAX 0X7FFFFFFF
#define INT_MIN 0X80000000

int divide(int dividend, int divisor)
{
    int result = 0;       // 存放结果值

    if(dividend == 0)    // 特殊情况判断

return 0;

    else if(dividend == INT_MIN && divisor == -1)    // 被除数为 INT_MIN 的两种特殊

情况

return INT_MAX;
    else if(dividend == INT_MIN && divisor == 1)
return INT_MIN;

    else if(dividend == INT_MIN && divisor == INT_MIN)   // 除数为 INT_MIN，就这两

种情况

        return 1;
    else if(divisor == INT_MIN)
        return 0;

    bool negative = (dividend ^ divisor) < 0;          // 判断结果是否为负数


    if(dividend == INT_MIN)             // 若被除数为 INT_MIN，先减一次，在再进行运算

    {
        dividend += abs(divisor);
        result++;
    }
    int t = abs(dividend);
    int d = abs(divisor);

    for(int i = 31; i >= 0; i--)
    {
        if((t >> i) >= d)
        {
            result += 1 << i;
            t -= d << i;
        }
    }

    if(result == INT_MIN)
        return INT_MAX;
    else
        return negative ? -result : result;
    return 0;
}
```

困难题

1.（4）寻找两个正序数组的中位数

```
double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int
nums2Size)
{
    int*newnums=(int*)malloc(sizeof(int)*(nums1Size+nums2Size));
   int i=0;
   int j=0;
   int index=0;
   while(i<=nums1Size-1&&j<=nums2Size-1)
   {
       if(nums2[j]<nums1[i])
       {
           newnums[index++]=nums2[j++];
       }
       else
       {
           newnums[index++]=nums1[i++];
       }
   }
   while(i<=nums1Size-1)
   {
       newnums[index++]=nums1[i++];
   }
   while(j<=nums2Size-1)
   {
       newnums[index++]=nums2[j++];
   }
    if((nums1Size+nums2Size)%2==0)
    {
        return ((double)newnums[(nums1Size+nums2Size)/2]+(double)newnums[(nums1Size+nums2Size)/2-1])/2;
    }
    else
    {
        return (double)newnums[(nums1Size+nums2Size)/2];
    }
}
```