时间复杂度分析

> 在problem2.pdf(或.docx)中给出两种算法的程序设计思路(可以加上对应算法的源代码)及时间复杂度分析.(3'*2)

O(N^2)的算法

```
//简单插入排序，两层循环嵌套，时间复杂度为O(N^2)
void sort1(int* array, int length)
{
    //从前往后遍历，将每个值插到左边序列中
    for(int i = 1; i < length; ++i){
        int pos = i, now = array[i];//将值存下来
        for(int j = i-1; j >= 0; --j) {//将前方大值往后移
            if(array[j] > now) {
                array[j+1] = array[j];
                pos = j;
            }
        }
        array[pos] = now;//将保存的值填入
    }
}
//寻找函数，两指针从头尾往中间走，遍历数组，时间复杂度为O(N)
int find(int* sortedArray, int length, int sum, int** resultArray)
{
    int *small = &sortedArray[0], *big = &sortedArray[length-1], hasFound
= 0;
    //和大，将大值缩小，和小，将小值放大
    while(*small <= *big){
        if(*small + *big == sum) {
            resultArray[0][hasFound] = *small;
            resultArray[1][hasFound] = *big;
            ++hasFound;
            ++small;
        }
        else if(*small + *big > sum) --big;
        else ++small;
    }
    return hasFound;
}
```

O(NlogN)的算法

```
//堆排序 时间复杂度为O(NlogN)
void sort2(int* array, int length)
{
    class priorityQueue
```

```
    { ... };//实现代码忽略
    priorityQueue myQueue(array, length);
    for(int i = 0; i < length; ++i)
        array[i] = myQueue.deQueue();
}

//寻找函数，两指针从头尾往中间走，遍历数组，时间复杂度为O(N)
int find(int* sortedArray, int length, int sum, int** resultArray)
{
    int *small = &sortedArray[0], *big = &sortedArray[length-1], hasFound
= 0;
    //和大，将大值缩小，和小，将小值放大
    while(*small <= *big){
        if(*small + *big == sum) {
            resultArray[0][hasFound] = *small;
            resultArray[1][hasFound] = *big;
            ++hasFound;
            ++small;
        }
        else if(*small + *big > sum) --big;
        else ++small;
    }
    return hasFound;
}
```