算法设计第一次作业

注：所有代码见文末。

2_7

Answer:

由已知条件得，算法可在 *O(i logi)* 时间内计算两个 *i* 次多项式的乘积。

且原式可表示为：

$$P(x) = (x-n_1)(x-n_2)\dots(x-n_d)$$

则由分治思想，可将 *d* 次多项式转换为2个 *d/2* 次多项式的乘积，则可设计算 *d* 次多项式的时间为 T(*d*)，则可表示为：

$$T(d)=\begin{cases} O(1), & d=1 \\ 2T(d/2)+O(d\,logd), & d>1 \end{cases}$$

则对此递归求解即可得：

$$T(d)=O(dlog^2 d).$$

2_9

Answer：

将此题解释即可分析出，此题希望求出函数一个解所占的整个解空间的规模大于一半，即可称为主元素。

则由此可得出结论， x 必为排好序的数组T的中位数。

易证明：

S(x) > *n /2* 时，则按照序关系排好后，必有中位数被主元素占据。

则只需要在线性时间中找出数组的中位数，再遍历查看中位数是否与主元素相同即可。则复杂度应为*O(n)*

详情代码可见本人的github网址：(https://github.com/vegetablechickenluo/design_algorithm/blob/master/work2_9/main.cpp)

输出样例：

```
请输入数组长度n： 5
请输入依次输入n个数： 2 3 4 4 3
不存在

请输入数组长度n： 3
请输入依次输入n个数： 1 1 2
存在

请输入数组长度n： 6
请输入依次输入n个数： 4 3 3 4 4 4
存在

```

此题易得出：时间复杂度为 *O(n)*

2_10

Answer:

由于此题目中数组 **T** 的元素不再具有序关系，因此不可以用 *2_9* 中线性时间寻找中位数的方法去寻找。

换用二分的思想来尝试解决该问题，假设[0 : (n - 1) /2] 部分与[(n - 1) / 2 : (n - 1)] 部分的主元素不为x，则合并后的整个数组的主元素不为x。因此，可以采用二分的思想来解决该问题。

则算法的递归式表示为：

$$T(n) = 2T(n/2) + O(n)$$

由《算法导论》中的主定理，易求出算法复杂度下界为：*O(n logn)*

详情代码可见本人的github网址——*由于此题数组设定为无序，则不会采用比较的方式*：

（https://github.com/vegetablechickenluo/design_algorithm/blob/master/work2_10/main.cpp）

输出样例：

```
请输入数组长度n： 5
请输入依次输入n个数： 2 3 4 4 3
不存在

请输入数组长度n： 3
请输入依次输入n个数： 1 1 2
存在,主元素是1

请输入数组长度n： 6
请输入依次输入n个数： 4 3 3 4 4 4
存在，主元素是4
```

算法优化：

不妨用数组来存放且将每一个元素取出且放入栈中，每当一个元素与栈顶元素相同时，则将两个元素同时消去。若栈中仍然存有元素，则最终通过遍历的方式来对数组中该元素进行计数，并最终与 $n / 2$比较，若栈中不存在元素或该元素未超过$n / 2$ 次，则认为数组中不存在主元素。则 $O(n)$ 为优化后的时间复杂度下界。

2_28

Answer：

注：已排好序默认为升序排列；

题目分析：

题目要求设计出 $O (log n)$ 时间复杂度的算法，则首先则需要对题目中的两个数组利用分治思想进行处理。

算法表示：

I. 分别寻找长度为 n 的 X、Y两个数组的中位数，由于两数组已排好序，则查找的时间复杂度为 $O(1)$；

II. 然后将 X、Y 两数组的中位数设为 x1,y1, 比较 x1 与 y1 的大小关系，时间复杂度为 $O(1)$；

III. **important**：依据中位数，将 X、Y 两个数组依次分为 $X1, X2$ 与 $Y1, Y2$ 四个数组，则**定有中位数小的数组的左半部分，与中位数大的数组的右半部分中不包含中位数**。因此，则可以将其砍去，以实现简单的问题总规模减半的目的。

复杂度分析：

$$T(n) = T(n/2) + O(1)$$

则易得出，算法时间复杂度为 *O(log n)*；

详情代码可见本人的github网址：

（https://github.com/vegetablechickenluo/design_algorithm/blob/master/work2_28/main.cpp）

输出样例：

```
请输入数组规模： 10
请依次输入a数组的数字： 1 2 3 4 5 6 7 8 9 10
请依次输入b数组的数字： 2 3 4 5 6 7 8 9 10 11
中位数为： 6

请输入数组规模： 10
请依次输入a数组的数字： 1 2 3 4 6 6 7 8 9 10
请依次输入b数组的数字： 2 3 4 5 7 7 8 9 10 11
中位数为： 6.5

请输入数组规模： 3
请依次输入a数组的数字： 1 2 3
请依次输入b数组的数字： 2 3 4
中位数为： 2.5
```

作业代码

2_9

```
#include <iostream>
#include <algorithm>
#define N 10000
int a[N];
using namespace std;

int partition(int p, int r ,int x){        //以x为基准划分元素，小于x在左，大于在右
    int m = 0;
    for(int i = p, j = r; i < j; i++){
        if(a[i] > x){
            while(i < j && a[j] > x)
```

```
                j--;
            if(i != j){
                int db = a[i];
                a[i] = a[j];
                a[j] = db;
                j--;
            }
        }
        m = i;
    }
    return m - 1;
}

void Sort(int m, int n){
    for (int i = m; i <= n; i++) {
        int x = i;
        for (int j = i + 1; j <= n; j++)
            if (a[x] > a[j])
                x = j;
        if (x != i) {
            int num = a[x];
            a[x] = a[i];
            a[i] = num;
        }
    }
}

int select(int left, int right, int k){
    if(right - left < 5){
        Sort(left, right);
        return a[left + k - 1];
    }
    for (int i = 0; i <= (right - left - 4) / 5; i++){
        Sort(left + 5 * i, left + 5 * i + 4);
        int jk = a[left + 5 * i + 2];
        a[left + 5 * i + 2] = a[left + i];
        a[left + i] = jk;
    }
    int x = select(left, left + (right - left - 4) / 5,
(right - left - 4) / 10);
    int local = partition(left, right, x), j = local - left +
1;
    if (k <= j) return select(left, local, k);
    else return select(local + 1, right, k - j);
}

```

```cpp
int main() {
    cout << "请输入数组长度n: ";
    int n = 0;
    cin >> n;
    cout << "请输入依次输入n个数: ";
    for(int i = 0; i < n; i++)  cin >> a[i];
    int finding = select(0, n - 1, n / 2);
    int count = 0;
    for(int i = 0; i < n; i++){
        //cout << a[i] << " ";
        if(a[i] == finding)
            count++;
    }
    if(count > n / 2)
        cout << "存在";
    else
        cout << "不存在";
    return 0;
}
```

2_10

```cpp
#include <iostream>
using namespace std;

int b[100000];

int Element_mid(int left, int right){
    int count1,count2 = 0;
    if(left == right) return b[left];
    int middle = (left + right) / 2;
    int Ele_left = Element_mid(left, middle);              //二分寻找主元素
    int Ele_right = Element_mid(middle + 1, right);    //二分寻找主元素
    for(int i = left; i <= right; i++){
        if(b[i] == Ele_left)    count1++;                  //计数
        if(b[i] == Ele_right)   count2++;                  //计数
    }
    if(count1 > (right - left) / 2) return Ele_left;       //判断是否为主元素
    if(count2 > (right - left) / 2) return Ele_right;      //判断是否为主元素
    return 0xffffff;
```

```cpp
}

int main(){
    cout << "请输入数组大小n：";
    int n = 0;
    cin >> n;
    cout << "请依次输入n个数字：";
    for(int i = 0; i < n; i++) cin >> b[i];
    int main_Ele = Element_mid(0, n - 1);
    if(main_Ele == 0xffffff)
        cout << "不存在";
    else
        cout << "存在，主元素是" << main_Ele;
    return 0;
}
```

2_28

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

double find_middle(vector<double> a,vector<double> b){
    int n = a.size();
    if(n % 2 == 1){
        double a1 = a[(n - 1) / 2];
        double b1 = b[(n - 1) / 2];
        if(a1 == b1)    return a1;
        if(a1 != b1 && n == 1)  return (a1 + b1) / 2;
        else if(a1 > b1){
            a.erase(a.begin() + (n + 1) / 2, a.end());
            b.erase(b.begin(), b.begin() + (n - 1) / 2);
            return find_middle(a, b);
        }
        else if(a1 < b1){
            b.erase(b.begin() + (n + 1) / 2, b.end());
            a.erase(a.begin(), a.begin() + (n - 1) / 2);
            return find_middle(a, b);
        }
    }
    else{
        double a1 = a[n / 2];
        double b1 = b[n / 2 - 1];
        if(a1 == b1)    return a1;
        if(n == 2 && a1 > b1){
```

```
            double m[4] = {0};
            m[0] = a[0];
            m[1] = a[1];
            m[2] = b[0];
            m[3] = b[1];
            sort(m, m+3);
            return (m[1] + m[2]) / 2;
        }
        if(a1 > b1){
            a.erase(a.begin() + n / 2 + 1, a.end());
            b.erase(b.begin(), b.begin() + n / 2 - 1);
            return find_middle(a, b);
        }
        else if(a1 < b1){
            b.erase(b.begin() + n / 2 + 1, b.end());
            a.erase(a.begin(), a.begin() + n / 2 - 1);
            return find_middle(a, b);
        }
    }
}
int main() {
    int n = 0;
    cout << "请输入数组规模：";
    cin >> n;
    vector <double> a(n);
    vector <double> b(n);
    cout << "请依次输入a数组的数字：";
    for(int i = 0; i < n ; i++)
        cin >> a[i];
    cout << "请依次输入b数组的数字：";
    for(int i = 0; i < n ; i++)
        cin >> b[i];
    cout << "中位数为：" << find_middle(a, b) << endl;
    return 0;
}
```