西安交通大学实验报告

课程名称：算法设计与分析　　实 验 名 称：分治与递归

学　　院：电子与信息学部　　实 验 日 期：2023 年 10 月 13 日

姓　　名：林圣翔　　班级：计试 2201　　学号：2223312202

诚信承诺：我保证本实验报告中的程序和本实验报告是我自己编写，没有抄袭。

一、实验目的

1.深入了解分治算法的内容及应用

2.设计在线性时间复杂度下寻找 n 个元素带权中位数算法

二、实验环境

系统类型：64 位操作系统, 基于 x64 的处理器

处理器：12th Gen Intel(R) Core(TM) i7-1260P　　2.10 GHz

软件：Dev-C++ 5.9.2 TDM-GCC 4.8.1 64-bit Debug

三、问题描述

设有 n 个互不相同的元素，$x_1,x_2,\cdots,x_n$，每个元素 X 带有一个权值 w_i，且 $\sum_{i=1}^{n} w_i = 1$。若元素 X_k 满足$\sum_{x_i<x_k} w_i \le \frac{1}{2}$ 且$\sum_{x_i>x_k} w_i \le \frac{1}{2}$，则称元素$x_k$ 为$x_1,x_2,\cdots,x_n$ 的带权中位数。请编写一个算法， 能够在最坏情况下用O(n)时间找n 个元素的带权中位数。

四、问题分析

由题意可得，我们需要在线性时间复杂度的情况下找到 n 个元素的带权中位值。我们可以先将题目简化，先不考虑带权的情况，即从在线性时间复杂度的情况下找到 n 个元素的中位数。针对这个简化版的问题，我们可以先将整个数组划分为若干小组，再对每个小组进行排序，选择出中位数，再在已经选出的 N 个中位数中找出他们的中位数，这个最终的中位数就是十分接近中间的数字，以这个数为基准进行划分……经过严密的推导，可以保证在最坏情况下，该算法依然以线性时间运行。然后我们再考虑带全值的情况，不难发现，我们只要对前面解决简化版问题的思路中“对每个小组进行排序，选择出中位数”稍作修改，排序按每个数据的元素数值为关键字，中位数改为题目中的 x_k 所需满足的条件（即元素 xk 满足$\sum_{x_i<x_k} w_i \le \frac{1}{2}$ 且$\sum_{x_i>x_k} w_i \le \frac{1}{2}$），然后其他部分作一些简单的修改，即可得出符合题目条件的算法。

五、算法设计

对于这道题，我们主要通过两个函数实现：一个是主函数 main(),负责数据的输入以及调用 WeightMedian(length,num, weight,index)函数；另一个即为自定义无返回值函数 WeightMedian(length,num, weight,index)，负责在 O(n)时间找 n 个元素的带权中位数算法思想的实现。在 main()函数中，定义两个 vector 类型的动态数组，其中 num 存储每

个元素的数值，weight 存储每个元素的权重，整数变量 n 为数据个数。在 WeightMedian(length,num, weight,index)函数中，length 表示选定数据范围大小，num 和 weight 即为输入的 weigh 和 num 的复制传递，index 即为当前处理区间第一位数据的位置，我们可以从 index 和 length 这两个量得出每次的数据范围，即 index 到 index+length-1。我们在动态数组 weight 的后面再增加一个数值 weigh[num.size()]，初始为 0.5，具体作用后文会有分析。然后我们可以注意到，当 length<5 时，这个时候的运算是常数级的，我们直接可以暴力运算，即进行冒泡排序，排序按每个数据的元素数值为关键字，然后从小到大，根据题意寻找题解，若满足条件，存在即直接输出，程序结束，如果不存在，说明答案不在这一组数据中，那进行下一组的寻找。接下来就是之前问题分析中算法的主要实现。我们将目前的区间划分为每组 5 个元素的子区间（最后一组可能不够 5 个），然后通过冒泡排序找出每组中的中位数，然后再从这些中位数中找出中位数作为划分的依据（比该数值小的放到前一个数组，比该数值大的放到后一个数值），即原数组变成了两个子数组。对前一个子数组求出所有权值之和，如果权值之和>weigh[num.size()]，说明所求的数据在前一个数组内，故对前一个数组范围进行递归；如果权值之和≤weigh[num.size()]，则说明所求的数据在后一个数组内，也就是说后面数组中权值和还需要达到 weigh[num.size()]-前一个数组的权值之和，故对 weigh[num.size()]重新赋值后，对后一个数组范围内进行递归。故通过上述步骤，我们就可以通过不断的递归，不断缩小数据范围，最后得到答案。

六、算法实现

```cpp
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;
void WeightMedian(int length,vector<int>num, vector<double>weight,int index)
{
    int st=index,ed=index+length-1,nn=num.size();
    if(length==nn) weight.push_back(0.5);
    if(length<5)
    {
        for(int i=1; i<=length-1; i++)
        {
            for(int j=st; j<=ed-i; j++)
            {
                if(num[j]>num[j+1])
                {
                    swap(num[j],num[j+1]);
                    swap(weight[j],weight[j+1]);
                }
            }
        }
        double sum=0;
        for(int i=st; i<=ed; i++)
        {
            sum=sum+weight[i];
            if(sum>weight[nn]) cout<<num[i],exit(0);
        }
    }
    for(int ii=0; ii<=(ed-st-4)/5; ii++)
    {
        int pl=st+5*ii;
        int pr=pl+4;
        for(int i=1; i<=4; i++)
```

```
        {
            for(int j=pl; j<=pr-1; j++)
            {
                if(num[j]>num[j+1])
                {
                    swap(num[j],num[j+1]);
                    swap(weight[j],weight[j+1]);
                }
            }
        }
        swap(num[st+ii],num[pl+2]);
        swap(weight[st+ii],weight[pl+2]);
    }
    for(int i=1; i<=(ed-st-4)/5; i++)
    {
        for(int j=st; j<=st+(ed-st-4)/5-i; j++)
        {
            if(num[j]>num[j+1])
            {
                swap(num[j],num[j+1]);
                swap(weight[j],weight[j+1]);
            }
        }
    }
    int xx=num[st+(ed-st+6)/10];
    int l=st,r=ed;
    while(l<r)
    {
        if(num[l]<xx) l++;
        if(num[r]>xx) r--;
        swap(num[l],num[r]);
        swap(weight[l],weight[r]);
    }
    int pos=l;
    double sum=0;
    for(int i=st; i<=pos; i++)  sum=sum+weight[i];
    if(sum>weight[nn]) WeightMedian(pos-st+1,num,weight,st);
    else weight[nn]=weight[nn]-sum,WeightMedian(ed-pos,num,weight,pos+1);
}
int main()
{
    int n,x;
    double y;
    vector<int> num;
    vector<double> weight;
    cin>>n;
    for(int i=0; i<=n-1; i++) cin>>x,num.push_back(x);
    for(int i=0; i<=n-1; i++) cin>>y,weight.push_back(y);
    WeightMedian(n,num,weight,0);
    return 0;
}
```

七、运行结果

首先我们可以制作一个简单的随机数据生成器，并且用 O(nlogn)复杂度的算法按照题意求出答案，然后再将生成的数据输入之前 O(n)的算法中，将两个算法得到的答案相比较，一样即为正确。

受版面的限制以及记录的方便，这里随机数据生成 n 较小，但只要稍微修改下面程序的参数，即可生成想要数据规模的数据。

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const ll N=1e6+10;
struct node{
    ll num;
    double weight;
};
node a[N];
```

```cpp
ll n;
double sum;
bool cmp(node x,node y)
{
    return x.num<y.num;
}
int main()
{
     ios::sync_with_stdio(false);
     cin.tie(0);
     srand(time(0));
     n=rand()%100;
     for(ll i=1;i<=n;i++) a[i].num=rand()%1000000;
     while(1)
     {
        for(ll i=1;i<=n-1;i++) a[i].weight=(double)(rand()%100000)/100000,a[n].weight=1-a[i].weight;
        if(a[n].weight>0) break;
     }
     cout<<n<<"\n";
     for(ll i=1;i<=n;i++) cout<<a[i].num<<"  ";
     cout<<"\n";
     for(ll i=1;i<=n;i++) cout<<a[i].weight<<"  ";
     cout<<"\n";
     sort(a+1,a+n+1,cmp);
     for(ll i=1;i<=n;i++)
     {
        sum=sum+a[i].weight;
        if(sum>=0.5)
        {
            cout<<a[i].num;
            break;
        }
     }
     return 0;
}
```

以下是部分数据输入与输出

第一组

生成数据程序输出：

```
11
27794  2952  28194  7253  4712  26136  24960  850  15212  27997  28668
0.26943  0.23236  0.03128  0.19456  0.07594  0.03414  0.02373  0.0251  0.27557  0.06784  0.93216
7253
```

O(n)算法程序输入和输出：

```
11
27794  2952  28194  7253  4712  26136  24960  850  15212  27997  28668
0.26943  0.23236  0.03128  0.19456  0.07594  0.03414  0.02373  0.0251  0.27557  0.06784  0.93216
7253
```

答案为 7253，程序运行正确

第二组

生成数据程序输出：

```
86
21961  24284  28081  32183  10251  10763  12421  2397  17651  24326  3258  10371  23659  2332  20281  26626  11577  3175
9  3212  6858  19679  3175  18406  29907  9919  8716  2915  21079  30339  22804  21904  32575  16828  9434  46  4827  53
21  10576  1075  13269  1787  16022  16146  22632  18935  21195  7484  22221  18523  7715  19943  23030  6052  23865  25
614  3711  16719  1447  2233  7138  10046  10453  8830  15850  1886  30990  5704  4714  25223  9433  1367  28462  19107
 31878  11400  21554  23571  5639  18647  19986  26190  25345  9506  23746  1995  27939
0.13616  0.12927  0.16674  0.11447  0.25073  0.24325  0.17262  0.11638  0.27399  0.15107  0.32245  0.30113  0.0039  0.10
188  0.12043  0.25421  0.3276  0.30543  0.31795  0.29695  0.25509  0.04169  0.21896  0.10872  0.27409  0.08889  0.10331
 0.01746  0.32197  0.00149  0.09502  0.17612  0.16722  0.28347  0.07296  0.21603  0.3266  0.00554  0.14298  0.16546  0.1
6948  0.23922  0.04298  0.30906  0.18288  0.25193  0.27677  0.12117  0.13343  0.11296  0.07155  0.17448  0.09967  0.2587
7  0.19741  0.27591  0.04338  0.19882  0.17354  0.02494  0.30553  0.0175  0.19299  0.28893  0.00639  0.29979  0.23987  0
.32211  0.32297  0.20408  0.17739  0.16383  0.12044  0.12319  0.16165  0.09346  0.08591  0.13737  0.24455  0.22425  0.21
971  0.04489  0.15914  0.20522  0.27888  0.72112
1447
```

O(n)算法程序输入和输出：

```
86
21961  24284  28081  32183  10251  10763  12421  2397  17651  24326  3258  10371  23659  2332  20281  26626  11577  3175
9  3212  6858  19679  3175  18406  29907  9919  8716  2915  21079  30339  22804  21904  32575  16828  9434  46  4827  53
21  10576  1075  13269  1787  16022  16146  22632  18935  21195  7484  22221  18523  7715  19943  23030  6052  23865  25
614  3711  16719  1447  2233  7138  10046  10453  8830  15850  1886  30990  5704  4714  25223  9433  1367  28462  19107
 31878  11400  21554  23571  5639  18647  19986  26190  25345  9506  23746  1995  27939
0.13616  0.12927  0.16674  0.11447  0.25073  0.24325  0.17262  0.11638  0.27399  0.15107  0.32245  0.30113  0.0039  0.10
188  0.12043  0.25421  0.3276  0.30543  0.31795  0.29695  0.25509  0.04169  0.21896  0.10872  0.27409  0.08889  0.10331
 0.01746  0.32197  0.00149  0.09502  0.17612  0.16722  0.28347  0.07296  0.21603  0.3266  0.00554  0.14298  0.16546  0.1
6948  0.23922  0.04298  0.30906  0.18288  0.25193  0.27677  0.12117  0.13343  0.11296  0.07155  0.17448  0.09967  0.2587
7  0.19741  0.27591  0.04338  0.19882  0.17354  0.02494  0.30553  0.0175  0.19299  0.28893  0.00639  0.29979  0.23987  0
.32211  0.32297  0.20408  0.17739  0.16383  0.12044  0.12319  0.16165  0.09346  0.08591  0.13737  0.24455  0.22425  0.21
971  0.04489  0.15914  0.20522  0.27888  0.72112
1447
```

答案为 1447，程序运行正确

第三组

生成数据程序输出：

```
49
29280  27653  461  24954  9168  11112  4294  29806  13726  30538  1408  30810  16841  20533  12977  14382  24150  20404
 30984  18926  24839  27913  18705  6406  4203  31261  31832  27359  26135  27837  18897  4527  17970  18434  22903  167
89  20535  14865  15156  15333  11510  16901  17665  17601  32220  29226  7246  5865  20853
0.08812  0.28324  0.18899  0.16687  0.09999  0.07561  0.24063  0.07265  0.2448  0.24996  0.25477  0.17547  0.12843  0.10
529  0.284  0.07904  0.18094  0.13655  0.20558  0.16129  0.05838  0.15272  0.1652  0.12044  0.09673  0.03497  0.3096  0.
30933  0.29868  0.03203  0.26905  0.21619  0.01734  0.27546  0.32491  0.08191  0.02652  0.01644  0.13558  0.01176  0.311
34  0.19899  0.23303  0.27818  0.18394  0.04078  0.19705  0.12144  0.87856
4203
```

O(n)算法程序输入和输出：

```
49
29280  27653  461  24954  9168  11112  4294  29806  13726  30538  1408  30810  16841  20533  12977  14382  24150  20404
 30984  18926  24839  27913  18705  6406  4203  31261  31832  27359  26135  27837  18897  4527  17970  18434  22903  167
89  20535  14865  15156  15333  11510  16901  17665  17601  32220  29226  7246  5865  20853
0.08812  0.28324  0.18899  0.16687  0.09999  0.07561  0.24063  0.07265  0.2448  0.24996  0.25477  0.17547  0.12843  0.10
529  0.284  0.07904  0.18094  0.13655  0.20558  0.16129  0.05838  0.15272  0.1652  0.12044  0.09673  0.03497  0.3096  0.
30933  0.29868  0.03203  0.26905  0.21619  0.01734  0.27546  0.32491  0.08191  0.02652  0.01644  0.13558  0.01176  0.311
34  0.19899  0.23303  0.27818  0.18394  0.04078  0.19705  0.12144  0.87856
4203
```

答案为 4203，程序运行正确

……