# 分治与递归 - 实验1报告

**李雨轩**
2204112913
计算机2205

## 一. 题目描述：Weighted Median

For $n$ **distinct** ordered elements $x_1, x_2, \cdots, x_n$ with positive weights $w_1, w_2, \cdots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the weighted median is the element $x_k$ satisfying:

$$\sum_{i=1}^{k-1} w_i \leq \frac{1}{2}$$
$$\sum_{i=k+1}^{n} w_i \leq \frac{1}{2}$$

The Question is that Using at most $O(n)$'s time in the worst case, to **Find the Weighted Median** in an unordered array which length is $n$.

## 二. 问题分析

题目要求寻找带权中位数，根据带权中位数的定义，一个很朴素的做法是遍历整个数组，对每个数进行是否带权中位数的判断，显然对于任何一个数的判断需要$O(n)$的时间复杂度，因此该方法的整体时间复杂度是$O(n^2)$。

考虑到有序数组能带来更多的信息，我们可以先花费$O(n \log n)$的时间对数组进行排序处理。进而可以借助前缀和的思想从小至大遍历数组，这样在计算每个数的带权中位数时只需要根据前一个元素的左右权值和进行加减即可获取，此时对于每个元素是否带权中位数的判断只需要$O(1)$的时间。因此总体的时间复杂度是$O(n \log n)$，也即对时间的主要影响因素来自对数组进行排序。

由于问题要求在$O(n)$的时间复杂度内，从一个无序数组中完成带权中位数的寻找。因此先对数组进行排序这一举措显然是不符合要求的。但是上面借助上一次对左右权值计算得到下一次权值计算的思想可以保留。因此联系 Linear Time Median Finding 的思想，可以在$O(n)$的时间内完成带权中位数的寻找。

## 三. 算法设计

算法的核心设计思想是通过递归地将问题分解为规模更小的子问题，并利用中位数的性质和权重的信息来不断缩小搜索范围，最终在线性时间内找到权重中位数。

让我们逐步解释这个算法的思想：

1. **基本情况处理**：当数组长度为1时，直接输出该元素即可，因为它就是唯一的中位数。
2. **初始化**：当 index 为0时，表示第一次递归调用，此时初始化左权重为0，右权重为1，并调用 WeightMedian 函数，将index设为数组长度，即表示开始对整个数组进行操作。
3. **寻找中位数**：在每次递归调用中，通过 nth_element 函数找到当前数组的中位数 midnum_value 。nth_element 函数的时间复杂度为O(n)，其中n为当前数组的长度。
4. **计算权重**：然后根据 midnum_value 将数组分为比 midnum_value 小和比 midnum_value 大的两部分，并计算这两部分的权重之和。通过遍历数组一次，可以在O(n)时间内完成此操作。
5. **更新权重**：根据权重之和与0.5的大小关系，更新左右权重值。如果右权重大于0.5，说明中位数在右边，此时减去比 midnum_value 小的部分的权重，同时更新左右权重值；如果左权重大于0.5，则中位数在左边，类似地进行更新。
6. **递归调用**：根据更新后的权重值，选择适当的方向进行递归调用。如果右权重大于0.5，则对比 midnum_value 大的部分进行递归调用；如果左权重大于0.5，则对比 midnum_value 小的部分进行递归调用。递归调用的时间复杂度可以通过递归树的分析得到，每次递归至少减少一半的元素，因此总的时间复杂度为O(n)。
7. **输出结果**：当左右权重都不大于0.5时，说明找到了权重中位数，直接输出 midnum_value 。

   这样，通过不断地将问题分解为规模更小的子问题，并根据中位数的性质和权重的信息来缩小搜索范围，最终在$O(n)$时间内找到了权重中位数。

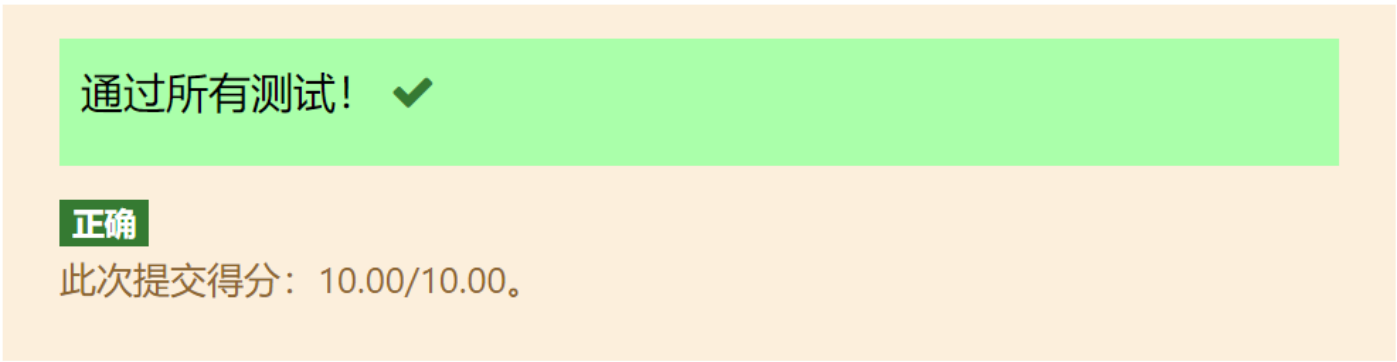
## 四. 算法实现：源代码

```
void WeightMedian(int length, vector<int> num, vector<double> weight, int index)
{
  static double left_weight, right_weight;

  if (length == 1)
  {
    cout << num[0] << endl;
    return;
  }
  else if (index == 0)
  {
    left_weight = 0;
    right_weight = 1;
    WeightMedian(length, num, weight, length);
  }
  else

  {
```

```
// O(n): get mid number of this array;
int sub_length = (length - 1) >> 1; // length / 2, And if length is Even: length / 2 = (length - 1) / 2

vector<int> temp_num = num;
nth_element(temp_num.begin(), temp_num.begin() + sub_length, temp_num.end());
int midnum_value = temp_num[sub_length];

int index = 0;
while (num[index] != midnum_value)
    index++;
double midnum_weight = weight[index];

// O(n):
double less_weights = 0, greater_weight = 0;
vector<int> less_nums = {}, greater_nums = {};
vector<double> less_nums_weight = {}, greater_nums_weight = {};

for (int i = 0; i < length; i++)
{
    if (num[i] < midnum_value)
    {
        less_nums.push_back(num[i]);
        less_nums_weight.push_back(weight[i]);
        less_weights += weight[i];
    }
    else if (num[i] > midnum_value)
    {
        greater_nums.push_back(num[i]);
        greater_nums_weight.push_back(weight[i]);
        greater_weight += weight[i];
    }
}

// update new weight in sub_length!
if (right_weight > 0.5)
{
    right_weight = right_weight - less_weights - midnum_weight;
    left_weight = 1 - right_weight - midnum_weight;
}
else if (left_weight > 0.5)
{
    left_weight = left_weight - greater_weight - midnum_weight;
    right_weight = 1 - left_weight - midnum_weight;
}

// choose a direction
if (right_weight > 0.5)
    WeightMedian(length - sub_length - 1, greater_nums, greater_nums_weight, 1);

else if (left_weight > 0.5)
    WeightMedian(sub_length, less_nums, less_nums_weight, 1);
```

```
        else // left_weight <= 0.5 && right_weight <= 0.5
            cout << midnum_value << endl;
    }

    return;
}
```

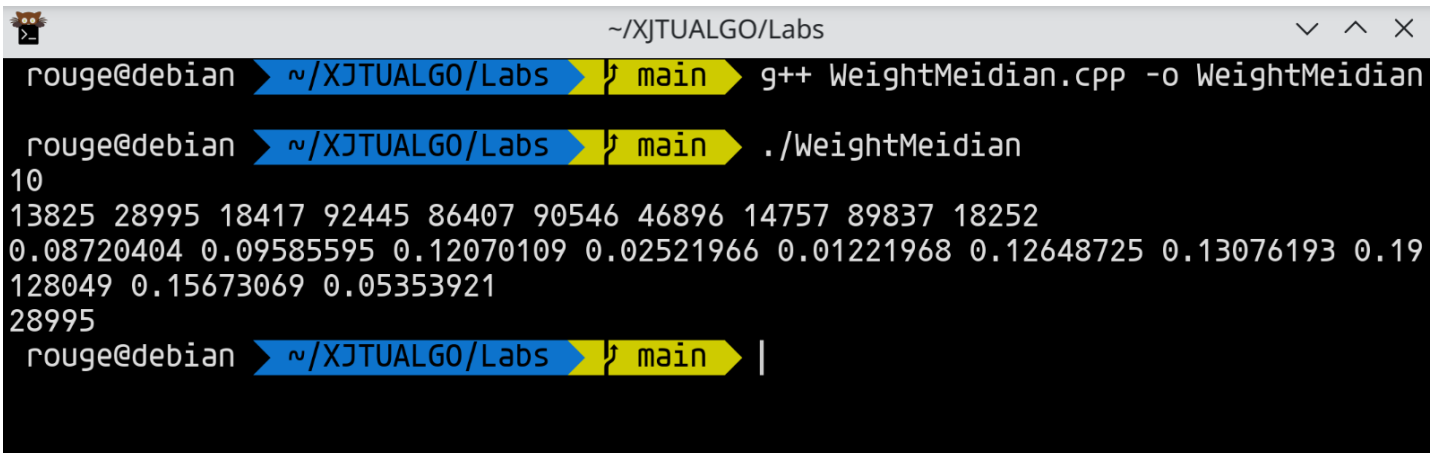
## 五. 运行结果：Moodle平台测试用例结果




您先前试答的概要

| 试答 | 状态 | 成绩 / 10.00 | 回顾 |
| --- | --- | --- | --- |
| 1 | 已结束<br>已提交 2024年03月25日 星期一 17:29 | 10.00 | 回顾 |

最高分: 10.00 / 10.00。




## 六. 实验过程说明

在实验的进行过程中，首先对问题进行了分析，从最朴素的枚举算法开始，进而从排序方法开始优化，最后得到最终的$O(n)$时间复杂度的算法。然后通过 cpp 代码实现，寻找样例进行了验证，最终得到结果。