管静

寻找数组中第K大的数，时间复杂度O(N)_salmonwilliam的博客-CSDN博客

blog.csdn.net 已更新2021年3月30日

寻找数组中第K大的数，时间复杂度O(N)

转载 HeisenbergWDG 2020-06-13 11:35:24 920 收藏 5 版权

分类专栏： 排序

给定一个数组A，要求找到数组A中第K大的数字。对于这个问题，解决方案有不少，此处我只给出三种：

有道题目寻找第K大，我用这3种方法都做了一遍。

方法1：

对数组A进行排序，然后遍历一遍就可以找到第K大的数字。该方法的时间复杂度为O(N*logN)

方法2：

利用简单选择排序法的思想，每次通过比较选出最大的数字来，比较上K次就能找出第K大的数字来。该方法的时间复杂度为O(N*K)，最坏情况下为O(N^2)。

方法3：

这种方法是本文谈论的重点，可以利用快排的思想，首先快排每次执行都能确定一个元素的最终的位置，如果这个位置是n-k(其中n是数组A的长度)的话，那么就相当于找到了第K大的元素。记住**快排**每次**确定**一个**元素**的**位置**后，左边都是比**<=**它，右边**>=**它，很关键的**前提**。设经过一次快速排序后确定的元素位置是m的话，如果m > n - k大的话，那么第K大的数字一定在A[0]~A[m - 1]之间；如果m < n - k的话，那么第K大的数字一定在A[m+1]~A[n - 1]之间。整个过程可以通过递归实现，具体代码如下：

```
#include<iostream>
#include<cassert>
#include<vector>
#include<stack>
#include<cstdio>
```

```
#include<unordered_map>
#include<queue>
#include<cstring>
#include<cstdlib>
#include<cmath>
#include<algorithm>
using namespace std;

int Partition(int* arr,int low ,int high)
{
    int temp = arr[low];
    while(low < high)
    {
        while(low < high && arr[high] >= temp)
            high--;
        arr[low] = arr[high];
        while(low < high && arr[low] <= temp)
            low++;
        arr[high] = arr[low];
    }
    arr[low] = temp;//确定参考元素的位置
    return low;
}
int KthElement(int * arr,int low, int high,int n ,int k)
{
    if(arr == nullptr || low >= high || k > n)//边界条件和特殊输入的处理
        return 0;
    int pos = Partition(arr,low,high);
    while(pos != n  - k)
    {
        if(pos > n - k)
        {
            high = pos - 1;
            pos = Partition(arr,low,high);
        }
        if(pos < n - k)
        {
            low = pos + 1;
            pos = Partition(arr,low,high);
        }
    }
    return arr[pos];

}

int main()
{

   int a[]={1,5,5,7,88,11};
```

```
    cout<<KthElement(a,0,5,6,2);

}
```

注意：

1.第K大的数字在数组中对应的位置为n-k(按照升序排序的话)。

2.该算法的时间复杂度整体上为O(N)。

3.需要注意的是：这种方法会改变数组中元素的顺序，即会改变数组本身。

4.如果要求第K小的数字的话，只需把n-k换成k-1即可(升序排序)。

==

接下来，我们仔细分析一下方法3的时间复杂度，其实方法3在《算法导论》第九章有着比较详细的描述，但《算法导论》说的是**期望为线性时间的选择算法**，即该算法的时间复杂度在平均情况下或者一般情况下为O(n)；因为此处利用的快排的思想，而快排的时间

复杂度在一般情况下为O(N*logN)，但在最坏的情况下(即整个数组原本就是有序的情况)时间复杂度为O(N^2)。所以说对于方法3，《算导》最后给定结果是这样的：**平均时间复杂度为O(N)，最坏情况下的时间复杂度为O(N^2)。**

但是，此处的“平均”同快排一样，是适用于绝大数的情况的。所以我们通常说该算法的时间复杂度为O(N)。

1.我们要搞清楚一点，快排是对参考元素两边都进行递归，而我们的方法3只考虑参考元素的一边，即只对一边进行递归。

2.我们可以粗略的估计下(具体计算还是参考《算导》)，在一般情况下方法3的时间复杂度计算公式，假设我们的数据足够的随机，每次划分都在数据序列的中间位置，根据条件1，那么第一次划分我们需要遍历约n个数，第二次需要遍历约n/2个数，...，这样递归下去，最后：

$$n+\frac{n}{2}+\frac{n}{2^2}+\frac{n}{2^3}+\ldots+\frac{n}{2^m}=n.(1+\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\ldots+\frac{1}{2^m})$$

当m趋于无穷大时，该式子收敛于2n，故可以认为其期望时间复杂度为**O(N)**.快排是有栈的深度，所以是NlogN

原文链接：https://www.cnblogs.com/wangkundentisy/p/8810077.html