1. 数组最大值和最小值问题

2.给定数组A，查找数组中第i小的数据

3. 代码下载

**1. 最大值和最小值问题**

1.1 给定一个数组A，如何求得数组A中的最大元素和最小元素？最直接的算法显然就是通过遍历数组实现，那么现在的问题是这个算法是否是最优的呢？类比比赛淘汰机制的话，可以看出上面的算法在比较次数上是最优的。简单的实现：

 int maxElementInArray(int\* arr, int length)

{

    int max = arr[0];

    for(int i = 1; i < length; ++i)

    {

        if (arr[i] > max)

            max = arr[i];

    }

    return max;

}

1.2 下面一个问题是上面问题的拓展，给定一个数组，如何通过最小的比较次数得该数组的最大值和最小值？这里有比较详细的代码实现。<http://www.cnblogs.com/xuqiang/archive/2010/12/07/1953368.html>

**2. 给定数组A如何查找数组的第i小元素**

 2.1朴素的算法，首先对数组A进行排序，然后查找第i小的元素。算法实现比较简单，略去。

2.2 上面的过程中。其中做了很多无用功，我们没有必要对数组进行排序，而仅仅需要找到第i小的元素，显然算法是存在优化的空间的。联想快速排序算法，通过分割的形式，将数组A分割成两部分，那么pivot元素在数组中的排序已经确定，显然通过递归能够比较快实现上面的算法。

void swap(int\* a, int\* b)

{

    int tmp = \*a;

    \*a = \*b;

    \*b = tmp;

}

// [start, end]闭区间

int quick\_partition(int\* arr,

    int start,

    int end)

{

    // 这里的pivot总是数组的最后一个元素

    int pivot = arr[end];

    // 开始分割

    int i = -1;

    int j = start;

    while( j < end )

    {

        // 递增顺序

        if ( arr[j] < pivot )

        {

            ++i;

            swap(&arr[j], &arr[i]);

        }

        j++;

    }

    // 找到pivot位置

    ++i;

    swap(&arr[i], &arr[end]);

    return i;

}

// index表示第i小的i

int my\_random\_select(int\* arr, int start, int end,

    int index)

{

    // 递归结束条件

    // 只有一个元素，返回

    if( start == end )

        return arr[start];

    int q = quick\_partition(arr, start, end);

    int k = q - start + 1;

    if (k == index)

        return arr[q];

    else if ( index < k )

        return my\_random\_select(arr, start, (q - 1), index);

    else    // index  >= k

        return my\_random\_select(arr, q + 1, end, ( index - k ));

}

int main()

{

    int arr[5] = { 1, 2, 10, 4, 5 };

    cout << my\_random\_select(arr, 0, 4, 5) << endl;

    return 0;

}

**3. 代码下载**

[/Files/xuqiang/algorithm/MaxArrayElement.rar](http://files.cnblogs.com/xuqiang/algorithm/MaxArrayElement.rar)