算法设计与分析第三次作业

姓名	学号
付一豪	SA20225161

EX6.5-7

HEAP-DELETE(A，i)操作将结点 i 中的项从堆 A 中删去。对含 n 个元素的最大堆，请给出时间为 $O(\lg n)$的 HEAP-DELETE 的实现。

答： 关键伪代码如下

将A[i] 与 A[heap-size[A]] 交换，将heap-size[A]减一；

如果交换后A[i] <= A[PARENT (i)] 使用MAX-HEAPIFY 向下调整

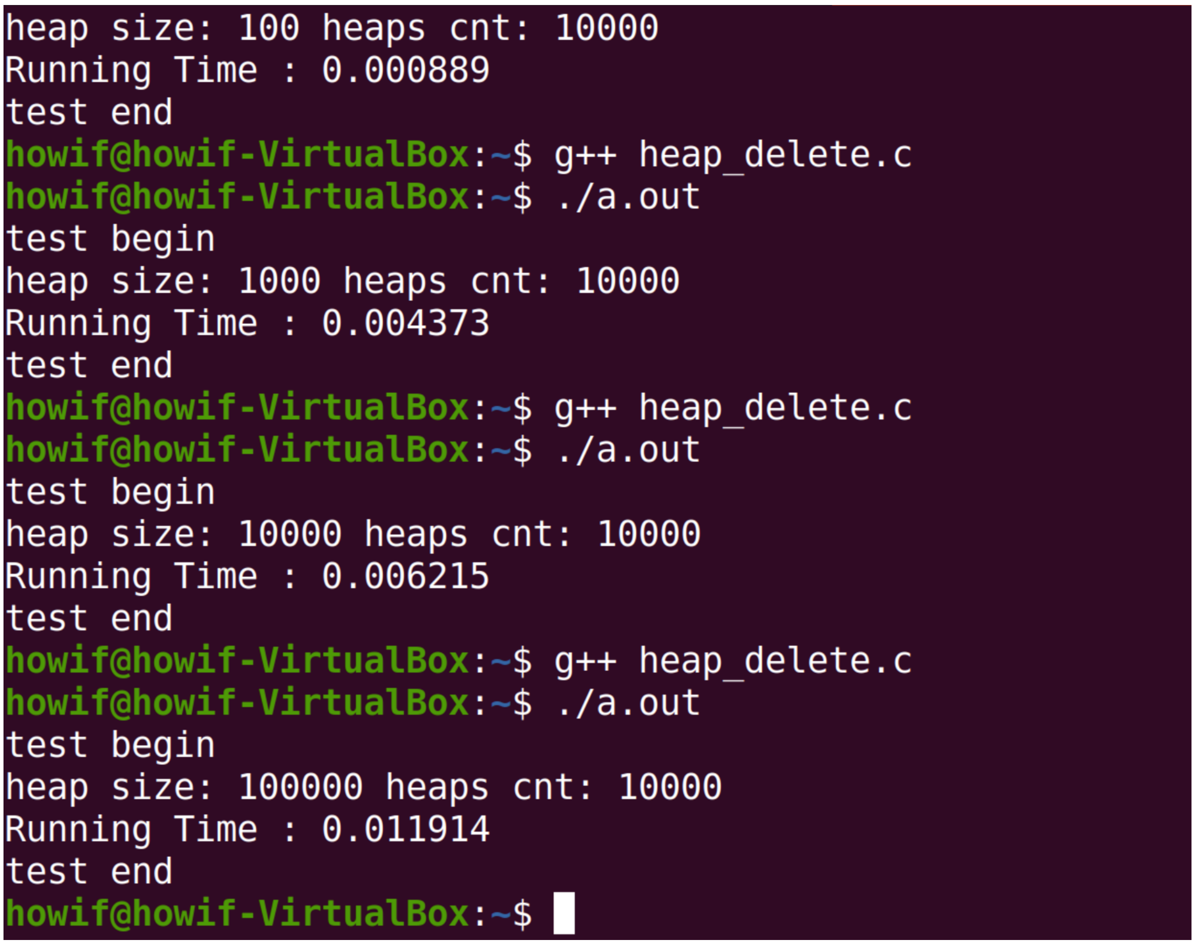
否则交换A[i] ↔ A[PARENT (i)] 向上调整

```
/*
 * Presudo HEAP-DELETE(A,i)
 * Input: A max-heap A and integers i.
 * Output: The heap A with the element a position i deleted.
 * A[i] ↔ A[heap-size[A]]
 * heap-size[A] ← heap-size[A] – 1
 * key ← A[i]
 * if key <= A[PARENT (i)] then
 *     MAX-HEAPIFY (A,i)
 * else
 *     while i > 1 and A[PARENT (i)] < key do
 *         A[i] ↔ A[PARENT (i)]
 *         i ← PARENT (i)
 *     end while
 * end
 */
```

完整代码

```
#include <vector>
#include <iostream>
#include <ctime>

using namespace std;
#define PARENT(i) (i/2)
#define LEFT(i)    (i<<1)
#define RIGHT(i)   (1+i<<1)

// assume LEFT(i) and RIGHT(i) is MAX-HEAP
void max_heapify(vector <int> &A, int index)
{
    int l = LEFT(index), r = RIGHT(index), largest;
    if ((l<=A[0]) && (A[l] > A[index])){
        largest = l;
    }
    else{
        largest = index;
    }

    if ((r<=A[0]) && (A[r] > A[largest])){
        largest = r;
    }

    if (largest != index){
        swap(A[index], A[largest]);
        max_heapify(A, largest);
    }

    return;
}

/*
 * Presudo HEAP-DELETE(A,i)
 * Input: A max-heap A and integers i.
 * Output: The heap A with the element a position i deleted.
 * A[i] ↔ A[heap-size[A]]
 * heap-size[A] ← heap-size[A] − 1
 * key ← A[i]
 * if key <= A[PARENT (i)] then
 *     MAX-HEAPIFY (A,i)
 * else
 *     while i > 1 and A[P ARENT (i)] < key do
 *         A[i] ↔ A[PARENT (i)]
 *         i ← PARENT (i)
 *     end while
 * end
 */
void heap_delete(vector<int> &A, int index){
    int key = A[index];
    if (index == A[0])
    {
        A[0]--;
        return;
    }
    swap(A[index], A[A[0]]);
    A[0]--;

    if ((index == 1) || (key <= A[PARENT(index)])){
        max_heapify(A, index);
    }else{
        while ((index > 1) && (A[PARENT(index)] < key)){
            swap(A[index], A[PARENT(index)]);
            index = PARENT(index);
        }
    }
}

/*
 * A[0] is size of heap
 */
vector<int> generate_max_heap(int size)
{
    vector<int> res;
    res.resize(size+1);
    res[0] = size;
    srand((int)time(0));
    res[1] = rand() % 20;
    for (int i=2; i<=size; i++)
    {
        res[i] = res[PARENT(i)] + rand() % 20;
    }

    return res;
}
#define HEAPS_CNT 10000
#define HEAP_SIZE 320
void test(void){
    vector<vector<int>> heaps;
    for (int i=0; i<HEAPS_CNT; i++){
        heaps.push_back(generate_max_heap(HEAP_SIZE));
    }
    srand((int)time(0));
    clock_t start = clock();
    for (int i=0; i<HEAPS_CNT; i++){
        heap_delete(heaps[i], rand()%HEAP_SIZE);
    }

    clock_t ends = clock();
    cout << "heap size: " << HEAP_SIZE << " heaps cnt: " << HEAPS_CNT << endl;
    cout <<"Running Time : "<<(double)(ends - start)/ CLOCKS_PER_SEC << endl;
    //cout << "test end" << endl;
    return;
}

int main(void)
{
    cout << "test begin" << endl;
    test();
    return 0;
}
```

运行时间



```
heap size: 100 heaps cnt: 10000
Running Time : 0.000889
test end
howif@howif-VirtualBox:~$ g++ heap_delete.c
howif@howif-VirtualBox:~$ ./a.out
test begin
heap size: 1000 heaps cnt: 10000
Running Time : 0.004373
test end
howif@howif-VirtualBox:~$ g++ heap_delete.c
howif@howif-VirtualBox:~$ ./a.out
test begin
heap size: 10000 heaps cnt: 10000
Running Time : 0.006215
test end
howif@howif-VirtualBox:~$ g++ heap_delete.c
howif@howif-VirtualBox:~$ ./a.out
test begin
heap size: 100000 heaps cnt: 10000
Running Time : 0.011914
test end
howif@howif-VirtualBox:~$ 
```

EX7.2-5

假设快速排序的每一层，所做划分的比例都是 $1-\alpha:\alpha$，其中 $0<\alpha\leqslant 1/2$ 是个常数。证明：在对应的递归树中，叶子结点的最小深度大约是 $-\lg n/\lg\alpha$，最大深度大约是 $-\lg n/\lg(1-\alpha)$。（无需考虑整数的取整舍入问题）

证明：

对于最小深度的递归树深度x：

$$n\alpha^x \le 1 \Rightarrow x \ge \log_\alpha \frac{1}{n}$$
$$\log_\alpha \frac{1}{n} = -\log_\alpha n = -\frac{\lg n}{\lg \alpha}$$

将上式α替换为1-α同理可证最大深度的递归树深度x大约是 -lgn/lg(1-α)

EX8.4-4

在单位圆中有 n 个点，$p_i=(x_i, y_i)$，使得 $0<x_i^2+y_i^2\leqslant 1$，$i=1, 2, \cdots, n$。假设所有点是均匀分布的，亦即，某点落在圆的任一区域中的概率与该区域的面积成正比。请设计一个 $\Theta(n)$ 期望时间的算法，来根据点到原点之间的距离 $d_i=\sqrt{x_i^2+y_i^2}$ 对 n 个点排序。（提示：在 BUCKET-SORT 算法中，设计适当的桶尺寸，以反映各个点在单位圆中的均匀分布。）

答： 将区域划分为n个等面积的圆环，桶尺寸为1/n (单位圆面积的1/n)

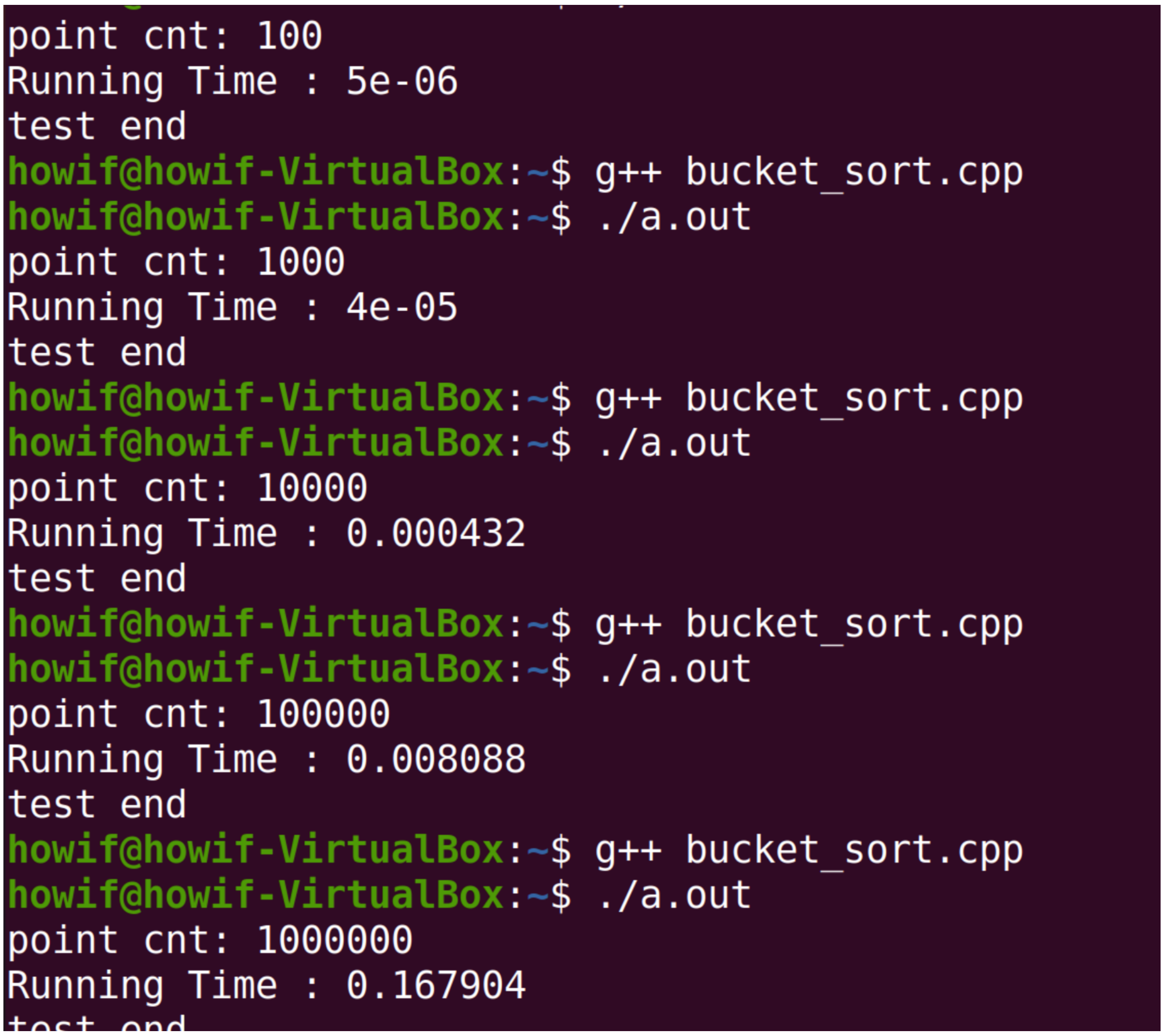
代码为了实现方便使用了尺寸的放缩 （实际坐标 = point.x / SCALE）

并使用距离的平方表示距离 （以距离平方等距划分确定桶范围 实际距离 = point.distance / SIZE_SCALE）

完整代码

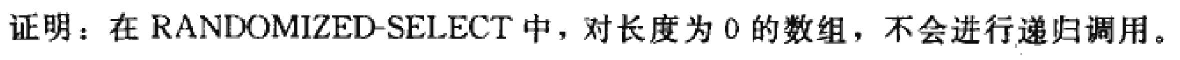
```
#include <vector>
#include <iostream>
#include <ctime>
using namespace std;
#define POINT_CNT 100
#define SCALE 10000
#define SIZE_SCALE 100000000
#define N (SIZE_SCALE / POINT_CNT)

typedef struct point {
    struct point *next;
    struct point *prev;
    int x;
    int y;
    bool x_flag; // true means positive, false means nagetive; doesn't matter in
this funciton
    bool y_flag;
    int distance;
}point;

static point res[POINT_CNT];

void generate_points (void){
    srand((int)time(0));
    //vector <point> res;
    //res.resize(POINT_CNT);
    for (int i=0; i<POINT_CNT; i++)
    {
        res[i].x = rand() % SCALE;
        res[i].y = rand() & SCALE;
        res[i].distance = res[i].x * res[i].x + res[i].y * res[i].y;
        if ((res[i].distance == 0) || (res[i].distance > SIZE_SCALE))
        {
            i--;
            continue;
        }
        res[i].x_flag = rand() % 2;
        res[i].y_flag = rand() % 2;
        res[i].next = NULL;
        res[i].prev = NULL;
    }
    //return res;
}
vector <point *> bucket(POINT_CNT);
vector <point *> tail(POINT_CNT);
void bucket_sort(void){
    clock_t start = clock();

    for (int i=0; i<POINT_CNT; i++)
    {
        bucket[i] = NULL;
        tail[i] = NULL;
    }

    // insert  to bucket
    for (int i=0; i<POINT_CNT; i++)
    {
        int bucket_number = (res[i].distance-1)/N;
        if (bucket[bucket_number] != NULL)
        {
            res[i].next = bucket[bucket_number];
            bucket[bucket_number]->prev = &res[i];

        }
        bucket[bucket_number] = &res[i];
    }

    // sort bucket[i] with insertion sort
    for (int i=0; i<POINT_CNT; i++)
    {
        if (bucket[i] == NULL)
        {
            continue;
        }

        point *head = bucket[i];
        tail[i] = head;
        point *head_save = head;
        point *tmp   = head->next;
        point *tmp2   = tmp;
        head->next = NULL;
        while (tmp != NULL)
        {
            point *prev = NULL;
            tmp2 = tmp->next;
            while (head && (head->distance < tmp->distance))
            {
                prev = head;
                head = head->next;
            }

            if (head == NULL)
            {
                prev->next = tmp;
                tmp->prev = prev;
                tmp->next = NULL;
                head = head_save;
                tail[i] = tmp;
            }
            else{
                if (head->prev == NULL)
                {
                    tmp->next = head;
                    tmp->prev = NULL;
                    head->prev = tmp;
                    head_save = tmp;
                    head = head_save;
                }
                else
                {
                    head->prev->next = tmp;
                    tmp->prev = head->prev;
                    tmp->next = head;
                    head->prev = tmp;
                    head = head_save;
                }
            }
            bucket[i] = head_save;
            tmp = tmp2;
        }
    }

    point *tmp_tail = NULL;
    point *ret_head = NULL;

    for (int i=0; i<POINT_CNT; i++)
    {

        if (bucket[i] != NULL)
        {
            if (ret_head == NULL)
            {
                ret_head = bucket[i];
            }
            else{
                bucket[i]->prev = tmp_tail;

                tmp_tail->next = bucket[i];
            }
            tmp_tail = tail[i];
        }
    }

    clock_t ends = clock();
    cout << "point cnt: " << POINT_CNT << endl;
    cout <<"Running Time : "<<(double)(ends - start)/ CLOCKS_PER_SEC << endl;
    cout << "test end" << endl;
#if 0
    point *tmp = ret_head;
    int i=0;
    while (tmp)
    {
            cout << i << "  "<< tmp->distance << " " << tmp->x << " " << tmp->y
<< endl;
            i++;
            tmp = tmp->next;
    }
#endif
}

int main(void)
{
    generate_points();

    bucket_sort();
    return 0;
}
```

运行时间



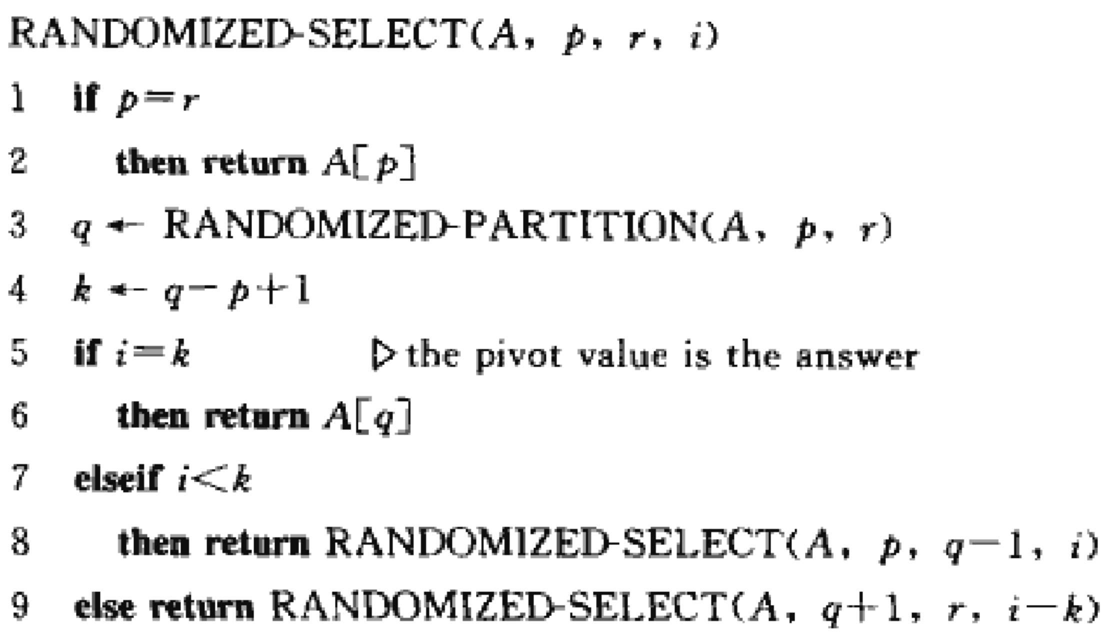
```
point cnt: 100
Running Time : 5e-06
test end
howif@howif-VirtualBox:~$ g++ bucket_sort.cpp
howif@howif-VirtualBox:~$ ./a.out
point cnt: 1000
Running Time : 4e-05
test end
howif@howif-VirtualBox:~$ g++ bucket_sort.cpp
howif@howif-VirtualBox:~$ ./a.out
point cnt: 10000
Running Time : 0.000432
test end
howif@howif-VirtualBox:~$ g++ bucket_sort.cpp
howif@howif-VirtualBox:~$ ./a.out
point cnt: 100000
Running Time : 0.008088
test end
howif@howif-VirtualBox:~$ g++ bucket_sort.cpp
howif@howif-VirtualBox:~$ ./a.out
point cnt: 1000000
Running Time : 0.167904
```

EX9.2-1



证明：在 RANDOMIZED-SELECT 中，对长度为 0 的数组，不会进行递归调用。

证明：RANDOM-SELECT伪代码如下



```
RANDOMIZED-SELECT(A, p, r, i)
1  if $p = r$
2      then return $A[p]$
3  $q \leftarrow$ RANDOMIZED-PARTITION(A, p, r)
4  $k \leftarrow q - p + 1$
5  if $i = k$              ▷ the pivot value is the answer
6      then return $A[q]$
7  elseif $i < k$
8      then return RANDOMIZED-SELECT(A, p, q-1, i)
9  else return RANDOMIZED-SELECT(A, q+1, r, i-k)
```

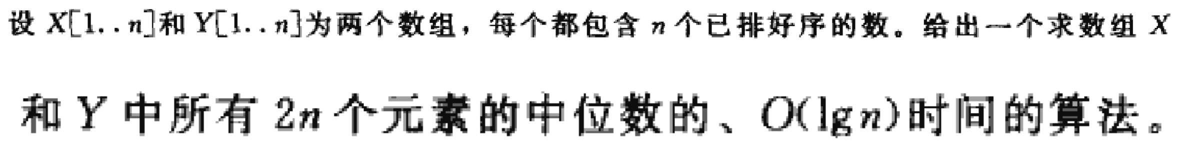
显然 伪代码中两处递归调用

第1处 i<k 数组长度为 q-1-p+1 = q-p 如果长度为0 需要p==q 这种情况下k == 1 不可能有i<k 所以第8行递归调用数组长度不会为0

第2处 i>=k 数组长度为r-q-1+1 = r-q 如果长度为0 需要q==r 这种情况下k == r-p+1 不可能有i>=k 所以第9行递归调用数组长度不会为0

综上在RANDOMIZED-SELECT中，对长度为0的数组，不会进行递归调用。

EX9.3-8



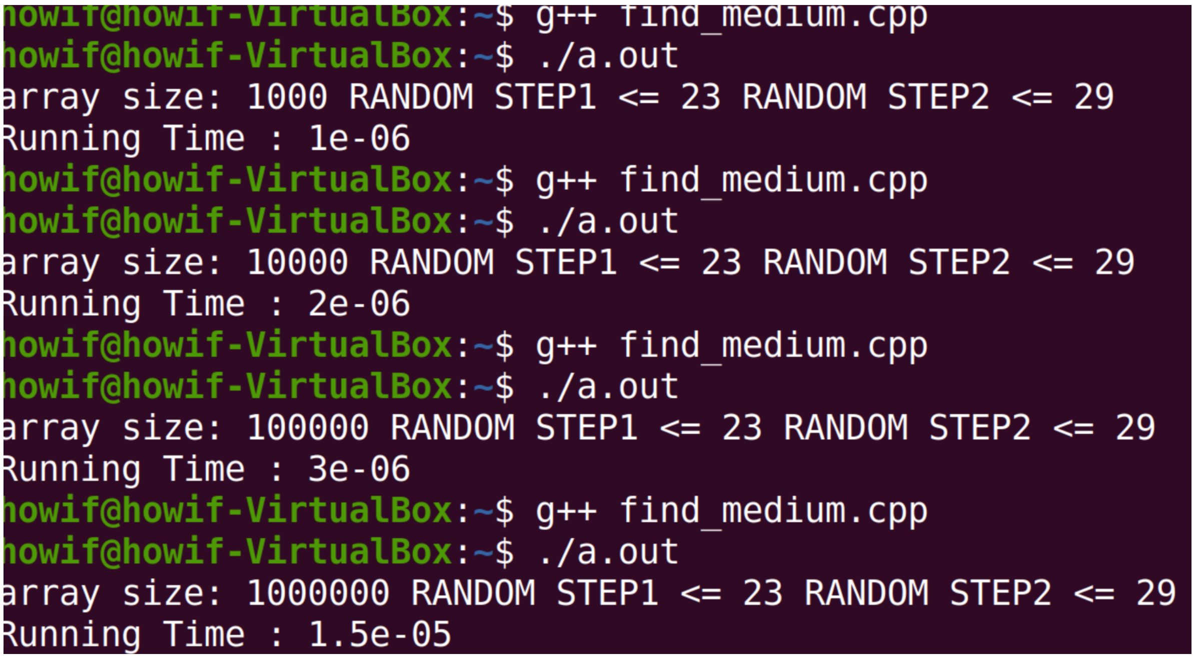
设 $X[1..n]$和 $Y[1..n]$为两个数组，每个都包含 n 个已排好序的数。给出一个求数组 X 和 Y 中所有 $2n$ 个元素的中位数的、$O(\lg n)$时间的算法。

答：思路为比较两个数组的中位数并每次去除肯定不包含中位数的数组的一部分，再递归调用该算法，直到获取中位数。

完整代码

```
#include <vector>
#include <iostream>
#include <ctime>
#include <climits>
using namespace std;
int retcheck(vector<int>& nums1, vector<int>& nums2, int i, int j, int l1, int
l2)
{
        int min1, min2, max1, max2;
        if (j < 0)
        {
            return -1;
        }

        if (j > l2)
        {
            return 1;
        }


        min1 = (i==0) ? INT_MIN : nums1[i-1];
        min2 = (j==0) ? INT_MIN : nums2[j-1];
        max1 = (i==l1) ? INT_MAX : nums1[i];
        max2 = (j==l2) ? INT_MAX : nums2[j];

        if (min1 > max2)
        {
            return -1;
        }

        if (min2 > max1)
        {
            return 1;
        }

        return 0;
    }

    double median(vector<int>& nums1, vector<int>& nums2, int i, int j)
    {
        int min1 = (i==0) ? INT_MIN : nums1[i-1];
        int min2 = (j==0) ? INT_MIN : nums2[j-1];
        int max1 = (i==nums1.size()) ? INT_MAX : nums1[i];
        int max2 = (j==nums2.size()) ? INT_MAX : nums2[j];

        if ((nums1.size() + nums2.size()) & 1 )
        {
            return min(max1, max2);
        }
        else
        {
            return ((double)(min(max1, max2) + max (min1, min2)))/2;
        }
    }

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
        if (nums1.size() < nums2.size())
        {
            // nums1.length() >= nums2.length()
            return findMedianSortedArrays(nums2, nums1);
        }

        int l1 = nums1.size();
        int l2 = nums2.size();
        int sum = l1 + l2;
        int mid = sum / 2; // divide sum into mid || mid or mid + 1
        //int s_m = sum -mid;
        //int flag = sum & 1; // flag = 0 means even, flag = 1 means odd;

        int i = l1 / 2;
        int j = mid - i;
        int ret;
        int high = l1;
        int low = 0;
        while ((ret  = retcheck (nums1, nums2, i, j, l1, l2)) != 0)
        {
            if (ret == -1)
            {
                high = i;
                i = (i+low)/2;

            }else
            {
                low = i;
                if (i == (i+high)/2)
                {
                    i++;
                }
                else
                {
                    i = (i+high)/2;
                }

            }

            j = mid - i;
        }

        return median (nums1, nums2, i, j);
    }
#define N 1000
#define STEP1 2
#define STEP2 3
void generate_vector(vector<int> &nums, int cnt, int step)
{
    srand((int)time(0));
    nums.resize(cnt);
    nums[0] = rand()%step;
    for (int i=1; i<cnt; i++)
    {
        nums[i] += nums[i-1] + rand()%step;
    }
}
int main(void)
{
    vector<int> nums1 = {}, nums2 = {};
    generate_vector (nums1, N, STEP1);
    generate_vector (nums2, N, STEP2);
    int ret =  findMedianSortedArrays(nums1, nums2);
    //cout << ret << endl;
    return 0;
}
```

运行时间



```
howif@howif-VirtualBox:~$ g++ find_medium.cpp
howif@howif-VirtualBox:~$ ./a.out
array size: 1000 RANDOM STEP1 <= 23 RANDOM STEP2 <= 29
Running Time : 1e-06
howif@howif-VirtualBox:~$ g++ find_medium.cpp
howif@howif-VirtualBox:~$ ./a.out
array size: 10000 RANDOM STEP1 <= 23 RANDOM STEP2 <= 29
Running Time : 2e-06
howif@howif-VirtualBox:~$ g++ find_medium.cpp
howif@howif-VirtualBox:~$ ./a.out
array size: 100000 RANDOM STEP1 <= 23 RANDOM STEP2 <= 29
Running Time : 3e-06
howif@howif-VirtualBox:~$ g++ find_medium.cpp
howif@howif-VirtualBox:~$ ./a.out
array size: 1000000 RANDOM STEP1 <= 23 RANDOM STEP2 <= 29
Running Time : 1.5e-05
```