Problem Set 3 (3.21)

blem Set 3.1

oblem 3.1.1 (BG 5.8)

uicksort can be modified to find the kth-smallest key among n keys so that in most cases it does nuch less work than is needed to sort the set completely.

a. Write a modified Quicksort algorithm called findKth for this purpose.

b. Show that when this algorithm is used to find the median, the worst case is in $\Theta(n^2)$.

c. Develop a recurrence equation for the average running time of this algorithm.

d. Analyze your algorithm's average running time. What is the asymptotic order?

a. findKth (A, p, r, k):

```
    if p=r then return A[p];
    q:= PARTITION(A,p,r); //与Quicksort中的partition相同
    x:= q-p+1
    if k=x then return A[q];
    else if k<x then ~~return~~
        return findkth(A,p,q-1,k); //在左半部分继续递归查找
    else
        return findkth (A,q+1,r,k-x); //在右半部分继续递归查找
```

(同书中P30 SELECT-LINEAR 算法)

b. 最坏情况下：每一次PARTITION仅只去除了1个元素，直到递归调用的A中只有1个元素时才结束的算法，此时算法的时间复杂度 两次递归时 $W(n)=W(n-1)+\Theta(n)$

$\therefore W(n)=W(1)+(n-1)\Theta(n)$

$\therefore W(n)=\Theta(n^2)$

c. 记小于等于基准元素个数为k，则$k\in[1,n]$，设算法的时间代价为$T(n)$

则 $T(n)\le\sum_{k=1}^{n}\frac{1}{n}(T(\max(k-1,n-k))+O(n))$

$=\sum_{k=1}^{n}\frac{1}{n}\times T(\max(k-1,n-k))+O(n)$

扫描全能王 创建

d. 由c中的递推表达式 $T(n) \le \sum_{k=1}^{n} \frac{1}{n} T(\max(k-1, n-k)) + O(n)$ 得:

$$E[T(n)] \le \frac{2}{n}\sum_{k=\lfloor \frac{n}{2}\rfloor}^{n-1} E[T(k)] + O(n)$$

假设∃常数c使得 $E[T(n)] \le cn$,

∃常数a使得 $O(n)$ 可被替换为 an,

则有 $E[T(n)] \le \frac{2}{n}\sum_{k=\lfloor \frac{n}{2}\rfloor}^{n-1} ck + an$

$$= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor \frac{n}{2}\rfloor - 1} k\right) + an$$

$$= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor \frac{n}{2}\rfloor - 1)\lfloor \frac{n}{2}\rfloor}{2}\right) + an$$

$$\le \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\frac{n}{2}-2)(\frac{n}{2}-1)}{2}\right) + an$$

$$= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an$$

$$\le cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

取 $\frac{cn}{4} - \frac{c}{2} - an \ge 0$，即 $n \ge \frac{2c}{c-4a}$ 时，就有 $E[T(n)] \le cn$ 成立，由此证明了 $E[T(n)]$

∴平均时间下的渐近复杂度为 $O(n)$。

Problem 3.1.2 (Huang 8.6)

给定一个有 n 个不同整数的集合 S，用 M 表示 S 的中位数。请设计算法找出 S 中与 M 的大小最接近的 k 个数（k 远小于 n）。例如，集合 S={6, 7, 50, 800, 900}，中位数 M 是 50，两个（$k=2$）与中值 M 最接近的数是 6 和 7。

1) 请设计一个时间复杂度为 $O(nlogn+k)$ 的算法。

2) 请设计一个时间复杂度为 $O(n+klogk)$ 的算法。

(1) 算法思路：先对集合S做一个排序（用快排、堆排序或归并排序等均可），然后找到中位数前、后各k个元素，

中位数前的元素按从右至左的顺序排为A{$a_1, a_2, \cdots a_k$} 其中 $a_1 \geq a_2 \geq \cdots \geq a_k$

中位数后的元素按从左至右的顺序排为B{$b_1, b_2, \cdots b_k$} 其中 $b_1 \leq b_2 \leq \cdots \leq b_k$

然后设 p, q 两个指针分别指向A与B当前遍历到的元素，p, q 初始值为 1，比较 $|A[p]-M|$ 与 $|B[q]-M|$ 的大小，将其中较小的值对应的A或B中的元素加入到k个与M最接近的数的集合中。

伪代码如下：

```
FindNearKNums (int S[ ], int n) {
    Sort(S);
    if (n%2==0) M= (S[n/2]+S[n/2+1])/2  else  M= S[(n+1)/2]/2
    for i:=1 to k do
        A[i]= S[k-i+1];
        B[i]= S[k+i];
    p=1, q=1;
    count=0
    while count!=k do
        if abs(A[p]-M) > abs(B[q]-M) then
            res[count++]= B[q++];
        else
            res[count++]= A[p++];
```

算法时间复杂度分析：Sort: $O(nlogn)$，for循环：$O(k)$，while循环：$O(k)$

∴ 时间复杂度为 $O(nlogn+k)$

①

(2) 算法思路：首先找到中位数，然后计算出其余元素与中位数的差的绝对值，然后找出这些绝对值中第k大的元素，然后再以k大的元素为pivot对这些绝对值进行划分，最终在pivot左侧的绝对值所对应的初始元素即为与中位数最接近的k个数。对左侧包含第k大元素在内的k个元素进行一次排序。

算法时间复杂度分析：寻找中位数的代价为O(n)，计算出其余元素与中位数的差的绝对值的时间复杂度为O(n)，找出第k大的元素的时间复杂度为O(n)，以k大的元素划分的时间复杂度为O(n)，对左侧k个元素排序的时间复杂度为O(klogk)，所以算法最终的时间复杂度为O(n+klogk)。

Problem 3.1.4 (Huang 9.12)

假设有一个数组 $A[1..n]$，它满足以下两个条件：$A[1] \geqslant A[2]$；$A[n-1] \leqslant A[n]$。当元素 $A[x]$ 并不大于它的两个邻居（$A[x-1] \geqslant A[x]$, $A[x+1] \geqslant A[x]$）时，我们称 $A[x]$ 为局部最小元素（local minimum）。例如，在图 9.4 的数组中有 6 个局部最小元素。

9	7	7	2	1	3	7	5	4	7	3	3	4	8	6	9

1) 我们可以利用 $O(n)$ 的时间通过扫描一遍数组找到一个局部最小元素。请设计一个用 $O(logn)$ 的时间找到一个局部最小元素的算法。

2) 在给定的边界条件下，请说明至少存在一个局部最小元素。

O(n)算法的

(1) 伪代码如下:

```
FindPartMinElement (int A[ ]):
    for i:=2 to n-1 do
        if (A[i] <= A[i+1])   //如果A[2]<=A[3],说明A[2]就是局部最小元素,否则
            return i;           A[2]>A[3],则进入下一次循环与后一个比较即可…
                                循环比较至比较完
    return -1
```

O(logn)算法的伪代码如下:

```
FindPartMinElement(int A[ ], l, r)
    int mid = (l+r)/2;
    if l==r then
        return A[l];
    else
        if A[mid] <= A[mid+1] and A[mid] <= A[mid-1] then
            return A[mid];
        else if A[mid] > A[mid-1] then
            return FindPartMinElement(A, l, mid);
        else
            return FindPartMinElement(A, mid, r);
```

该算法的合理性在(2)中证得。

(2) 证明: 假设在给定边界的条件下,A数组中不存在任何的局部最小元素,也就是说每一个数组当中的元素都至少比邻居中的一个大,考虑第二个元素A[2]与第三个元素A[3]:

由A[1]≥A[2],A[2]不是局部最小元素可得 A[2]>A[3],则可得A[3]>A[4] ……

A[n-1]>A[n] 与题中条件 A[n-1]≤A[n] 不符,∴矛盾。

∴ 在给定边界的条件下,对于任意情况来说,A中都至少存在一个局部最小元素。

由(2)证明(1)中O(logn)的正确性: 在递归调用时为什么可以直接缩小范围? 因为满足判断条件使得递归的新区间仍满足A数组的条件,所以一定会存在局部元素,因此可直接每次缩小一半的范围。

3.1.5

(a) 算法思路:

① 给n个元素排序按权重大小从小到大排序 ($O(n\log n)$)

② 从左往右遍历n个元素，累加各元素权重至当前值大于 $\frac{W}{2}$ 时停止遍历，返回当前元素 ($\frac{1}{2}$-median) ($O(n$

该算法时间复杂度为 $O(n\log n)$

(b) 算法思路

① 使用SELECT-WLINEAR算法中的选取pivot的方法对n个元素按权重进行划分，权重小于 m^* 的在一侧，权重大于 m^* 的在另一侧。算出权重小于 m^* 的元素的权重和 W_1，算出权重大于 m^* 的元素的权重和 W_2 ($O(n)$)。

② 对SELECT-WLINEAR的递归部分进行修改:

if $W_1 < \frac{1}{3}W$ and $W_2 < \frac{2}{3}W$ then

  return 权重为 m^* 对应的元素

else if $W_1 < \frac{1}{3}W$

  $m^* += W_1$; 从SELECT-WLINEAR继续在大于 m^* 的元素中递归所得值返回

else

  $m^* += W_2$;

  return SELECT-WLINEAR继续在小于 m^* 的元素中递归所得值。

3.1.6

证明

(a) 在任何情况下总是比 m^* 小的元素是位于AB的元素，这些元素的个数为 $(\lceil \frac{n}{3} \rceil \times \frac{1}{2} - 1) \times 2 + 1$
$= \lceil \frac{n}{3} \rceil - 1$ $n/3$ 不为整数时，$\lceil \frac{n}{3} \rceil \le \frac{n+2}{3} + 1 < \frac{n}{3} + 3$

(b) 由(a)可知，对于SELECT3算法而言，在任何情况下总是比 m^* 小的元素不超过 $\frac{n}{3}+3$，考虑这种情况下比 m^* 大的元素个数 $\ge n - \frac{n}{3} - 3 = \frac{2n}{3} - 3$

$\therefore$ SELECT3算法的最坏情况时间复杂度有：

$$T(n) \ge \underbrace{T(\lceil \tfrac{n}{3} \rceil)}_{\text{递归地选择中位数的中位数.}} + \underbrace{T(\tfrac{2n}{3} - 3)}_{\text{递归对大于 } m^* \text{ 的元素做选择}} + \Theta(n)$$

$$= T(\lceil \tfrac{n}{3} \rceil) + T(\tfrac{2n}{3} - 3) + \Omega(n)$$

(c) 假设对于某个常数 a，有 $T(n) \ge an\log n$，对于某个常数 c，有 $\Omega(n) = cn$，代入上式

有：$T(n) \ge T(\lceil \frac{n}{3} \rceil) + T(\frac{2n}{3} - 3) + cn$

$\ge \frac{an}{3}\log\frac{n}{3} + (\frac{2n}{3} - 3)a\log(\frac{2n}{3} - 3) + cn$

$= an\log\frac{n}{3} - \frac{2}{3}an\log\frac{n}{3} + \frac{2}{3}an\log(\frac{2n}{3} - 3) - 3a\log(\frac{2n}{3} - 3) + cn$

$= an\log n - an\log 3 - \frac{2}{3}an\log\frac{n}{3} + \frac{2}{3}an\log(\frac{2n}{3} - 3) - 3a\log(\frac{2n}{3} - 3) + cn$

$= an\log n + (an(\frac{2}{3}\log(\frac{2n}{3} - 3) - \frac{2}{3}\log n - \frac{1}{3}\cdot\log 3) - 3a\log(\frac{2n}{3} - 3) + cn))$

$= an\log n + (an(\frac{2}{3}\log(2n-9) - \frac{2}{3}\log n - \log 3) - 3a\log(2n-9) + 3a\log 3 + cn))$

$= an\log n + (3a\log 3 - an\log 3 + cn - an\log 3 + \frac{2(n-3)}{3}a\log(2n-9))$

$n<6$ 时，令 $T(n) = O(1)$. $= an\log n + (a(n-3)(\frac{2}{3}\log(2n-9) - \log 3) + cn - an\log 3)$

$n \ge 6$ 时，有 $a(n-3)(\frac{2}{3}\log(2n-9) - \log 3) > 0$，只需再令 $c > a\log 3$ 即有 $T(n) \ge an\log n$ 成立，由此证明了 $T(n) = \Omega(n\log n)$

3.2.1

算法思路：①给2^k个球编号，编号的方式用二进制来表示，共k位即可为所有小球编号。

②从最低位开始，取最低位编号为0的球（共2^{k-1}个）分为两组，每组有2^{k-2}个球，放在天平两边，如果重量相同，说明不一样重的小球末位编号为1，否则其末位编号为0。

③然后再将上一轮中重量不相同的两组合为一组，再按②中方法确定倒数第2位的值，一直重复此过程到只剩两个球的时候。

④只剩两个球时意味着编号的第0位到第k-1位都确定了，那么最后一次比较只需任取两个球之外的一个球与两个球中的一个做比较即可，然后就能确定下来哪一个球是重量与其它的球不一致的了。

……然后就能确定下来哪一个球是重量与其它的球不一致的了。

3.2.2

1. 伪代码如下：

```
int Height(tree t){ //tree为树节点类的指针)
        if(t==NULL) return 0;
        else return max(Height(leftChild), Height(rightChild))+1;}
```

设该算法运行时间为T(n)，由题中可假设T的子树是平衡的，则左右子树节点可视作近似相同

$$T(n)=2T\left(\frac{n-1}{2}\right)+O(1)$$

由master定理，$T(n)\in O(n)$

2. 伪代码如下:

```
d=0; // 全局变量记录更新的最长直径
int dtree(tree t, int d1){
    if(t==NULL) return 0;
    int l=dtree(t->leftChild, d1), r=dtree(t->rightChild, d1);
    d=max(l+r+1, d); // 要么是左边,要么在右边,要么跨越当前节点形成最长直径
    return max(l,r)+1; // 返回的是当前节点的单支最长高度
}
dtree(root, d);
```

设该算法运行时间为 $T(n)=2T(\frac{n-1}{2})+O(1)$

$\therefore T(n)\in O(n\log n)$