Homework #2

211220019 陈骏桢

4.1

证明：
设二叉树中有一个子节点的节点个数为 n_1，有两个子节点的节点个数为 n_2，共有 n 个节点，则 $n = L + n_1 + n_2$，除了根节点外的其他节点都由一条边向下延伸得到，则该二叉树总共有 $n-1$ 条边，同时，有一个子节点的节点会向下延伸出一条边，有两个子节点的节点向下延伸出两条边，则 $n-1 = 2n_2 + n_1$，联合两个式子可得：$2n_2 + n_1 + 1 = L + n_1 + n_2$，则 $L = n_2 + 1$；
因为高度为 h，且高度为 0 的节点无子节点，即叶节点，所以 $n_2 \leq \sum_{i=0}^{h-1} 2^i = 2^h - 1$；
所以 $L \leq 2^h$。

4.4

1) 先两两比较，然后将两个较大值比较找到最大值，两个较小值比较找到最小值，然后将剩余两个比较大小；
   算法伪代码如下：

```
Algorithm 1: sort(A[1..4])
Input: A[1..4]
Output: A[1..4]
if A[1] > A[2] then
    SWAP(A[1], A[2]);
end
if A[3] > A[4] then
    SWAP(A[3], A[4]);
end
if A[1] > A[3] then
    SWAP(A[1], A[3]);
end
if A[2] > A[4] then
    SWAP(A[2], A[4]);
end
if A[2] > A[3] then
    SWAP(A[2], A[3]);
end
```

2) 首先可知：将一个元素插入两个有序元素中最多只需要 2 次比较；而将一个元素插入三个有序元素中时，可先与中间的元素比较，然后和两边的元素中的一个比较，最多也只要 2 次比较；

不妨设这 5 个数为 a、b、c、d、e，首先比较 a、b 大小，比较 c、d 大小，不妨设 a<b,c<d。接着比较 a、c 大小，不妨设 a<c，于是，得到一个长度为 3 的有序序列 a<c<d，同时知道 a<b，用 2 次比较将 e 插入到 a、c、d 这个数列中。由于已知 a<b，于是 b 最多只与 c、d、e 三个数确定相对大小（若 a≥e，则 e<a<c<d，b 与 e 的相对大小也已经确定了，所以后面讨论 a<e 的情况），而且前面已将 e、c、d 这三个数之间的相对大小确定，不妨设 c<d<e，则我们可以用 2 次比较将 b 插入到 c、d、e 这个序列中，又因为已知 b、c、d、e 都比 a 大，于是这 5 个数排序完毕，一共用了 7 次比较。

算法伪代码如下：

Algorithm 2: Insert($A[1,2,3], a$)

Input: $A[1,2,3], a$

Output: $B[1,2,3,4]$

if $a < A[2]$ **then**
  if $a < A[1]$ **then**
    $B[1,2,3,4] = \{a, A[1], A[2], A[3]\}$;
  else
    $B[1,2,3,4] = \{A[1], a, A[2], A[3]\}$;
else
  if $a < A[3]$ **then**
    $B[1,2,3,4] = \{A[1], A[2], a, A[3]\}$;
  else
    $B[1,2,3,4] = \{A[1], A[2], A[3], a\}$;

Algorithm 3: Sort($A[1..5]$)

Input: $A[1..5]$

Output: $A[1..5]$

$B[1..4] := \{0,0,0,0\}$;
if $A[1] > A[2]$ **then**
  SWAP($A[1], A[2]$);
if $A[3] > A[4]$ **then**
  SWAP($A[3], A[4]$);
if $A[1] > A[3]$ **then**
  $B[1,2,3] = A[3,1,2]$;
  $B[1..4] =$Insert($B[1,2,3], A[5]$);
  $A[1] = B[1]$;
  $A[2..5] =$Insert($B[2,3,4], A[4]$);
else
  $B[1,2,3] = A[1,3,4]$;
  $B[1..4] =$Insert($B[1,2,3], A[5]$);
  $A[1] = B[1]$;
  $A[2..5] =$Insert($B[2,3,4], A[2]$);

4.8

将快速排序进行到 log k 层即可，但是要把快速排序改一下，每层排序都要找到中位数再往下一层排。算法伪代码如下：

Algorithm 4: k-sorted

```
Input: A[1..n], k
Output: A[1..n]
if n=k then
    return;
i:=2; j:=n; key:=A[1];
while true do
    while A[i]≤key do
        i++;
        if i≥j then break;
    while A[j]≥key do
        j–;
        if j≤i then break;
    if i≥j then
        if j=n/2 then
            break;
        else if j>n/2 then
            SWAP(A[1],A[j]);
            i=1; key=A[1];
        else
            SWAP(A[1],A[j]);
            j=n; key=A[i];
    else
        SWAP(A[i],A[j]);
SWAP(A[1],A[j]);
k-sorted(A[1..n/2]);
k-sorted(A[n/2+1..n]);
```

4.9

可以用快速排序法，先用某个螺母将螺钉划分，然后用于该螺母对应的螺钉对螺母划分，然后递归。算法伪代码如下：

Algorithm 5: sort(A,key)

```
Input: A[1..n], key
Output: A[1..n]
index:=0;
if n=1 then
    return 1;
end
i:=0; j:=1;
while j≤n do
    if A[j]≤key then
        i++;
        SWAP(A[i],A[j]);
        if A[i]=key then
            index=i;
        end
    end
end
SWAP(A[i],A[index]);
return i;
```

Algorithm 6: match(nail[1..n],nut[1..n])

```
Input: nail[1..n], nut[1..n], k
Output: nail[1..n], nut[1..n]
index:=sort(nail[1..n],nut[1]);
sort(nut[1..n],nail[index]);
match(nail[1..q-1],nut[1..q-1]);
match(nail[q+1..n],nut[q+1..n]);
```

4.11

1) 反证法：

   假设存在 $(i,j)(A[i] > A[j])$，使得 $j-i>2$，则 i 和 j 至少存在两个数，设为 $a,b(i<a<b<j)$，则可分为以下两种情况：

   i) $A[i] \le A[a]$，则 $A[a] > A[j]$，(a,j) 为逆序对，又因为只有 2 个逆序对，所以必有 $A[a] <\le A[b]$，则 $A[b] > A[j]$，(b,j) 也为逆序对，有 3 个逆序对，矛盾；

   ii) $A[i] > A[a]$，则 (i,a) 为逆序对，又因为只有 2 个逆序对，所以必有 $A[i] \le A[b]$，则 $A[b] > A[j]$，则 (b,j) 为逆序对，有 3 个逆序对，矛盾；

   综上所述，假设不成立，$j-i\le 2$。

2) 只需要一直比较相邻元素，遇到逆序对时将两元素互换，然后比较互换后较小元素与前一元素大小，若为顺序对，则原逆序对中较大元素与后一元素互换。算法伪代码如下：

Algorithm 7: sort

```
Input: A[1..n]
i:=1;
if A[1]>A[2] then
    SWAP(A[1],A[2]);
    SWAP(A[2],A[3]);
    break;
while true do
    i=i+1;
    if A[i]>A[i+1] then
        SWAP(A[i],A[i+1]);
        if A[i]<A[i-1] then
            SWAP(A[i-1],A[i]);
        else
            SWAP(A[i+1],A[i+2]);
        break;
```

4.14

首先先考虑函数：先判断两词字母数量，若不一样，则不为易位词，跳出；否则，将每个单词内的字母排序，然后再对排序后的单词进行排序，则相邻的相同的单词为易位词。

我们可以先将文件中所有词按字母数量分组，然后只在相同组中进行判断，同时，在分组时要判断是否已出现相同词语，因此可以在分组时就将词语排序。

7.1

将数组分成两半，在对其进行排序的同时，计算左右各自的逆序对数之和，合并时再计算两半之间的逆序对个数，在相加。算法伪代码如下两个函数：

Algorithm 8: merge(A[1..n],mid,C)

```
Input: A[1..n],mid,C
ln=mid; rn=n-mid;
L[1..ln]=A[1..ln];
R[1..rn]=A[ln+1..n];
L[ln+1]:=+∞; R[rn+1]:=+∞;
i:=1; j:=1; num:=0;
for k:=1 to n do
    if L[i]>C·R[j] then
        A[k]:=R[j];
        num:=num+mid-i+1;
        j:=j+1;
    else
        A[k]:=L[i];
        i:=i+1;
    end
end
return num;
```

```
Algorithm 9: count(A[1..n],C)
Input: A[1..n],C
if n≤1 then return 0;
mid:=⌊n/2⌋;
lnum:=count(A[1..mid],C);
rnum:=count(A[mid+1..n],C);
mnum:=merge(A[1..n],mid,C);
return lnum+rnum+mnum;
```

7.4

1) $T(n,k)=\sum_{i:=1}^{k-1} i\cdot n=\frac{nk(k-1)}{2}$
所以时间复杂度为 $\frac{nk(k-1)}{2}=\Theta(nk^2)$；

2) 递归算法：当 $k=1$ 时，直接返回；若 $k\geq 2$，则将 k 个数组分成两份，每份 $k/2$ 个数组，先将每份中的所有数组合并，得到两个大的排好序的数组，然后将这两个大的数组合并。算法伪代码如下：

```
Algorithm 10: merge(A[1..k][1..n])
Input: A[1..k][1..n]
Output: B[1..nk]
if k=1 then
    return A[1][1..n];
B[1..nk]:=0;
C[1..nk/2]:=merge(A[1..k/2][1..n]);
D[1..nk/2]:=merge(A[k/2+1..k][1..n]);
C[nk/2+1]:=+∞; D[nk/2+1]:=+∞;
a:=1; b=1;
for k:=1 to nk do
    if C[a]>D[b] then
        B[k]:=D[b]; b:=b+1;
    else
        B[k]:=C[a]; a:=a+1;
return B[1..2n];
```

7.5

1) 递归算法，叶节点的高度为 0，非叶节点的高度为其左右子树高度的较大值 +1。算法伪代码如下：

Algorithm 11: height(root)

```
Input: root
Output: height
if root=NULL then return -1;
return max(height(root->left),height(root->right))+1;
```

2) 定义 lway,rway 表示节点的左右子树中沿远离根结点方向的最长路径的长度，则直径为所有节点的 (lway+rway+2) 的最大值。算法伪代码如下：

Algorithm 12: diameter(root,&way)//引用 way

```
Input: root
Output: diameter
if root=NULL then
    way:=-1;
    return -1;
end
lway:=0; rway:=0;
ld:=diameter(root->left,lway); //引用 lway
rd:=diameter(root->right,rway); //引用 rway
way:=max(lway,rway)+1;
return max(ld,rd,lway+rway+2)
```

14.1

证明：设 h 为堆中某元素在堆中的索引，则：$\lfloor \frac{1}{2}h \rfloor$ 为该节点父节点的索引，$\lceil \log(h+1) \rceil$ 为该节点的深度，则 $\lceil \log(\lfloor \frac{1}{2}h \rfloor + 1) \rceil$ 为其父节点的深度；而堆中节点的深度等于其父亲节点的深度 +1，所以命题正确。

14.2

因为 $k \ll n$，所以第 k 大的元素最多在第 k 层，用堆排序算法，排序前 k 层共 2^k-1 个元素即可，时间复杂度为 $O(k2^k)$。算法伪代码如下：

Algorithm 13: maxk(heap[1..n])

```
Input: heap[1..n]
dad:=1; son:=1;
SWAP(heap[1],heap[2^k − 1]);
for i := 2 to k do
    dad:=1; son:=2dad;
    while son ≤ 2^k − i do
        if (son + 1 ≤ 2^k − i) and (heap[son] < heap[son + 1]) then
            son:=son+1;
        end
        if heap[dad] > heap[son] then
            break;
        else
            SWAP(heap[dad],heap[son]);
            dad=son; son=2dad;
        end
    end
    SWAP(heap[1],heap[2^k − i])
end
return heap[2^k − k];
```

14.3

设下标为 i 的节点在第 h 层第 k 个，则：
$i=(1+d+\cdots+d^{h-2})+k=\frac{d^{h-1}-1}{d-1}+k$ 证明：

1) D-ARY-PARENT(i)：该节点的父节点在第 $h-1$ 层的第 $\lceil\frac{k}{d}\rceil$ 个，则其父节点的索引为：$\frac{d^{h-2}-1}{d-1}+\lceil\frac{k}{d}\rceil=\lfloor\frac{d^{h-2}-1}{d-1}+\frac{k+d-1}{d}\rfloor=\lfloor\frac{d^{h-1}-d+kd-k+(d-1)^2}{d(d-1)}\rfloor$
$=\lfloor\frac{1}{d}(\frac{d^{h-1}-1}{d-1}+k)+\frac{d^2-3d+2}{d(d-1)}\rfloor=\lfloor\frac{i+d-2}{d}\rfloor=\lfloor\frac{i-2}{d}+1\rfloor$；
所以 D-ARY-PARENT(i) 正确；

2) D-ARY-CHILD(i,j)：该节点的第 k 个子节点在第 $h+1$ 层的第 $(k-1)d+j$ 个，则其第 j 个子节点的索引为：$\frac{d^h-1}{d-1}+(k-1)d+j$
$=\frac{d^h-d}{d-1}+(k-1)d+j+1=d(\frac{d^{h-1}-1}{d-1}+k-1)+j+1=d(i-1)+j+1$；
所以 D-ARY-CHILD(i,j) 正确。

14.4

数学归纳法证明：

i) 当 $n=1$ 时，节点高度之和为 $0\le n-1=0$，结论成立；

ii) 假设当 $n<k(k>1)$ 时，结论均成立，则当 $n=k$ 时，由于堆是完全二叉树，所以其左子树的高度 $\ge$ 右子树的高度。设左右子树高度分别为 h_1,h_2，节点数分别为 n_1,n_2 分以下 2 种情况：

① $h_1>h_2$，则 $h_1=h_2+1=\lfloor\log(n)\rfloor-1$，右子树为完美二叉树，$n_2=2^{h_2+1}-1$，右子树高度和为 $\sum_{i:=0}^{h_2} i\cdot 2^{h_2-i}=\sum_{i:=0}^{h_2-1} 2^{h_2-i}-h_2=2^{h_2+1}-h_2-2$；
左子树：$n_1=n-1-n_2=n-2^{h_2+1}$，
左子树高度和 $\le n_1-1=n-2^{h_2+1}-1$，
所以该堆高度和 $\le n-2^{h_2+1}-1+2^{h_2+1}-h_2-2=n-h_2-3\le n-1$；结论成立；

② $h_1=h_2$，则左子树为完美二叉树，$n_1=2^{h_1+1}-1$，左子树高度和为 $\sum_{i:=0}^{h_1} i\cdot 2^{h_1-i}=\sum_{i:=0}^{h_1-1} 2^{h_1-i}-h_1=2^{h_1+1}-h_1-2$；
右子树：$n_2=n-1-n_1=n-2^{h_1+1}$，
右子树高度和 $\le n_2-1=n-2^{h_1+1}-1$，
所以该堆高度和 $\le 2^{h_1+1}-h_1-2+n-2^{h_1+1}-1=n-h_1-3\le n-1$；结论成立；

iii) 综上所述：一个有 n 个节点的堆中，所有节点的高度之和最多为 n-1;

堆的高度和为 $2^{h_1+1}-h_1-2+2^{h_2+1}-h_2-2+h_1+1$
$=2^{h_1+1}+2^{h_2+1}-h_2-3$，分以下 2 种情况讨论：

i) 当 $h_1>h_2$ 时，则 $h_1=h_2+1=\lfloor\log(n)\rfloor-1$，若高度和为 $n-1$，则 $n=2^k(k\in\mathbb{N})$；

ii) 当 $h_1 = h_2$ 时，则 $h_1 = h_2 = \lfloor \log(n) \rfloor - 1$, 若高度和 $< n - 1$；

所以当 $n = 2^k (k \in \mathbb{N})$ 时，高度和为 $n - 1$。

14.5

设置一个 k 个节点的最小堆，先将每个链表中的第一个元素存入最小堆，则此时堆顶元素一定是 n 个元素中的最小元素，存入答案链表，然后从堆顶元素所在原链表中取出第二个元素放入堆顶，然后维护最小堆，则此时堆顶元素一定是 n 个元素中第二小的元素，重复这个过程直到所有链表中元素全部被取出。算法伪代码如下两个函数：

Algorithm 14: heapify(heap[1..n])//维护最小堆

```
dad:=1; son:=1;
SWAP(heap[1],heap[2^k − 1]);
for i := 2 to k do
    dad:=1; son:=2dad;
    while son ≤ 2^k − i do
        if (son + 1 ≤ 2^k − i) and (heap[son] < heap[son + 1]) then
            son:=son+1;
        end
        if heap[dad] > heap[son] then
            break;
        else
            SWAP(heap[dad],heap[son]);
            dad=son; son=2dad;
        end
    end
    SWAP(heap[1],heap[2^k − i])
end
```

Algorithm 15: merge(*A[1..k])//合并

```
Input: *A[1..k]
Output: *B
cur:=B; num:=0;
heap[1..k]:=A[1..k];
for i:=k/2 to 0 do
    heapify(heap[i..k]);
while num<k do
    cur->data:=heap[1];
    cur=cur->next;
    if heap[1]->next then
        heap[1]:=heap[1]->next;
    else
        heap[1]:=heap[k-num];
        num:=num+1;
    heapify(heap[1..k-num]);
return B;
```

14.6

利用对顶堆，A 为最大堆，B 为最小堆，当元素数量为偶数时，A, B 元素数量相同，中位数为两堆顶元素的平均数，为奇数时，A 中元素数量比 B 中多 1，中位数为 A 的堆顶元素。

1) 插入操作：比较 A, B 元素数量，若元素数量相同，则插入 A 中，并对 A 进行维护；若 A 元素数量比 B 大，则插入 B 中，并对 B 进行维护；

2) 删除操作：先找到要删除的元素的位置，然后根据位置在哪个堆中进行对应的删除操作并维护，然后判断两堆元素数量，若 A 元素数量-B 元素数量 =0 或 1，则无需维护；若大于 1，则将 A 中堆顶元素删除并维护，然后将该元素插入 B 中并维护；若小于 0，则将 B 中堆顶元素删除并维护，然后将该元素插入 A 中并维护；

3) 找中位数操作：比较 A, B 元素数量，若元素数量相同，则中位数为两堆顶元素的平均数；若 A 元素数量比 B 大，则中位数为 A 的堆顶元素。