1. (a) 算法思路：考虑若邮资N可以完成兑换的话需要兑换几张5分面值的邮票，若为0张，说明N%3==0，若为1张，说明(N-5)%3==0，若为2张，说明(N-10)%3==0 (N≥10)，若N不满足N%3==0，(N-5)%3==0，(N-10)%3==0中的任何一条的话，说明不可能完成兑换。

伪代码：

```
bool ifcanchange (int N){
    if (N%3==0) return true;
    if(N>=5){
        if((N-5)%3==0) return true;
        if (N>=10){
            if((N-10)%3==0) return true;}
        }
    }
return false;}
```

(b) 证明：当N<3时，易知不可能完成兑换，ifconchange函数返回false，算法正确。

假设3<N<k (k∈Z, k>3)时算法均正确。

N=k时，若k%3==0，易知算法正确。

考虑k%3!=0的情况

根据算法思路中对5分面值邮票的归纳即可证出算法是正确的。

2.(a) 设命题P：对于$\forall k \in N$，对于满足本题条件的序列来说，$F(4k)$、$F(4k+3)$为偶数，$F(4k+1)$、$F(4k+2)$为奇数。

证明P为真：$k=0$时，由$F(0)=0$、$F(4)=F(0)+F(1)+F(2)=2$为偶数，$F(1)=1$、$F(2)=1$均为奇数可知P为真。

假设$k=n-1$时（$n>1$且$n \in Z$）P成立。

$k=n$时，$F(4n)=F(4n-1)+F(4n-2)+F(4n-3)$，$\because k=n-1$时P成立，$\therefore$有$F(4n-4)$、$F(4n-1)$为偶数，$F(4n-3)$、$F(4n-2)$为奇数，$\therefore F(4n)$为偶数。又由$F(4n+1)=F(4n)+F(4n-1)+F(4n-2)$可知$F(4n+1)$为奇数，同理可得$F(4n+2)$为奇数，$F(4n+3)$为偶数，$\therefore$ P为真。

由P成立可知$F(n)$为偶数时当且仅当$n=4k_1$（$k_1 \in N$）或$n=4k_2+3$（$k_2 \in N$）时，$n=4k_1$时 n%4==0，$n=4k_2+3$时 (n+1)%4==0，所以$F(n)$为偶数当且仅当n或者n+1能被4整除。

(b)
```
Hanoi(A,B,C,n)
  { Hanoi (A,B,C,n-1)
     mov A→C ;
     Hanoi (B,A,C,n-1)
     mov C→B;
  }
```

$T(n)=2T(n-1)+2$

$T(n)=O(2^n)$

3.(a) 假设 $\exists(i,j)$ 为逆序对，$j-i>2$，那么 $A[i]$ 与 $A[j]$ 之间至少隔了2个数，由于 $A[i]>A[j]$，

① 若 $A[i+1]>A[i]$，那么由 $A[i+1]>A[i]>A[j]$，$i+1<j$ 可知这是第2对逆序对。考虑 $A[i+2]$，若 $A[i+2]<A[i+1]$，则 $(i+1,i+2)$ 为第3对逆序对，与 $A[1\ldots n]$ 中至多有2个逆序对矛盾，若 $A[i+2]>A[i+1]$，则 $A[i+2]>A[j]$，则 $(i+2,j)$ 为第3对逆序对，与 $A[1\ldots n]$ 中至多有2个逆序对矛盾，$\therefore A[i+1]$ 不可能大于 $A[i]$。

② 若 $A[i+1]<A[i]$，那么 $(i,i+1)$ 是第2对逆序对，$A[i+2]$ 如果小于 $A[i]$，$(i,i+2)$ 为第3对逆序对与至多2个逆序对矛盾，$A[i+2]$ 如果大于 $A[i]$，$(i+2,j)$ 为第3对，矛盾。

$\therefore$ 原假设不成立，$\therefore$ 若 (i,j) 为逆序对，则 $j-i<=2$。

(b) 由于逆序对满足 $j-i\le 2$，$\therefore j-i=2$ 或 $j-i=1$

$j-i=2$ 时，两个逆序对一定是连在一起的，证明：$j-i=2$ 时，$A[i]$ 与 $A[j]$ 中间隔了1个数，$A[j]<A[i]$，若 $A[i+1]<A[i]$，则 $(i,i+1)$ 就是第2对逆序对，若 $A[i+1]>A[j]$，则 $(i+1,j)$ 是第2对逆序对，所以两个逆序对是相邻的。

因此只需从数组头部开始与每个元素相邻元素比较即可找出逆序对，找出一对 (i,j) 后再看一看 $(i-1,j)$ 是不是逆序对。然后把逆序对交换后即完成了排序（比较次数不超过 n）

4.(1) ①首先通过归并排序对A排序(从小到大排成A[1].A[2]…A[n])

②按照 A[1], A[n], A[2], A[n-1], A[3]…的顺序排好。

时间复杂度分析:①用了归并排序,时间复杂度为O(nlogn)

② 选择元素排列 nO(1) =O(n)

∴时间复杂度为 O(nlogn)

A[2n+1]

(2)①用最坏情况线性时间选择算法找到中位数(第n+1大的元素)。然后用PARTITION算法把中位数作为pivot,把比中位数小的分在左侧,比中位数大的放在右侧。

②按照 A[1]. A[n]. A[2], A[n-1]… A[n+1]的顺序排好(即从中位数左边取1个.再从中位数右边取1个…)。

5. 伪代码： (tree 是 Tree *) maxHeight (int)，restree (tree) 均为全局变量。
(初值为 0) (初值为 NULL)

```
void FindMax (tree t) {
    if (t==NULL) return;
    if (PerfectHeight(t) > maxHeight)
        maxHeight = PerfectHeight(t);
    FindMax (t->leftChild);
    FindMax (t->rightChild);
}

int PerfectHeight (tree t) {
    if (t==NULL) return 0;
    if (t->leftChild == NULL || t->rightChild == NULL) return 0;
    else {
        int l = PerfectHeight (t->leftChild);
        int r = PerfectHeight (t->rightChild);
        if (l==r)
            return l+1;
        else
            return min(l, r)+1; //去掉不完美的子树
    }
}
```

分析 PerfectHeight，$T(n) = T(left) + T(right) + O(1)$，最坏情况下复杂度为 $O(n)$

FindMax：$T(n) = T(left) + T(right) + O(n)$，与 QuickSort 的分析相似，最坏情况下复杂度为 $O(n\log n)$。

∴ 算法最坏情况下时间复杂度为 $O(n\log n)$

数组大小固定

6.(a) S集合用一个数组来实现(假设已知数据范围)
insert操作就是在当前数组的第1个空位上

```
int S[MAX];   int rear=0; //第1个空位
void insert (int x, int S[])
        S[rear]=x;
        rear++;
        if (rear ≥ max) cout<<"error"<<endl; }

void Remove-half (int S[]){
        int k=SELECT-WLINEAR(S, 0, rear, ⌈(rear+1)/2⌉);
        int q= PARTITION (A, 0, rear); //pivot 用k。
        for (int i=0; i≤ rear-⌈k/2⌉; i++){ S[i]=S[i+q]; rear--;}
    }
```

insert: $O(1)$

Remove-half: $O(n)$

(b) ∵ Remove-half操作的时间复杂度为$O(n)$,设$T_{Remove}=kn$

则:

	C_{act}	C_{acc}	C_{am}
Insert	1	k	k+1
Remove-half	k|S|	-k|S|	0

∵在insert的时候提前预支了Remove时的代价 ∴ C_{acc}的和大于0 ∴ 单次操作的均摊代价不超过 k+1 次 总时间复杂度的上界为 $O(n(k+1))=O(n)$