习题2-9

TC第7.1节练习2；TC第7.2节练习4；TC第7.3节练习2

TC第7.4节练习2；TC第7章问题4、5；TC第8.1节练习3、4

TC第8.2节练习4；TC第8.3节练习4；TC第8.4节练习2

TC第8章问题2；TC第9.1节练习1；TC第9.3节练习5、7

7.1-2

What value of q does PARTITION return when all elements in the array $A[p \,..\, r]$ have the same value? Modify PARTITION so that $q = \lfloor (p+r)/2 \rfloor$ when all elements in the array $A[p \,..\, r]$ have the same value.

```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
        do if A[j] ≤ x
                then i ← i + 1
                        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
if i + 1 = r return ⌊(p + r)/2⌋
    else  return i + 1
```

$A =< 1,2,3 \ldots r >$

7.1-2

What value of q does PARTITION return when all elements in the array $A[p\,..\,r]$ have the same value? Modify PARTITION so that $q = \lfloor (p+r)/2 \rfloor$ when all elements in the array $A[p\,..\,r]$ have the same value.

$$A = <1,2,3\ldots,0>$$

```
PARTITION(A,p,r)
  x=A[r];
  i=p-1;
  flag=false;
  for(j=p to r-1)
     if(A[j]<x) flag=true;
     if(A[j]<=x)
       i=i+1;
       exchange A[i] with A[j];
  if(flag==false)
     return (p+r)/2;
  else
     exchange A[i+1] with A[r]
     return i+1;
```

```
PARTITION(A, p, r)
  x ← A[r]
  i ← p − 1   c ← 0;
  for j ← p to r − 1   if A[j] = x  c ← c + 1;
        do if A[j] ≤ x
              then i ← i + 1
                   exchange A[i] ↔ A[j]
  exchange A[i + 1] ↔ A[r]
  if [c + 1 = r] return ⌊(p + r)/2⌋
     else return i + 1
```

7.3-2

When RANDOMIZED-QUICKSORT runs, how many calls are made to the random-number generator RANDOM in the worst case? How about in the best case? Give your answer in terms of Θ-notation.

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

The new quicksort calls RANDOMIZED-PARTITION in place of PARTITION:

```
RANDOMIZED-QUICKSORT(A, p, r)
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

参照快速排序的最坏情况,每次随机选到的$A[i]$都是最大的或者最小的,这样一共划分了n次,因此调用了$\Theta(n)$次.

如果每次选择的$A[i]$都是均值,每次的数组都被二分,这样只要调用$\Theta(\log(n))$次.

8.1-3

Show that there is no comparison sort whose running time is linear for at least half of the $n!$ inputs of length n. What about a fraction of $1/n$ of the inputs of length n? What about a fraction $1/2^n$?

Assume there is a comparison sort whose running time is linear for at least half of the n! inputs of length n.

考虑该算法的decision-tree;
则在该decision-tree中至少有n!/2个叶节点的level<=cn, c为一个常数
而高度为cn的二叉树最多具有2^{cn}个叶节点
所以，$\frac{n!}{2} \leq 2^{cn}$
两边取对数得：$\lg(n!) - 1 \leq cn$

$$\lg(n!) \leq cn - 1$$
$$\lg(n!) = O(n)$$

而我们已知$\lg(n!) = \Theta(n\lg n)$
矛盾，假设不成立

8.3-4

Show how to sort n integers in the range 0 to $n^3 - 1$ in $O(n)$ time.

如果n^2-1有k位,那么我们先对最后一位排序,由于可以直接记录0 9的个数,这样的排序可以在线性时间内完成,,接着按照第$k-1$位排序,...,最后排第一位,这样一共进行了$\mathbf{O}(kn)$次排序,由于k是固定的常数,所以$\mathbf{O}(kn)=\mathbf{O}(k)$

We find d=lg(n^3)

Then we use d-digits radix sort

We need time O(d(n+k))=O(n)

将数字转换成 n 进制，可知最大的位数为 2 位，因此我们可将其看作是在 n 进制下 2 位数

的排序，其中每个数位有 n 个可能的取值，因此根据引理 8.3，可知耗时为 Θ (2(n+n))= Θ (4n)

可知时间复杂度为 O(n)。

8-2 Sorting in place in linear time $\langle \text{key}, value \rangle$

Suppose that we have an array of n data records to sort and that the key of each record has the value 0 or 1. An algorithm for sorting such a set of records might possess some subset of the following three desirable characteristics:

1. The algorithm runs in $O(n)$ time.
2. The algorithm is stable.
3. The algorithm sorts in place, using no more than a constant amount of storage space in addition to the original array.

a. Give an algorithm that satisfies criteria 1 and 2 above. **Counting Sort**

b. Give an algorithm that satisfies criteria 1 and 3 above.

c. Give an algorithm that satisfies criteria 2 and 3 above. **Bubble Sort/Insertion Sort**

d. Can you use any of your sorting algorithms from parts (a)–(c) as the sorting method used in line 2 of RADIX-SORT, so that RADIX-SORT sorts n records with b-bit keys in $O(bn)$ time? Explain how or why not.

e. Suppose that the n records have keys in the range from 1 to k. Show how to modify counting sort so that it sorts the records in place in $O(n+k)$ time. You may use $O(k)$ storage outside the input array. Is your algorithm stable? (*Hint:* How would you do it for $k = 3$?)

1. The algorithm runs in $O(n)$ time.
2. The algorithm is stable.
3. The algorithm sorts in place, using no more than a constant amount of storage space in addition to the original array.

b. Give an algorithm that satisfies criteria 1 and 3 above.

Key只有0,1两种选择　　　　回想quit-sort!

方案一：

在A中最后添加一项key=0.5，直接PARTITION，但不交换最后一次key

方案二：

```
SORT-IN-PLACE(A, l, r)
while l < r
        do while A[l] = 0
                do l++
           while A[r] = 1
                do r--
           swap A[l] and A[r]
```

e. Suppose that the n records have keys in the range from 1 to k. Show how to modify counting sort so that it sorts the records in place in $O(n+k)$ time. You may use $O(k)$ storage outside the input array. Is your algorithm stable? (*Hint:* How would you do it for $k = 3$?)

```
COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

```
i = 0;
Copy C as D
While(i < n)
    if( D[A[i] − 1] <= i < D[A[i]])
        i ← i + 1;
    else
        swap(A[i], A[C[A[i]] − 1]);
        C[A[i]] ← C[A[i]] − 1;
```