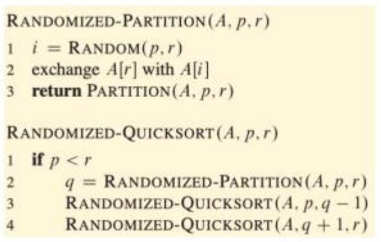
A4 - 毛九弢 - 221900175

1 Alternative quicksort analysis

(From CLRS 7-3)



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

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

An alternative analysis of the running time of randomized quicksort focuses on the expected running time of each individual recursive call to `RANDOMIZED-QUICKSORT`, rather than on the number of comparisons performed. As in the analysis of Section 7.4.2, assume that the values of the elements are distinct.

1. Argue that, given an array of size n, the probability that any particular element is chosen as the pivot is $1/n$. Use this probability to define indicator random variables $X_i = I\{i\text{th smallest element is chosen as the pivot}\}$. What is $\mathbb{E}[X_i]$?

$$P(X_i = 1) = \frac{1}{n}$$

$$\therefore E[X_i] = 0 \times \frac{n-1}{n} + 1 \times \frac{1}{n} = \frac{1}{n} = P(X_i = 1)$$

2. Let $T(n)$ be a random variable denoting the running time of quicksort on an array of size n. Argue that

$$\mathbb{E}[T(n)] = \mathbb{E}\left[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))\right].$$

$$\mathrm{E}[\mathrm{T}(n)] = \sum_{q=1}^{n} P(X_q = 1)(\mathrm{E}[\mathrm{T}(q-1)] + \mathrm{E}[\mathrm{T}(n-q)] + \Theta(n))$$

$$= \sum_{q=1}^{n} \mathrm{E}[X_q]\left(\mathrm{E}[\mathrm{T}(q-1)] + \mathrm{E}[\mathrm{T}(n-q)] + \Theta(n)\right)$$

$$= \mathrm{E}\left[\sum_{q=1}^{n} X_q\left(\mathrm{T}(q-1) + \mathrm{T}(n-q) + \Theta(n)\right)\right]$$

3. Show how to rewrite equation(7.2) as

$$\mathbb{E}[T(n)] = \frac{2}{n}\sum_{q=1}^{n-1}\mathbb{E}[T(q)] + \Theta(n). \qquad (7.3)$$

$$\mathrm{E}[\mathrm{T}(n)] = \sum_{q=1}^{n} P(X_q = 1)(\mathrm{E}[\mathrm{T}(q-1)] + \mathrm{E}[\mathrm{T}(n-q)] + \Theta(n))$$

$$= \frac{1}{n}\left(\sum_{q=1}^{n} \mathrm{E}[\mathrm{T}(q-1)] + \sum_{q=1}^{n} \mathrm{E}[\mathrm{T}(n-q)] + \sum_{q=1}^{n} \Theta(n)\right)$$

$$= \frac{2}{n}\sum_{q=0}^{n-1}\left(\mathrm{E}[\mathrm{T}(q)] + \Theta(n)\right)$$

4. Show that

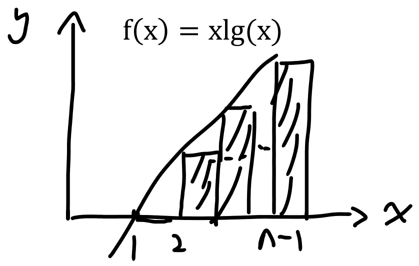
$$\sum_{q=1}^{n-1} q\lg q \le \frac{n^2}{2}\lg n - \frac{n^2}{8}. \qquad (7.4)$$

for $n \ge 2$. (*Hint:* Split the summation into two parts, one summation for $q = 1, 2, \ldots, \lceil n/2 \rceil - 1$ and one summation for $q = \lceil n/2 \rceil, \ldots, n-1$.)

$$\sum_{q=1}^{n-1} q\lg q \le \int_1^n x\lg x\, dx = \frac{1}{2}\int_1^n \lg x\, dx^2$$

$$= \frac{1}{2}x^2\lg x \Big|_{x=1}^{n} - \frac{\int_1^n x^2\, d\lg x}{2}$$

$$= \frac{n^2}{2}\lg n - \frac{n^2}{4} + \frac{1}{4}$$



$\because n \ge 2$

$$\therefore \sum_{q=1}^{n-1} q\lg q \le \frac{n^2}{2}\lg n - \frac{n^2}{8}$$

5. Using the bound from equation (7.4), show that the recurrence in equation (7.3) has the solution $\mathbb{E}[T(n)] = O(n\lg n)$. (*Hint:* Show, by substitution, that $\mathbb{E}[T(n)] \le an\lg n$ for sufficiently large n and for some positive constant a.)

Let's guess: $\mathrm{E}[\mathrm{T}(n)] = O(n\lg(n))$,
and assume $E[T(n)] \le an\lg n$;

$$\because\ \mathrm{E}[\mathrm{T}(n)] = \frac{2}{n}\sum_{q=0}^{n-1}\left(\mathrm{E}[\mathrm{T}(q)] + \Theta(n)\right),\ \sum_{q=1}^{n-1} q\lg q \le \frac{n^2}{2}\lg n - \frac{n^2}{8}$$

$$\therefore \mathrm{E}[\mathrm{T}(n)] \le \frac{2}{n}\sum_{q=0}^{n-1}\left(aq\lg q + \Theta(n)\right) \le \frac{2a}{n}\left(\frac{n^2}{2}\lg n - \frac{n^2}{8}\right) + \frac{2}{n}n\Theta(n)$$

$$= an\lg n - \frac{an}{4} + \Theta(n)$$

Given a specific $\Theta(n)$, we can find a specific c_0, N_0, s.t. $\forall n \ge N_0, \Theta(n) \le c_0 n$;
$\therefore$ Let $a \ge 4c_0$, we have:
$\forall n \ge N_0,\ \mathrm{E}[\mathrm{T}(n)] \le an\lg n - \frac{4c_0 n}{4} + \Theta(n) \le an\lg n$

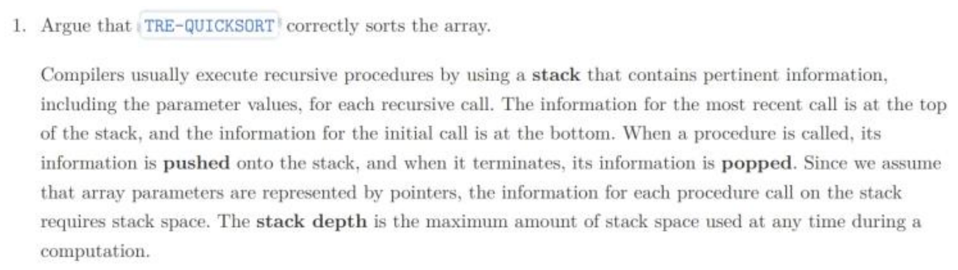
2 Stack depth for quicksort

(From CLRS 7-5)

```
QUICKSORT(A, p, r)
if p < r
    // Partition the subarray around the pivot, which ends up in A[q].
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)  // recursively sort the low side
    QUICKSORT(A, q + 1, r)  // recursively sort the high side
```

The QUICKSORT procedure of Section 7.1 makes two recursive calls to itself. After QUICKSORT calls PARTITION, it recursively sorts the low side of the partition and then it recursively sorts the high side of the partition. The second recursive call in QUICKSORT is not really necessary, because the procedure can instead use an iterative control structure. This transformation technique, called **tail-recursion elimination**, is provided automatically by good compilers. Applying tail-recursion elimination transforms QUICKSORT into the TRE-QUICKSORT procedure.

```
TRE-QUICKSORT(A, p, r)
while p < r
    // Partition and then sort the low side.
    q = PARTITION(A, p, r)
    TRE-QUICKSORT(A, p, q − 1)
    p = q + 1
```


1. Argue that TRE-QUICKSORT correctly sorts the array.

Compilers usually execute recursive procedures by using a **stack** that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is called, its information is **pushed** onto the stack, and when it terminates, its information is **popped**. Since we assume that array parameters are represented by pointers, the information for each procedure call on the stack requires stack space. The **stack depth** is the maximum amount of stack space used at any time during a computation.

称命题P(N)为TRE-QUICKSORT能使长度为N的数组有序。

因为p,r的绝对大小并不影响,我们只关心数组的长度，且N=p-r+1。

P(1)P(2)显然成立

假设P(1)...P(k)均成立,下证P(k+1)成立,

称第i次循环的p,q,r为p_i q_i r_i

易得第一个循环不变式：$\forall i, j$: $r_j = r_i$

在第3行我们有：$p_i \le q_i \le r_i$，

且因为第4行有循环不变式:

在第i次循环结束后数组在$[p_i, q_i]$是有序的（归纳，因为长度小于k+1），

又因第5行我们有: $p_{i+1} = q_i + 1$, 故有$p_i < p_{i+1}$,故而易得循环的终止性,

不妨设第m次循环后跳出

而结合在第i次循环结束后数组在$[p_i, q_i]$是有序的不变式,

我们得到另一个不变式：在第i次循环结束后数组在$[p_1, q_i]$,

而由于循环的跳出条件为$p_{m+1} > r_{m+1} = r_m$,

又因为 $p_{m+1} = q_m + 1, q_m \le r_m$

故有：$q_m + 1 > r_{m+1} = r_m \ge q_m$

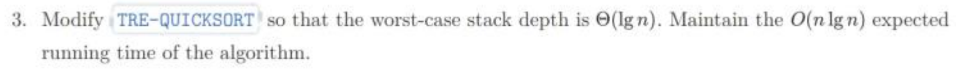
我们得到：$r = r_1 = r_m = q_m$

故循环跳出时，数组在$[p_1, r_1]$上有序

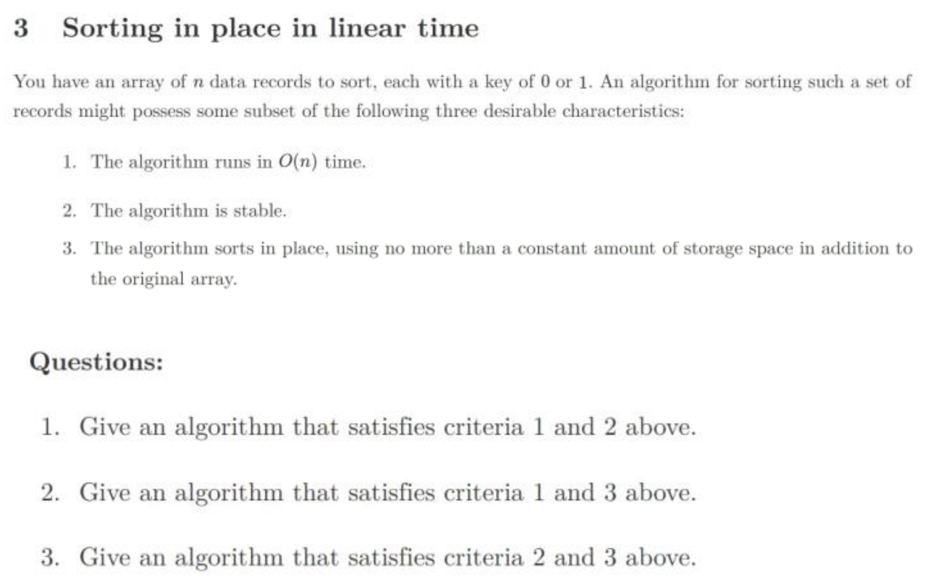
归纳完成。

2. Describe a scenario in which TRE-QUICKSORT's stack depth is $\Theta(n)$ on an n-element input array.

每次partition得到的q = r;


3. Modify TRE-QUICKSORT so that the worst-case stack depth is $\Theta(\lg n)$. Maintain the $O(n \lg n)$ expected running time of the algorithm.

```
TRE − QUICKSORT(A, p, r)
while $p < r$
    $q = \text{PARTITION}(A, p, r)$
    if $q < \frac{p+r}{2}$
        TRE − QUICKSORT(A, p, q − 1)
        $p = q + 1$
    else
        TRE − QUICKSORT(A, q + 1, r)
        $r = q - 1$
```



3 Sorting in place in linear time

You have an array of n data records to sort, each with a key of 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.

Questions:

1. Give an algorithm that satisfies criteria 1 and 2 above.
2. Give an algorithm that satisfies criteria 1 and 3 above.
3. Give an algorithm that satisfies criteria 2 and 3 above.

Question1. 1&2:

使用一个数组arr装0，一个先进先出队列que装1，

从前向后遍历，遍历完将que中的1全部放入数组

```
ans[n]
que
size:=0
for num in arr:
    if num == 0:
        size++
        arr[size]=0;
    else:
        que.push(1)
while que is not empty:
    size++
    arr[size] = que.front();
    que.pop();
```

Question1. 1&3:

从前向后遍历,如果是1就继续,

如果是0，就和最前面那个1交换位置，

```
i:=1
p:=1
while true:
    while i  <= n and arr[i] == 1 and :
        i++
    if i > n:
        break
    if p < i:
        swap(arr[i],arr[p])
    p++,i++
```

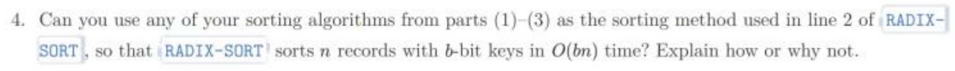
Question1. 2&3:

从前向后遍历,如果是1就继续,

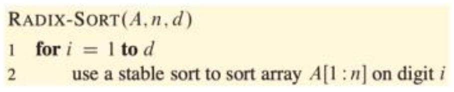
如果是0，且前一个不是0就一直与前一个交换位置，

```
i:=1
p:=1
```

```
while true:
    while i  <= n and arr[i] == 1 :
        i++
    if i > n: break
    for ( j:=i; j>=p+1; j--):
        swap(arr[j-1],arr[j])
    i++,p++
```



4. Can you use any of your sorting algorithms from parts (1)–(3) 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.



```
RADIX-SORT(A, n, d)
1  for i = 1 to d
2      use a stable sort to sort array A[1 : n] on digit i
```

这里我们需要一个O(bn)且stable的算法

显然（2）不是stable，不行；

而（3）是一个O(n^2)不是O(bn)，不行；

而（1）满足既是stable又是O(bn)，可行