

论题 2-7 作业

姓名：陈劭源

学号：161240004

1 [TC] Problem 7.1-2

The procedure returns r when all elements in the array have the same value.

MODIFIED-PARTITION(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8   $j = i$ 
   // Note that the operator 'and' is the short-circuit version!
9  while  $j \geq p$  and  $A[j] = A[i]$ 
10      $j = j - 1$ 
11 return  $\lfloor (i + j) / 2 \rfloor$ 
```

2 [TC] Problem 7.2-4

We have already proved before that the asymptotically running time of INSERTION-SORT is $O(n + I)$, where n is the length of the input array and I is the number of inversions in the input array. There are not too many inversions in an almost-sorted array, so the procedure INSERTION-SORT performs well for an almost-sorted array. However, in QUICKSORT, if we use the first element as a pivot to partition an almost-sorted array, the split is quite unbalanced. Even if we use the randomized version of quicksort, the best running time is $O(n \lg n)$. Therefore, the procedure INSERTION-SORT would tend to beat the procedure QUICKSORT on this problem.

3 [TC] Problem 7.3-2

The number of calls to the number generator equals to the number of calls to the procedure PARTITION.

Best case: $\Theta(n)$

Worst case: $\Theta(n)$

4 [TC] Problem 7.4-2

Let $T(n)$ denote the best-case time for the procedure QUICKSORT on an input of size n . We have the recurrence

$$T(n) = \min_{0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

We guess that $T(n) \geq cn \lg n$ for some constant c . Substituting it into the recurrence, we obtain

$$\begin{aligned} T(n) &\geq \min_{0 \leq q \leq n-1} c(q \lg q + (n-q-1) \lg(n-q-1)) + \Theta(n) \\ &\geq 2c \frac{n-1}{2} \lg \frac{n-1}{2} + \Theta(n) \\ &= c \frac{n-1}{\lg} \frac{n-1}{2} + \Theta(n) \\ &= cn \lg n + cn \lg \frac{n-1}{n} - c \lg(n-1) - c(n-1) + \Theta(n) \\ &\geq cn \lg n - cn - cn - cn + \Theta(n) \\ &= cn \lg n - 3cn + \Theta(n) \\ &\geq cn \lg n \end{aligned}$$

The second inequality is obtained by differentiating with respect to q . The third inequality holds when $n > 2$. Assuming that $\Theta(n) \geq c_0 n$ for sufficiently large n , then the last inequality holds if we choose $c < c_0/3$.

Therefore, the best-case running time of quicksort is $\Omega(n \log n)$.

5 [TC] Problem 7.4

a. We use mathematical induction to prove the correctness of the algorithm.

For the base step, the length of array is 0 or 1, i.e. $p \geq r$, the procedure does nothing and the array is sorted.

For the induction step, assume that the process sorts any array of length $k < n$ correctly. For array of length n , the procedure partitions the array into two parts, and every element in the left subarray is less than or equal to every element in the right subarray. Then the procedure recursively calls it self, making the left subarray sorted, by induction hypothesis. The assignment $p = q + 1$ updates the parameters of the procedure, and then it jumps to line 1, in order to sort the right subarray, and by induction hypothesis, the right subarray can be sorted correctly. Since every element in the left subarray is less than or equal to every element in the right subarray, the whole array is sorted.

By mathematical induction, TAIL-RECURSIVE-QUICKSORT($A, 1, A.length$) correctly sorts the array A .

b. When the procedure PARTITION always produces a left subarray of length $n - 1$ and a right subarray of length 0, the procedure TAIL-RECURSIVE-QUICKSORT will be recursively called for n times, and the stack depth is $\Theta(n)$.

- c. Always let the recursive call sort the smaller subarray. Since the length of the smaller subarray is less than half of the length of the original array, the worst-case stack depth is $\Theta(\lg n)$.

MODIFIED-RECURSIVE-QUICKSORT(A, p, r)

```

1  while  $p < r$ 
2       $q = \text{PARTITION}(A, p, r)$ 
3      if  $q < (p + r)/2$ 
4          TAIL-RECURSIVE-QUICKSORT( $A, p, q - 1$ )
5           $p = q + 1$ 
6      else
7          TAIL-RECURSIVE-QUICKSORT( $A, q + 1, r$ )
8           $r = q - 1$ 

```

6 [TC] Problem 7.5

a.

$$p_i = (i-1)(n-i) / \binom{n}{3} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$$

b.

$$\begin{aligned}
 \lim_{n \rightarrow \infty} p_{\lfloor (n+1)/2 \rfloor} / (1/n) &= \lim_{n \rightarrow \infty} \frac{6(\lfloor (n+1)/2 \rfloor - 1)(n - \lfloor (n+1)/2 \rfloor)}{(n-1)(n-2)} \\
 &= \lim_{n \rightarrow \infty} \frac{6(n-1)^2}{4(n-1)(n-2)} \\
 &= 3/2
 \end{aligned}$$

So the likelihood is improved by 50% compared with the ordinary implementation.

c.

$$\begin{aligned}
 \sum_{i=n/3}^{2n/3} p_i / (n/3) &= (3/n) \sum_{i=n/3}^{2n/3} \frac{6(i-1)(n-i)}{n(n-1)(n-2)} \\
 &\approx \frac{18}{n(n-1)(n-2)} \int_{n/3}^{2n/3} (-i^2 + (1+n)i - n) di \\
 &= \frac{(13n-27)n}{9(n-1)(n-2)} \\
 \lim_{n \rightarrow \infty} \sum_{i=n/3}^{2n/3} p_i / (n/3) &= \frac{13}{9} \approx 1.44
 \end{aligned}$$

So the likelihood of getting a good split is improved by 44.4% compared with the ordinary implementation when $n \rightarrow \infty$.

- d. For median-of-3 method, the best case is still that PARTITION produces two subarray of the same size, and it takes a running time of $\Theta(n \lg n)$. Therefore, the running time of the median-of-3 method is still $\Omega(n \lg n)$, i.e. it only affects the constant factor.

7 [TC] Problem 8.1-3

Assume there exists such algorithm. Consider a decision tree of this algorithm, operating on half of the $n!$ inputs of length n such that the running time of the algorithm is linear. This decision tree has $n!/2$ leaves, and its height is at least $\lg(n!/2)$, i.e. $\Theta(n \log n)$, by the property of a binary tree. However, the algorithm sorts the $n!/2$ inputs in a linear time. That means, there exists N , for every $n > N$, the number of the comparisons the algorithm make is less than the height of the decision tree, which leads to a contradiction.

For a fraction of $1/n$ and $1/2^n$, the corresponding minimum heights of the decision trees are $\lg((n-1)!/2)$ and $\lg(n!/2^n)$, i.e. $\Theta(n \log n)$ and $\Theta(n \log n)$, which are still impossible. Therefore, such algorithm does not exist.

8 [TC] Problem 8.1-4

In this problem, there are $(k!)^{n/k}$ possible inputs, so the height of the decision tree is at least $\lg(k!)^{n/k}$, i.e. $\Theta(n \lg k)$, as n and k grow large. That means, at least $\Theta(n \lg k)$ comparisons are needed to solve this problem. Therefore, the lower bound on the number of comparisons is $\Omega(n \lg k)$.

9 [TC] Problem 8.2-4

10 [TC] Problem 8.3-4

11 [TC] Problem 8.4-2

12 [TC] Problem 8.2

13 [TC] Problem 9.1-1

14 [TC] Problem 9.3-5

15 [TC] Problem 9.3-7