Lab01-Algorithm Analysis

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2021.

∗ Name: Haoyi You   Student ID: 519030910193   Email: yuri-you@sjtu.edu.cn

1. *Complexity Analysis.* Please analyze the time and space complexity of Alg. 1 and Alg. 2.

Algorithm 1: QuickSort

Input: An array $A[1, \cdots, n]$
Output: $A[1, \cdots, n]$ sorted nondecreasingly

$pivot \leftarrow A[n]$; $i \leftarrow 1$;
for $j \leftarrow 1$ **to** $n-1$ **do**
  if $A[j] < pivot$ **then**
    swap $A[i]$ and $A[j]$;
    $i \leftarrow i+1$;
swap $A[i]$ and $A[n]$;
if $i > 1$ **then** QuickSort($A[1, \cdots, i-1]$);
if $i < n$ **then** QuickSort($A[i+1, \cdots, n]$);

Algorithm 2: CocktailSort

Input: An array $A[1, \cdots, n]$
Output: $A[1, \cdots, n]$ sorted nonincreasingly

$i \leftarrow 1$; $j \leftarrow n$;$sorted \leftarrow false$;
while ***not*** *sorted* **do**
  $sorted \leftarrow true$;
  for $k \leftarrow i$ ***to*** $j-1$ **do**
    if $A[k] < A[k+1]$ **then**
      swap $A[k]$ and $A[k+1]$;
      $sorted \leftarrow false$;
  $j \leftarrow j-1$;
  for $k \leftarrow j$ ***downto*** $i+1$ **do**
    if $A[k-1] < A[k]$ **then**
      swap $A[k-1]$ and $A[k]$;
      $sorted \leftarrow false$;
  $i \leftarrow i+1$;

(a) Fill in the blanks and **explain** your answers. You need to answer when the best case and the worst case happen.

Algorithm	Time Complexity[1]	Space Complexity
$QuickSort$	$\Omega(nlog(n))$ $O(nlog(n))$ $O(n^2)$	$O(log(n))$(worst case $O(n)$)
$CocktailSort$	$\Omega(n)$ $O(n^2)$ $O(n^2)$	$O(1)$

[1] The response order can be given in *best*, *average*, and *worst*.

(b) For Alg. 1, how to modify the algorithm to achieve the same expected performance as the **average** case when the **worst** case happens?

Solution.

(a):

QuickSort:
1)Time Complexity
Best case: $\Omega(nlog(n))$.

The best case happens when $A[n]$ is the $\lceil \frac{n}{2} \rceil^{th}$ largest number in the array.Then when the program enters and exits the loop,it at least needs $2*(n-1)$ operations.Adding assignment operation before loop and swap operations after loop,there are at least $2n$ operations. So one recursion costs 2 times length of the array. The length of next recursion is half of this time's. So the total amount of comparison will be $2n + \lceil\frac{2n}{2}\rceil + \lceil\frac{\lceil\frac{2n}{2}\rceil}{2}\rceil ... = \Omega(nlog(n))$

Average case: $O(nlog(n))$.

Assume the expectation of numbers of operations of n-length array in quicksort is $f(n)$. From the best case we know in one recursion cost $2n$ operations. So we know

$$
\begin{aligned}
f(n) &= 2n + E[next\ recursions] \\
&= 2n + (\sum_{i=0}^{n-1} \frac{f(i) + f(n-1-i)}{2})/n \\
&= 2n + (\sum_{i=0}^{n-1} f(i))/n.
\end{aligned}
\tag{1}
$$

And $f(0) = 0$,from the equation 1,we can solve the result $f(n) = O(nlog(n))$ by induction.If $f(k) = O(klog(k))$, assume $d * (klog(k)) \leq f(k) \leq c * (klog(k))$ for $k \leq n-1$,and $n \geq 10$

$$
\begin{aligned}
&(\sum_{i=0}^{n-1} f(i))/n \leq (\sum_{i=0}^{n-1} c * i * log(i))/n \leq (\sum_{i=0}^{n-1} c * i * log(n))/n \leq c * (\frac{n}{2}log(n)) \\
\Rightarrow &f(n) = 2n + (\sum_{i=0}^{n-1} f(i))/n \leq 2n + c * (\frac{n}{2}log(n)) \leq c * \frac{n}{2}log(n) + c * (\frac{n}{2}log(n)) \\
&= c * nlog(n)
\end{aligned}
\tag{2}
$$

. And we know $(nln(n))^{(2)} = \frac{1}{n} > 0$,so

$$
\begin{aligned}
&i * log(i) + (n-i)log(n-i) \geq 2 * (\frac{n}{2})log(\frac{n}{2}) \\
\Rightarrow &(\sum_{i=0}^{n-1} f(i))/n) \geq \frac{n}{2}log(\frac{n}{2}) \\
\Rightarrow &f(n) = 2n + (\sum_{i=0}^{n-1} f(i))/n) \geq 2n + d * (\frac{n}{2}log(\frac{n}{2})) \geq d * nlog(n)
\end{aligned}
\tag{3}
$$

So from equation 2,4,as long as we choose suitable c and d,there are $O(nlog(n)) \leq O(nlog(n)) \leq O(nlog(n))$.Therefore, $f(n) = O(nlog(n))$

Worst case: $O(n^2)$.

The worst case happens when the array is ordered from 1 to n.Then for the proof above,in each recursion,it cost 2n operations,and then the next will have n-1 elements.The total amount of operations will be $\sum_{i=1}^{n} 2n = n * (n+1) = O(n^2)$

2) Space Complexity:
It only use 3 values $i, j, pivot$ in one recursion.However,the depth of recursion is $logn$ in average and best case, and n in worst case.So the space comlexity is $O(logn)$ in average and best case,and $O(n)$ in worst case.

CocktailSort:
1)Time Complexity:
Best case:$\Omega(n)$
The best case happens when the array is already sorted. Then it only need to enter the loop

once and scan the whole array twice(from 1 to n-1 and from n-1 to 2),then the total operations are $\Omega(n)$.
Average case:$O(n^2)$
The time depends on the times of entering the loop, and the k^{th} time entering the loop cost about $2(n+2-2k)-1$ operations.Assume the expectation of numbers of operations are $f(n)$. Firstly,it at most enters the loop for $\lceil \frac{n}{2} \rceil$ times, because this time $i \leq j$ .As a result,
$f(n) \leq \frac{(n+1)*n}{2} = O(n^2)$
Meanwhile,let the $g(n) = \sum_{i=1}^{n} |A[i] - i|$ .So it is obviously that $g(n) \leq f(n) * 2$,for after one swap $g(n)$ at most minus 2.
As for $g(n)$,assume there is an array of $n-1$ elements in a random order.We extend this array to n elements by appending the element n at the end of the previous array.Then we random sway the element n and a random element from 1 to n(we can sway n with itself).After that,the new array of n is still in a random order. If we choose the element in first $n-1$ place,the possibility is $\frac{n-1}{n}$. Assume we choose the element i in j^{th} place. So we have the recurrence formula below.

$$
\begin{aligned}
g(n) &= g(n-1) + \frac{n-1}{n} * \frac{1}{(n-1)^2} * (\sum_{i=1}^{n-1}\sum_{j=1}^{n-1} n - i + n - j - |i-j|) \\
&= g(n-1) + \frac{1}{(n-1)n} * [(n-1)^2 * 2n - 2 * (n-1) * \frac{n*(n-1)}{2} - \sum_{i=1}^{n-1}\sum_{j=1}^{n-1} |i-j|) \\
&= g(n-1) + n - 1 - \frac{1}{(n-1)n} * (2 * \sum_{i=1}^{n-1} \frac{i*(i-1)}{2}) \\
&= g(n-1) + n - 1 - \frac{1}{(n-1)n} * \frac{(n-1)*n*(n-2)}{3} \\
&= g(n-1) + \frac{2n-1}{3}
\end{aligned} \tag{4}
$$

We have known $g(1) = 0, g(2) = 1, g(3) = \frac{8}{3}$,from the recursion formula 4 we can get

$$
g(n) = g(1) + \sum_{i=2}^{n} \frac{2n-1}{3} = \frac{n^2-1}{3} \tag{5}
$$

So $f(n) \geq g(n)/2 = \frac{n^2-1}{6} = O(n^2)$.And we have proved $f(n) \leq O(n^2)$.
So $f(n) = O(n^2)$
Worst case:$O(n^2)$
The worst case happens when the array is reverse ordered.Then we have to enter the loop for $\lceil \frac{n}{2} \rceil$ times.So the number of operations is $O(n^2)$.
2) Space Complexity:
It only use 3 values $i, j, sorted$,so it is $O(1)$

(b):
For Alg. 1,the worst case originate from the situation that chosen element assigned to *pivot* is fixed to be $A[n]$. So as long as the array is almost ordered or reverse ordered, every time the recursion can only shorten the length of the array 1 element rather than almost half of the elements.
Aiming at the shortcomings of the algorithm, it should change the assignment element. Each

recursion it need assign a random element to *pivot* rather than the last element of the array. Just change the first line to

$$pivot \leftarrow A[random(1 \ to \ n)]; i \leftarrow 1; \tag{6}$$

□

2. *Growth Analysis.* Rank the following functions by order of growth with brief explanations: that is, find an arrangement $g_1, g_2, \ldots, g_{15}$ of the functions $g_1 = \Omega(g_2), g_2 = \Omega(g_3), \ldots, g_{14} = \Omega(g_{15})$. Partition your list into equivalence classes such that functions $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. Use symbols "=" and "$\prec$" to order these functions appropriately. Here $\log n$ stands for $\ln n$.

1	n	$\log n$	$\log(\log n)$	$n \log n$
$\log_4 n$	2^n	4^n	$2^{\log n}$	2^{2^n}
$\log(n!)$	$n!$	$(2n)!$	$n^{1/2}$	n^2

Solution.

$$\begin{aligned} 1 \prec log(log\ n) \prec log_4\ n = log\ n \prec n^{\frac{1}{2}} \prec 2^{log\ n} \prec n \\ \prec log(n!) = n\ log\ n \prec n^2 \prec 2^n \prec 4^n \prec n! \prec (2n)! \prec 2^{2^n} \end{aligned} \tag{7}$$

□