Lab01-Algorithm Analysis

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

∗ If there is any problem, please contact TA Haolin Zhou. Also please use English in homework.

∗ Name:Yanjie Ze   Student ID: 519021910706   Email: zeyanjie@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	$\Omega(nlogn)$,$O(nlogn)$,$O(n^2)$	$O(logn)$
CocktailSort	$\Omega(n), O(n^2), O(n^2)$	O(1)

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

Solution.
For *QuickSort*:
Best Case: $\Omega(nlogn)$
Appears when every time the pivot separates the array into two equally-sized subarrays.

$$T(n) = \sum_{j=1}^{logn} \frac{n}{2^j} \times 2^j = nlogn$$

Worst Case: $O(n^2)$

Appears when every time the pivot always separates the array into 1 and n-1 sized subarrays. In this situation, quick sort will just modify one element's position in each loop.

$$T(n) = \sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

Average Case: $O(nlogn)$

Suppose the ground truth order for an n-element array is $\{a_1, a_2, ..., a_n\}$, and the probability for each element selected as a pivot is equal. Then we have the following formula:

$$T(n) = \frac{1}{n} \sum_{i=1}^{n} [T(i) + T(n-i) + n]$$

Thus we can get:

$$nT(n) = n^2 + 2 \sum_{i=1}^{n-1} T(i)$$

$$(n+1)T(n+1) = (n+1)^2 + 2 \sum_{i=1}^{n} T(i)$$

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

After simplification we get:

$$\frac{T(n+1)}{n+2} = \frac{T(n)}{n+1} + \frac{2n+1}{(n+1)(n+2)}$$

Obviously we can get the comparison:

$$\frac{T(n)}{n+1} + \frac{1}{n+1} \leq \frac{T(n+1)}{n+2} \leq \frac{T(n)}{n+1} + \frac{2}{n+1}$$

From the left side we do inference:

$$\frac{T(n+1)}{n+2} \geq 1 + \frac{1}{2} + ... + \frac{1}{n+1} = \sum_{i=1}^{n+1} \frac{1}{i}$$

The result we get is known as **the harmonic series**. Suppose n is big enough.Thus:

$$\frac{T(n+1)}{n+2} \geq log(n+1) + \gamma + \epsilon,$$

where γ is the Euler–Mascheroni constant and $\epsilon \sim \frac{1}{2(n+1)}$which approaches 0 as n+1 goes to infinity.

Similarly ,we get another comparison relation from the right side:

$$\frac{T(n+1)}{n+2} \leq 2log(n+1) + 2\gamma + 2\epsilon$$

Finally:

$$(n+2)[log(n+1) + \gamma + \epsilon] \leq T(n+1) \leq 2(n+2)[log(n+1) + \gamma + \epsilon]$$

Therefore, the average case is $O(nlogn)$.

Space Complexity: $O(logn)$

This is because quick sort algorithm needs to use a stack to record the recursion result. And the stack's height is the same as the recursion tree's height, which is $logn$.

For $CocktailSort$:

Best Case: $\Omega(n)$

Appears when one loop ends, the flag *sorted* is true. This means the array has been sorted in the beginning.

$$T(n) = (n-1) + (n-2) = 2n - 3$$

Worst Case: $O(n^2)$

Appears when the array is originally sorted increasingly. In this case, the flag *sorted* is always turned into $false$ in each outer loop, until the array is sorted.

$$T(n) = (n-1) + (n-2) + (n-3) + ... + 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

Average Case: $O(n^2)$

We consider a pair $P =< A, B >$. In $CocktailSort$, the time cost is mainly from the inversion of a pair.We denote $Prob(A, B)$ as the probability of inversion of A and B.

Essentially, in the best case $Prob(A[k], A[k+1]) = 0$ and in the worst case $Prob(A[k], A[k+1]) = 1$.

Therefore, we assume that $Prob(A[k], A[k+1]) = 0.5$ in the average case.

Thus, In each outer loop, the times of inversion will be the half of that in the worst case. Based on this, we get:

$$T(n) = \frac{1}{2}[(n-1) + (n-2) + (n-3) + ... + 1] = \frac{1}{2}\sum_{i=1}^{n-1} i = \frac{n(n-1)}{4}$$

Finally, the average case is $O(n^2)$.

Space Complexity: $O(1)$

The extra space is for $i, j, sorted$, thus the space complexity is $O(1)$. □

(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.

The worst case is caused by the bad choice of pivot. During the worst case, the array is sorted originally, which made QuickSort fails ti divide and conquer. Therefore, to improve the worst case, we can use a new way to choose an efficient pivot.

This way can be random choice or median number choice.

In our new improved algorithm, we use the random choice.

Algorithm 3: QuickSortImproved

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

$n_{pivot} = RANDOM(1, n)$;
$pivot \leftarrow A[n_{pivot}]$;
swap $A[n]$ and $A[n_{pivot}]$;
$i \leftarrow 1$;
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($A[i+1, \cdots, n]$);

□

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.

$1 \prec log(logn) \prec log_4 n \prec logn \prec n^{\frac{1}{2}} \prec n \prec 2^{logn} \prec log(n!) \prec nlogn \prec n^2 \prec 2^n \prec 4^n \prec n! \prec (2n)! \prec 2^{2^n}$

Explain:

(1)

$$2^{logn} = 2^{log_2 n \cdot log2} = n2^{log2}$$

Thus:

$$n \prec 2^{logn} \prec nlogn$$

(2)

$$4^n = 2^{2n} \prec 2^{2^n}$$

(3)

$$\frac{(2n+2)!}{(2n)!} = (2n+2)(2n+1)$$

$$\frac{2^{2^{n+1}}}{2^{2^n}} = 2^{2^n}$$

Thus:

$$(2n)! \prec 2^{2^n}$$

(4)

For n and $log(n!)$, turn them into e^n and $n!$.

Because $e^n \prec n!$, we get $n \prec log(n!)$.

For $nlogn$ and $log(n!)$, turn them into n^n and $n!$.

Because $n! \prec n^n$, we get $log(n!) \prec nlogn$.

For 2^{logn} and $log(n!)$, turn them into $logn$ and $log_2(log(n!)) = log_2 e \cdot log(log(n!))$.

Because $n \prec log(n!)$, we get $2^{logn} \prec log(n!)$.

Finally, we get:

$$2^{logn} \prec log(n!) \prec nlogn$$

$\square$

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.