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:Xin Xu   Student ID:519021910726   Email: xuxin20010203@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	*best*:$\Omega(n \log n)$ *average*:$O(n \log n)$ *worst*:$O(n^2)$	*best*:$O(\log n)$ *average*:$O(\log n)$ *worst*:$O(n)$
CocktailSort	*best*:$\Omega(n)$ *average*:$O(n^2)$ *worst*:$O(n^2)$	$O(n)$

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

for QuickSort:

the best case happens when $A[n]$ can be divided into two same-size array everytime. The worst case happens when $A[n]$ is divided into $A[1]$ and $A[n-1]$, which means the prime $A[n]$ sorted nondecreasingly.

Proof:

The best case: everytime $A[n]$ is divided into two same-size arraies just like a binary tree. The spend time is the sum of all nodes arise from the father node. So, let $n = 2^k - 1$, $T(n) = \sum_{j=0}^{k-1} 2^j \times (2^{k-j} - 1) = O(n \log n)$.

The average case: We suppose that every seperation has equal possibility. So, $T(n) = 2(\frac{T(0)+T(1)+...T(n-1)}{n}) + c \times n$, and $c \times n$ means the time to execute division. Then, $nT(n) = 2(T(0) + T(1) + \ldots + T(n-1)) + c \times n^2$. When $n = n-1$, we have $(n-1)T(n-1) = 2(T(0) + T(1) + \ldots + T(n-2)) + c \times (n-1)^2$. So, taking these equations into consideration, we have $nT(n) - (n-1)T(n-1) = 2T(n-1) + 2c \times n - c$.

Ignoring $-c$, we have $nT(n) = (n+1)T(n-1) + 2c \times n$. So, $\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$. Applying the equation reaptedly, we finally have $\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c\sum_{i=3}^{n+1}\frac{1}{i} = O(\log n)$. So, In average case,$T(n) = O(n \log n)$.
The worst case: everytime $A[n]$ is divided into $A[1]$ and $A[n-1]$ just like a chain, which can be regarded as a degenerated binary tree. So,$T(n) = \sum_{j=1}^{n} = O(n^2)$.

for CocktailSort:

the best case happens when $A[n]$ is sorted nonincreasingly primitively. The worst case happens when $A[n]$ is sorted nondecreasingly primitively.
proof:
The best case: Because $A[n]$ is sorted nonincreasingly, the loop just runs once.
The average case: The total number of loop running is equal, so the possibilities to run the loop once, twice, ..., $n/2$ times are equal. So, $T(n) = 2(\frac{(n-1)+(n-2)}{n} + \frac{(n-1)+(n-2)+(n-3)+(n-4)}{n} + \frac{(n-1)+(n-2)+(n-3)+(n-4)+(n-5)+(n-6)}{n} + \ldots + \frac{(n-1)+(n-2)+(n-3)+(n-4)+\ldots 1}{n}) = O(n^2)$.
The worst case: Because $A[n]$ is sorted nondecreasingly, the loop will run $n/2$ times. So $T(n) = (n-1) + (n-2) + (n-3) + \ldots + 1 = O(n^2)$.

(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.
In step 1, *pivot* $\leftarrow (A[n] + A[1])/2$. In this way, we can avoid the situation where $A[n]$ is divided into $A[n]$ and $A[1]$.
Delete step 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.
$1 \prec \log(\log n) \prec \log n = \log_4 n \prec n^{1/2} \prec 2^{\log n} \prec n \prec \log(n!) \prec n\log n \prec n^2 \prec 2^n \prec 4^n \prec n!$ $\prec (2n)! \prec 2^{2^n}$
proof:
$2^{\log n} = 2^{\frac{\log_2 n}{\log_2 e}} = n^{\frac{1}{\log_2 e}}$. And $1 < \log_2 e < 2$, so $n^{1/2} \prec 2^{\log n} \prec n$.
Since for any infinite number a,$a^n < n!$, so $e^n < n!$, so $n \prec \log(n!)$.
2^{2^n} is in a explosive growth, so it's largest. □

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