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: Junbo Wang   Student ID: 519021910683   Email: sjtuwjb3589635689@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

1 $pivot \leftarrow A[n]$; $i \leftarrow 1$;
2 **for** $j \leftarrow 1$ **to** $n - 1$ **do**
3   **if** $A[j] < pivot$ **then**
4   swap $A[i]$ and $A[j]$;
5         $i \leftarrow i + 1$;
6 swap $A[i]$ and $A[n]$;
7 **if** $i > 1$ **then** QuickSort($A[1, \cdots, i - 1]$);
8 **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

1 $i \leftarrow 1$; $j \leftarrow n$; $sorted \leftarrow false$;
2 **while** ***not*** $sorted$ **do**
3     $sorted \leftarrow true$;
4   **for** $k \leftarrow i$ ***to*** $j - 1$ **do**
5   **if** $A[k] < A[k + 1]$ **then**
6   swap $A[k]$ and $A[k + 1]$;
7             $sorted \leftarrow false$;
8     $j \leftarrow j - 1$;
9   **for** $k \leftarrow j$ ***downto*** $i + 1$ **do**
10   **if** $A[k - 1] < A[k]$ **then**
11   swap $A[k - 1]$ and $A[k]$;
12             $sorted \leftarrow false$;
13     $i \leftarrow i + 1$;

(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	$\Omega(n \log n)$, $O(n \log n)$, $O(n^2)$	$O(\log n)$
CocktailSort	$\Omega(n)$, $O(n^2)$, $O(n^2)$	$O(1)$

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

Solution. For the *QuickSort* algorithm,
(i) the best case occurs when the operation in line 6 swaps the *pivot* to the middle of the array every time it recurs. To solve this, we can simply assume n is the power of 2, $n = 2^k$. Then every time the partition happens, the size of array shrinks but the number of arrays doubles. For the $k + 1 - j$th, there exist 2^{k+1-j} arrays with size of 2^{j-1} each. So the time complexity is

$$T(n) = \sum_{j=1}^{k+1} (2^{k+1-j}) \times 2^{j-1} = n \log n.$$

(ii) The worst case happens when the operation in line 6 swaps the *pivot* to the most right hand of the array every time it recurs, which makes the for loop ahead iterating time equals to $n, n-1, n-2, ..., 1$ and only generate one not the common two recursion every time in the last. Thus the time complexity

$$T(n) \cong 1 + 2 + ... + n = O(n^2).$$

(iii) The average case means during divide in line 6, all the cases can happen, that is $\{0, n-1\}, \{1, n-2\}, \{2, n-3\}$ and so on. And in the average case we assume every division occurs in the same probability so we have

$$T(n) = 2\left(\frac{T(0) + (T(1) + T(2) + ... + T(n-1)}{n}\right) + O(n).$$

That is $nT(n) = 2(T(0) + T(1) + T(2) + ... + T(n-1)) + O(n^2)$, also take $n-1$ as n, $(n-1)T(n-1) = 2(T(0)+T(1)+T(2)+...+T(n-2))+O(n^2)$. And subtract getting

$$nT(n) = (n+1)T(n-1) + O(n),$$

that is

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + O(\frac{1}{n}).$$

Iteration has

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + \sum_{i=3}^{n+1} \frac{1}{i} = O(\log n),$$

so the average time complexity is $O(n \log n)$.

(iv) For the space complexity, in every recursion there are just $O(1)$ local variables needed, but the average recursion will be $\log n$ times and there local variables get accumulated. So the space complexity is $O(\log n)$.

For the $CocktailSort$ algorithm,

(i) the best case occurs when the array is already sorted, then the outer *while* loop will not iterate and inside it the two for loops traverse the whole array to compare and find the element that violates the sequence, giving the time complexity $\Omega(n)$.

(ii) The worst case happens when the array is in reverse order, then it needs $(n-1) + (n-2) + ... + 1 = O(n^2)$ times comparisons and swaps.

(iii) For the average case, we can see that the sorting iteration can vary from 0 to $n-1$ times, and we assume the probability of each one is equal, that is $\frac{1}{n}$. For i times, it need $(n-1) + (n-2) + ... + (n-i) = in - \frac{i(i+1)}{2}$, so the average time complexity is

$$T(n) = \frac{1}{n}\sum_{i=0}^{n-1}(in - \frac{i(i+1)}{2}) = \frac{n(n-1)}{2} - \frac{1}{2}\frac{n-1}{2} - \frac{1}{2}\frac{(n-1)(2n-1)}{6} = O(n^2).$$

(iv) For the space complexity, the algorithm just state $O(1)$ variables, so just $O(1)$. $\square$

(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. We just need to modify the pivot's choice, we can first randomly choose an element as the pivot and exchange it with the $A[n]$ and then let $A[n]$ be the pivot. The modified algorithm with randomization is:

$\square$

Algorithm 3: QuickSort-modified

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

```
k ← RANDOM(1, n); exchange A[k] with A[n];
pivot ← A[n]; i ← 1;
for j ← 1 to n − 1 do
    if A[j] < pivot then
        swap A[i] and A[j];
        i ← i + 1;
swap A[i] and A[n];
if i > 1 then QuickSort(A[1, ··· , i − 1]);
if i < n then QuickSort(A[i + 1, ··· , 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. The answer should be
$1 \prec \log(\log n) \prec \log n = \log_4 n \prec n^{1/2} \prec 2^{\log n} \prec n \prec n \log n = \log(n!) \prec n^2 \prec 2^n \prec 4^n \prec n! \prec (2n)! \prec 2^{2^n}$.
Among them, it is easy to first find
$1 \prec \log(\log n) \prec \log n \prec n^{1/2} \prec n \prec n \log n \prec n^2 \prec 2^n \prec n! \prec 2^{2^n}$.
For $\log_4 n$, we can use $transformthebottomoflagarithmofformular$ getting $\log_4 n = \frac{\log n}{\log 4}$, that is $\log_4 n = \Theta(\log n)$.
For $2^{\log n}$, also we can get $2^{\log n} = 2^{\frac{\log_2 n}{\log_2 e}} = n^{\frac{1}{\log_2 e}}$. Then it is easy to see $n^{\frac{1}{2}} \prec n^{\frac{1}{\log_2 e}} = 2^{\log n} \prec n^1 = n$.
For $\log(n!)$, we go to prove that $\log(n!) = \Theta(n \log n)$. First, $\log(n!) = \sum_{i=1}^{n} \log i < \sum_{i=1}^{n} \log n = n \log n$, giving $\log(n!) = O(n \log n)$. Second, $\log(n!) = \sum_{i=1}^{n} \log i > \sum_{i=\frac{n}{2}}^{n} \log i > \sum_{i=\frac{n}{2}}^{n} \log \frac{n}{2} = \left(\frac{n}{2} + 1\right) \log \frac{n}{2}$, giving $\log(n!) = \Omega(n \log n)$. Therefore, $\log(n!) = \Theta(n \log n)$.
For 4^n, we have $\lim\limits_{n\to\infty} \frac{2^n}{4^n} = \lim\limits_{n\to\infty} \frac{1}{2^n} = 0$, giving $2^n \prec 4^n$. $4^n \prec n!$ is easy to see.
For $(2n)!$, we have $\lim\limits_{n\to\infty} \frac{n!}{(2n)!} = \lim\limits_{n\to\infty} \frac{1}{2n \times (2n-1) \times \ldots \times 1} = 0$, giving $n! \prec (2n)!$. $(2n)! \prec 2^{2^n}$ is easy to see.
So every function have been found the appropriate position. $\square$

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