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: Beichen Yu Student ID: 519030910245 Email: polarisybc@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	best: $O(n \log n)$ worst: $O(n^2)$ average: $O(n \log n)$	best: $O(\log n)$ worst: $O(n)$ average: $O(\log n)$
CocktailSort	best: $O(n)$ worst: $O(n^2)$ average: $O(n^2)$	$O(1)$

(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. (a) First, let us consider the QuickSort algorithm. In fact, this algorithm makes a partition(from line 2 to line 5), and then finish the recursion(from line 7 to line 8). When analysing the complexity of it, we should consider the two steps as well.

The partition is easy, just doing a loop, so the time it spend is just linear. And once we chosen the pivot, we repeat the algorithm on its left part and its right part. If we denote the time we use the Alg. 1 to sort an array of length n by $T(n)$, than we have:

$$T(n) = T(j-1) + T(n-j) + cn$$

Above, j is the index of the pivot, and cn represents the time used for partition. When the array is empty or there is only one item inside, we don't need to sort, so $T(0) = T(1) = 1$.

When the best case happened, the array is always just divided into two arrays of equal length, which means $T(n) = T(n/2) + T(n/2) + cn = 2T(n/2) = cn$. Then we can make a iteration:

$$\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + cn \\
&= 2(T(\frac{n}{4}) + c\frac{n}{2}) + cn \\
&= 2T(\frac{n}{4}) + 2cn \\
&= 4T(\frac{n}{8}) + 4cn \\
&\dots \\
&= kT(1) + kcn
\end{aligned}$$

k is the number of iteration times, and obviously it is equal to $\log n$. So in the best case, the time complexity of the Alg. 1 is $O(n \log n)$.

When the worst case happened, the array always bad divided, which means one part is empty and the other part has all the items in the origin array. That is $T(n) = T(0) + T(n-1) + cn$. Then we make a iteration:

$$\begin{aligned}
T(n) &= T(n-1) + cn + 1 \\
&= T(n-2) + c(n-1) + cn + 2 \\
&\dots \\
&= T(0) + c\frac{n(n-1)}{2} + n \\
&= c\frac{n(n-1)}{2} + n + 1
\end{aligned}$$

Obviously in the worst case the time complexity of the Alg. 1 is $O(n^2)$.

For the average case, we consider the index of the pivot has equal possibilities from 1 to n. Regarding $T(n) = T(j-1) + T(n-j) + cn$, let j from 1 to n, and add these equations together:

$$nT(n) = 2\sum_{i=0}^{n-1} T(i) + cn^2$$

Using misplaced subtraction we get:

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

Do some mathematical processing on this equation:

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

$$\begin{aligned}
\frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + c\frac{2n-1}{n(n+1)} \\
&= T(0) + c\sum_{i=1}^{n} \frac{2i-1}{i(i+1)} \\
&= T(0) + 2c\sum_{i=1}^{n} \frac{1}{i+1} - c\sum_{i=1}^{n} \frac{1}{i(i+1)} \\
&\to T(0) + 2c\log(n+1) - c(1 - \frac{1}{n+1})(n \to \infty)
\end{aligned}$$

So we get:

$$T(n) = 2c(n+1)\log(n+1) + (1-c)n + 1$$

Which means in average the time complexity of the Alg. 1 is $O(n \log n)$.

The space complexity of the algorithm is easier. During each recursion, we do not use additional space, so the space complexity is equal to the number of times we make the recursion. In the best case, each time we divide the array by half, so finally we do the recursion by $\log n$ times, which means that the space complexity is $O(\log n)$. In the worst case, each time we just decrease the length of the array by one item, so finally we do the recursion by n times, which means that the space complexity is $O(n)$. In the average, as the time complexity is $n \log n$ and during each partition the time we need to spend is n, we can conclude that we make the partition for $\log n$ times, which means the space complexity is $O(\log n)$ in average.

The CocktailSort is the improved version of the BubbleSort. In the algorithm, there is no additional space need, and no recursion is used, which means that the space complexity is always $O(1)$.

In the algorithm, each turn we can make sure the smallest and the biggest number are in the right place. Then we can ignore the two number and sort other numbers in the array, until we find the whole array is sorted. In the best case, we find the array is already sorted in the first turn, and it take us only time of $2n$. IN the worst case, we take $n/2$ turns to make sure the array is finally sorted. The time finally we take is:

$$T(n) = 2n + 2(n-2) + \cdots + 2 = n(n+1)/2$$

So in the worst case the time complexity is $O(n^2)$. In the average case, it takes $n/4$ turns, so:

$$T(n) = 2n + 2(n-2) + \cdots + 2(n - 2 \times (n/4 - 1)) = (3n+4)n/8$$

Which means in the average case the time complexity is $O(n^2)$.

(b) Consider the chosen of pivot. In the Alg. 1, we always choose the last item in an array as the pivot. If it is the smallest or the biggest item in the array, then when we make the partition we will get into the worst case.

It is easy to solve the problem. We can choose the first item, the last item and the item in the middle in the array and make a comparison, and set the pivot as the item that neither the biggest one nor the smallest one. Then we can make sure that the worst case will never happen.

Algorithm 3: QuickSort-modified

```
Input: An array $A[1, \cdots, n]$
Output: $A[1, \cdots, n]$ sorted nondecreasingly
$pivot \leftarrow middle(A[n], A[1], A[n/2])$; $i \leftarrow 1$;
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($A[i + 1, \cdots, n]$);
```

$\square$

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 $\ln n$.

$$\begin{array}{ccccc} 1 & n & \log n & \log(\log n) & n \log n \\ \log_4 n & 2^n & 4^n & 2^{\log n} & 2^{2^n} \\ \log(n!) & n! & (2n)! & n^{1/2} & n^2 \end{array}$$

Solution.

$$\begin{array}{cccccccccc} & 1 & \prec & \log(\log n) & \prec & \log n & = & \log_4 n & \prec & n^{1/2} \\ \prec & 2^{\log n} & \prec & n & \prec & n \log n & = & \log(n!) & \prec & n^2 \\ \prec & 2^n & \prec & 4^n & \prec & n! & \prec & (2n)! & \prec & 2^{2^n} \end{array}$$

(a) $2^{\log n}$: Using the bottom change formula,we get that $2^{\ln n} = 2^{\frac{\log_2 n}{\log_2 \mathrm{e}}} = \sqrt[\log_2 \mathrm{e}]{n}$. As $1 < \log_2 \mathrm{e} < 2$, we can know that $n^{1/2} \prec 2^{\log n} \prec n$.

(b) $\log(n!)$: According the Stirling's approximation, we know that $n! \approx \sqrt{2\pi n}(\frac{n}{\mathrm{e}})^n$, so $\ln(n!) \approx \ln(\sqrt{2\pi n}(\frac{n}{\mathrm{e}})^n) = \frac{1}{2}\ln(2\pi n) + n\ln(\frac{n}{\mathrm{e}})$. So $\log(n!) = n \log n$.

(c) 2^{2^n}: $2^{2^n} = (2^n)^{\frac{2^n}{n}}$. When n is big, $2^n > 2n$ and $2^n > 2n^2$, so $(2^n)^{\frac{2^n}{n}} > (2n)^{2n}$. Obviously, we have $(2n)! \prec (2n)^{2n}$, so $(2n)! \prec 2^{2^n}$.

Others are easy to make the comparison. $\square$

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