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: Zirui Liu Student ID:519021910343 Email: L.prime@sjtu.edu.cn

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

```
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	$\Theta(n\log n)$ $\Theta(n\log n)$ $\Theta(n^2)$	$\Theta(\log n)$
CocktailSort	$\Theta(n)$ $\Theta(n^2)$ $\Theta(n^2)$	$\Theta(1)$

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

For Quick Sort Time Analyse:

Best Case:
Among the most balanced divisions PARTITION can do, both sub-questions have the size no-bigger than $n/2$. Since one of the sub-questions has the size of $\lfloor n/2 \rfloor$, and the other one has the size of $\lceil n/2 \rceil - 1$. In this case, quick sort has the following recursive:

$$T(n) \le 2T(n/2) + \Theta(n) \tag{1}$$

So we can conclude that

$$T(n) = O(n\lg n) \tag{2}$$

Worst Case:
The worst situation of quick sort happens when the two parts of the division separately

include $n-1$ elements and 1 element. Since it's the worst situation, we can assume that this unbalanced division happens in every recursive process. The time price for this division is $\Theta(n)$ every time. After we do a recursive process for an array whose size is 0, we can get $T(0)=\Theta(1)$. So the time price for this situation can be presented as:

$$T(n)=T(n-1)+T(0)+\Theta(n)=T(n-1)+\Theta(n) \tag{3}$$

After we added the price of this equation recursively, we can conclude that the time price for quick sort in the worst situation can be $\Theta(n^2)$

For Cocktail Sort Time Analyse:

Best Case:
The best situation of Cocktail sort happens when the original array has already been placed nonincreasingly, in this case, the whole sorting process rakes only $2n-3$ time units,which gives us the time price of $\Theta(n)$.
Worst Case:
When the original array is placed nondecreasingly, we have the worst situation with the time price of $\Theta(n^2)$

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

Solution. We can apply the modification in two different parts. In the first part, we change the way we choose the pivot. Instead of choosing the fixed number or choose randomly, we apply the median-of-three method. We choose the left end ,the right end and the middle three elements to eliminate some worst cases of original arrays, thus reducing 14% of comparison times.
In the second part, we apply the following methods to further improve the performance.
(1) When the length of the sequence to be sorted is split to a certain size, we switch to insertion sort. The reason is that for very small or partly organized arrays, the efficiency of quick sort is no better than insertion sort. The specific length of array is around 5 to 20. We choose the best universal length to be 10, which avoids some harmful degradation scenarios.
(2) At the end of a split, elements equal to the key can be clustered together, so that the next split does not need to split the elements equal to the key again. In the actual process, there will be two steps. During the division process, the elements equal to key are put into the two ends of the array for the first step. The second step is to move the elements equal to key around the pivot after the division is finished. The reason why this method is so effective is that in an array, if there are equal elements, then it can reduce a lot of redundant divisions. This is particularly obvious in repeated arrays with so many same elements.
After we apply the upper methods, the quick sort achieved better time efficiency even in the worst situations. □

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. $1 \prec \log(\log n) \prec \log_4 n = \log n \prec n^{1/2} \prec 2^{\log n} \prec n \prec \log(n!) = n \log n \prec n^2 \prec 2^n$
$\prec 4^n \prec n! \prec (2n)! \prec 2^{2^n}$ □

Proof. The upper formula can be proofed below:
for $1 < \log(\log n)$, we make 2 to be the base number for two times:

$$\lim_{n\to\infty} \frac{4}{n} = 0 \tag{4}$$

for $\log(\log n) < \log_4 n$:

$$\lim_{n\to\infty} \frac{\log(\log n)}{\log_4 n} = \lim_{n\to\infty} \frac{2 * \log_2 n}{n} = 0 \tag{5}$$

for $\log_4 n = \log n$:

$$\lim_{n\to\infty} \frac{\log_4}{\log n} = \frac{1}{2} \tag{6}$$

for $\log n < n^{1/2}$:

$$\lim_{n\to\infty} \frac{\log n}{n^{1/2}} = 0 \tag{7}$$

for $n^{1/2} < 2^{\log n}$:

$$\lim_{n\to\infty} \frac{n^{1/2}}{n} = 0 \tag{8}$$

for $2^{\log n} < n$:

$$\lim_{n\to\infty} \frac{2^{\log n}}{n} = 0 \tag{9}$$

for $n < \log(n!)$:

$$\lim_{n\to\infty} \frac{n}{\log(n!)} = 0 \tag{10}$$

for $\log(n!) = n \log n$:

$$\lim_{n\to\infty} \frac{\log(n!)}{n \log n} = 1 \tag{11}$$

for $n \log n < n^2$:

$$\lim_{n\to\infty} \frac{n \log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = 0 \tag{12}$$

for $n^2 < 2^n$:

$$\lim_{n\to\infty} \frac{n^2}{2^n} = 0 \tag{13}$$

for $2^n < 4^n$:

$$\lim_{n\to\infty} \frac{2^n}{4^n} = \lim_{n\to\infty} \frac{1}{2^n} = 0 \tag{14}$$

for $4^n < n!$:

$$\lim_{n\to\infty} \frac{4^n}{n!} \leq \lim_{n\to\infty} \frac{4^n}{4! * 5 * 6 * ... * n} = 0 \tag{15}$$

for $n! < (2n)!$:

$$\lim_{n\to\infty} \frac{n!}{(2n)!} = \lim_{n\to\infty} \frac{1}{(n+1) * (n+2) * ...(2n)} = 0 \tag{16}$$

for $(2n)! < 2^{2^n}$:

$$\lim_{n\to\infty} \frac{(2n)!}{2^{2^n}} = 0 \tag{17}$$

$\square$

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

These are part of our codes.

1 First appendix

extcolor[rgb]0.98,0.00,0.00**Input c++ source1:** [language=c++]./test.cpp