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: Wendi Chen Student ID: 519021910071 Email: chenwendi-andy@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 ← 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]);
```

Algorithm 2: CocktailSort

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

```
i ← 1; j ← n;sorted ← false;
while not sorted do
    sorted ← true;
    for k ← i to j − 1 do
        if A[k] < A[k + 1] then
            swap A[k] and A[k + 1];
            sorted ← false;
    j ← j − 1;
    for k ← j downto i + 1 do
        if A[k − 1] < A[k] then
            swap A[k − 1] and A[k];
            sorted ← false;
    i ← 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	$\Omega(n \log n)$	$O(n \log n)$	$O(n^2)$	$\Omega(\log n)$	$O(\log n)$	$O(n)$
CocktailSort	$\Omega(n)$	$O(n^2)$	$O(n^2)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

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

(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) Let us analyze *QuickSort* first.

Time Complexity:
The **best** case happens when every time the pivot separates the array into two equally-sized subarrays. In this situation, *QuickSort* will separate the array approximately $\log n$ times. In the i-th time, there are approximately $2^{\log(n)-i} = \frac{n}{2^i}$ arrays of size 2^i. Thus, the number of comparison in *QuickSort* is at least

$$\sum_{i=1}^{\log n} \frac{n}{2^i} \times 2^i = n \log n \tag{1}$$

The **worst** case happens when every time the pivot separates the array into 1 and $n-1$ sized subarrays. For example, the array is originally ordered or reverse ordered. In this situation, *QuickSort* will separate the array approximately n times. In the i-th time, the number of comparison is $n-i$. Thus, the number of comparison in *QuickSort* is at most

$$\sum_{i=1}^{n} i - 1 = \frac{n(n-1)}{2} \tag{2}$$

For **average** case, define $T(n)$ as the average amount of comparison when sorting n elements through *QuickSort*. If we assume in every separation the final position the pivot is equally likely, we have the recurrence

$$\begin{aligned} T(n) &= n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-i-1)] \\ &= n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} T(i) \end{aligned} \tag{3}$$

In fact, we also know $T(0) = 0$ and $T(1) = 0$. To solve this, we set $P(n) = \frac{T(n)}{n+1}$

$$\begin{aligned} nT(n) - (n-1)T(n-1) &= n(n-1) - (n-1)(n-2) + 2T(n-1) \\ \frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + \frac{2(n-1)}{n(n+1)} \end{aligned} \tag{4}$$

Then, we have $P(0) = P(1) = 0$ and

$$\begin{aligned} P(n) &= P(n-1) + \frac{4}{n+1} - \frac{2}{n} \\ P(n) - \frac{2}{n+1} &= P(n-1) - \frac{2}{n} + \frac{2}{n+1} \end{aligned} \tag{5}$$

Therefore, $P(n)$ satisfies

$$\begin{aligned} P(n) - \frac{2}{n+1} &= \sum_{j=1}^{n} \frac{2}{j+1} - 2 = O(\log n) \\ P(n) &= O(\log n) \end{aligned} \tag{6}$$

Finally, we get

$$T(n) = (n+1)P(n) = O(n \log n) \tag{7}$$

Space Complexity:
The space complexity of *QuickSort* is mainly the use of stack space caused by recursion. In the **best** case described above, the recursion tree is a balanced binary tree whose depth is $\log_2 n$. Thus the space complexity is at least $\Omega(\log n)$. For the **worst** case, the balanced binary tree degenerates into a linked list whose depth is n. Thus the space complexity is at most $O(n)$. For **average** case, we can solve this problem from a probability perspective. Every time we choose a pivot, if the final position of the pivot is located from 25 percent to 75 percent, we call it a **good pivot**. Obviously, the probability of a pivot being a good pivot is $\frac{1}{2}$. A good pivot will separate the array into two parts, both of which has a length no more than $\frac{3n}{4}$. If every time the pivot is a good

pivot, the recursion depth will be about $\log_{\frac{3}{4}} n$. As the expectation of the number of times to get a good pivot is two. The expectation of the recursion depth is

$$2\log_{\frac{3}{4}} n \tag{8}$$

Thus the average space complexity of *QuickSort* is $O(\log n)$.

Then let us analyze *CocktailSort*.

Time Complexity:
The **best** case happens when the array is originally ordered. In this situation, it almost traverses the array twice and exits at once. The total amount of comparison will be

$$n - 1 + n - 2 = 2n - 3 = \Omega(n) \tag{9}$$

The **worse** case happens when the array is reverse ordered. Then in each iteration, it will move the largest or smallest element to the head or tail of the array. Thus the total amount of comparison will be

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2) \tag{10}$$

For **average** case, we define $T(n)$ as the average amount of comparison when sorting n elements through *CocktailSort*. As we already know the upper bound of $T(n)$ is $\frac{n(n-1)}{2}$, what we have to do is to determine the lower bound of $T(n)$. If$A[i] > A[j]$ when $i < j$, we call them a **reversed pair**. The principle of *CocktailSort* is actually reducing the reversed pairs by one through one swap. Although amount of comparison is hard to count, the number of reversed pairs in the array is determined only by the array itself. Set $P(n)$ as the average number of reversed pairs in the array, it naturally follows

$$T(n) \geq P(n) \tag{11}$$

because comparison is the prerequisite of swapping a reversed pair. Thus, we can use $P(n)$ as the lower bound of $T(n)$.
Since for each pair of two element, the probability of being a reversed pair is obviously $\frac{1}{2}$. So the expectation of the number of reversed pair in a n-element array is

$$P(n) = \frac{1}{2}\frac{n(n-1)}{2} = \frac{n(n-1)}{4} \tag{12}$$

Thus, we have

$$T(n) \geq \frac{n(n-1)}{4} \tag{13}$$

which indicates $T(n) = O(n^2)$.

Space Complexity:
Since *CocktailSort* only uses i,j,k as temporary variables, the space complexity is $\Theta(1)$.

(b) The strategy of choosing *pivot* matters. The main cause of the worst case is that the *pivot* is always the largest or the smallest element, which will separates the array into 1 and $n1$ sized subarrays. In order to reduce the probability of this situation, we can

adopt a stochastic-based strategy. We can simply choose a random element as the pivot. The revised code is listed below(Alg. 3).

Algorithm 3: Optimized QuickSort

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

```
1  k ← random(1 to n);
2  swap A[k] and A[n];
3  pivot ← A[n]; i ← 1;
4  for j ← 1 to n − 1 do
5      if A[j] < pivot then
6          swap A[i] and A[j];
7          i ← i + 1;
8  swap A[i] and A[n];
9  if i > 1 then QuickSort(A[1, ··· , i − 1]);
10 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.
We can use the limit of the ratio of two functions to rank these functions. In fact, we have

$1 \prec \log(\log n)$ because

$$\lim_{n\to\infty} \frac{1}{\log(\log n)} = 0 \quad \Rightarrow \quad 1 = o(\log(\log n)) \tag{14}$$

$\log(\log n) \prec \log_4 n$ because

$$\lim_{n\to\infty} \frac{\log(\log n)}{\log_4 n} = \lim_{n\to\infty} \frac{\frac{1}{n \log n}}{\frac{c}{n}} = 0 \quad \Rightarrow \quad \log(\log n) = o(\log_4 n) \tag{15}$$

$\log_4 n = \log n$ because

$$\lim_{n\to\infty} \frac{\log_4 n}{\log n} = c \neq 0 \quad \Rightarrow \quad \log_4 = \Theta(\log n) \tag{16}$$

$\log n \prec n^{\frac{1}{2}}$ because

$$\lim_{n\to\infty} \frac{\log n}{n^{\frac{1}{2}}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2n^{\frac{1}{2}}}} = 0 \quad \Rightarrow \quad \log n = o(n^{\frac{1}{2}}) \tag{17}$$

$n^{\frac{1}{2}} \prec 2^{\log n}$ because

$$\lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{2^{\log n}} = \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{2^{c\log_2 n}} = \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{n^c} \tag{18}$$

where $c = log2 > 0.5$, then

$$\lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{n^c} = 0 \quad \Rightarrow \quad n^{\frac{1}{2}} = o(2^{\log n}) \tag{19}$$

$2^{\log n} \prec n$ because

$$\lim_{n\to\infty} \frac{2^{\log n}}{n} = \lim_{n\to\infty} \frac{2^{c\log_2 n}}{n} = \lim_{n\to\infty} \frac{n^c}{n} \tag{20}$$

where $c = log2 < 1$, then

$$\lim_{n\to\infty} \frac{n^c}{n} = 0 \quad \Rightarrow \quad 2^{\log n} = o(n) \tag{21}$$

$n \prec n\log n$ because

$$\lim_{n\to\infty} \frac{n}{n\log n} = \lim_{n\to\infty} \frac{1}{logn} = 0 \quad \Rightarrow \quad n = o(n\log n) \tag{22}$$

$n\log n = \log(n!)$ because

$$\begin{aligned} \log(n!) &= \sum_{i=2}^{n} \log i \leq n\log n \\ \log(n!) &= \sum_{i=2}^{n} \log i \geq \int_1^n logxdx = n\log n - n + 1 \\ &\Rightarrow n\log n = \Theta(\log(n!)) \end{aligned} \tag{23}$$

$n\log n \prec n^2$ because

$$\lim_{n\to\infty} \frac{n\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = 0 \quad \Rightarrow \quad n\log n = o(n^2) \tag{24}$$

$n^2 \prec 2^n$ because

$$\lim_{n\to\infty} \frac{n^2}{2^n} = \lim_{n\to\infty} \frac{2n}{2^n \log 2} = \lim_{n\to\infty} \frac{2}{2^n(\log 2)^2} = 0 \quad \Rightarrow \quad n^2 = o(2^n) \tag{25}$$

$2^n \prec 4^n$ because

$$\lim_{n\to\infty} \frac{2^n}{4^n} = \lim_{n\to\infty} \frac{1}{2^n} = 0 \quad \Rightarrow \quad 2^n = o(4^n) \tag{26}$$

$4^n \prec n!$ because

$$0 \leq \lim_{n\to\infty} \frac{4^n}{n!} \leq \lim_{n\to\infty} \frac{4^4}{1\times 2\times 3\times 4}(\frac{4}{5})^{n-4} = 0 \quad \Rightarrow \quad 4^n = o(n!) \tag{27}$$

$n! \prec (2n)!$ because

$$\lim_{n\to\infty} \frac{n!}{(2n)!} = \lim_{n\to\infty} \frac{1}{2n \times (2n-1) \times \cdots \times (n+1)} = 0 \quad \Rightarrow \quad n! = o((2n)!) \tag{28}$$

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

$$\lim_{n\to\infty} \frac{(2n)!}{2^{2^n}} = \lim_{n\to\infty} \frac{1\times 2}{2^{2^1}} \frac{3\times 4}{2^{2^1}} \cdots \frac{(2n-3)\times(2n-2)}{2^{2^{n-2}}} \frac{(2n-1)\times(2n)}{2^{2^{n-1}}} \tag{29}$$

when n is large enough

$$\frac{(2n-1)\times(2n)}{2^{2^{n-1}}} < \frac{1}{2}$$

Thus, we have

$$0 \le \lim_{n\to\infty} \frac{1\times 2}{2^{2^1}} \frac{3\times 4}{2^{2^1}} \cdots \frac{(2n-3)\times(2n-2)}{2^{2^{n-2}}} \frac{(2n-1)\times(2n)}{2^{2^{n-1}}} \le \lim_{k\to\infty} C(\frac{1}{2})^k = 0 \tag{30}$$

$$\Rightarrow (2n)! = o(2^{2^n}) \tag{31}$$

The result is listed as below.

$$1 \prec \log(\log n) \prec \log_4 n = \log n \prec n^{\frac{1}{2}} \prec 2^{\log n} \prec n \prec n\log n = \log(n!) \prec n^2 \prec 2^n \prec 4^n \prec n! \prec (2n)! \prec 2^{2^n} \tag{32}$$

□

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