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:Renyang Guan Student ID:519021911058 Email: guanrenyang@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 ← A[n]; i ← 1;
2 for j ← 1 to n − 1 do
3     if A[j] < pivot then
4         swap A[i] and A[j];
5         i ← i + 1;
6 swap A[i] and A[n];
7 if i > 1 then
      QuickSort(A[1, ··· , i − 1]);
8 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

```
1  i ← 1; j ← n;sorted ← false;
2  while not sorted do
3      sorted ← true;
4      for k ← i to j − 1 do
5          if A[k] < A[k + 1] then
6              swap A[k] and A[k + 1];
7              sorted ← false;
8      j ← j − 1;
9      for k ← j downto i + 1 do
10         if A[k − 1] < A[k] then
11             swap A[k − 1] and A[k];
12             sorted ← false;
13     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	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$O(\log n)$
CocktailSort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$

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

Quick Sort:

Best case time complexity:

The best case occurs when every time the partition seperates the sequence with n elements into two subsequences, one of which has $\lfloor \frac{n}{2} \rfloor$ elements and the other has $\lceil \frac{n}{2} \rceil - 1$ elements.

For the sake of convenience, let n to be an even number. When n is an odd number, the equation is similar, however.

Assume that $T(n)$ is the time complexity of quicksorting a sequence with n elements. It is obviously that the time complexity of the ***for*** loop is $\Theta(n)$.

Thus,

$$T(n) = 2T(n/2) + \Theta(n)$$

According to the master theory, the solution of the equation is

$$T(n) = \Theta(n)$$

.

Worst case time complexity: The worst case occurs when every time the partition seperates the sequence with n elements into two subsequences, one of which has 0 elements and the other has $n-1$ elements.

Continue to use the same notation $T(n)$. In this case,

$$T(n) = T(0) + T(n+1) + \Theta(n)$$

We could get the solution of the equation by substitute method:

$$T(n) = \Theta(n^2)$$

Average case time complexity: Assume that probabilities of all of the permutations are equal.

Suppose that every partition seperates the array into two arrarys with i and $n-i-1$ elements. The time complexity of partition is $\Theta(n)$, so

$$T(n) = \frac{1}{n}\sum_{i=0}^{n-1}[T(i) + T(n-i-1) + \Theta(n)]$$

.

The solution of the equation is $T(n) = \Theta(n \log n)$.

Proof.

Assume that $\Theta(n) = cn$ when $n > n_0$, where c is a positive number and n_0 is a positive integer.

$$\begin{aligned}
(n+1)T(n+1) &= \sum_{i=0}^{n}[T(i) + T(n-i) + c(n+1)] && (1)\\
&= \sum_{i=0}^{n-1}[T(i) + T(n-i-1) + c(n)] + 2T(n) + c(2n+1) && (2)\\
&= (n+2)T(n) + c(2n+1) && (3)\\
& && (4)
\end{aligned}$$

$$\begin{aligned}
&\Rightarrow \frac{T(n+1)}{n+2} - \frac{T(n)}{n+1} = \frac{c(2n+1)}{n^2+3n+2} \leq \frac{1}{n+1}\\
&\Rightarrow \frac{T(n+1)}{n+2} \leq \sum_{i=1}^{n+1}\frac{1}{i} = \Theta(\log n)T(n+1) = O(n \log n)
\end{aligned} \qquad (5)$$

Since the average time complexity can not exceed the restriction of best case time complexity which is $\Theta(n)$, thus

$$T(n+1) = \Theta(n \log n)$$

□

Space complexity:
The space complexity of every lawyer of the recursion tree is $O(1)$. The progress of recursion will cost stack space which equals to $O(\log n)$.
Thus, the space complexity of QuickSort is $O(\log n)$.

Cocktail Sort:

Best case time complexity: The best case occurs when the sequence has been ***non-increasingly sorted***. The ***while*** loop is only excuted once and the two ***for*** loop is excuted $2n-3$ $((n-1)+(n-2))$ times. Thus, the **best case** time complexity is $\Theta(n)$.

Worst case time conplexity: The worst case occurs when the sequence has been ***nondecreasing sorted***.
The worst case time is

$$\left\lfloor \frac{n}{2} \right\rfloor \times \sum_{i=1}^{\left\lfloor \frac{n}{2} \right\rfloor - 1} 2 \times (n-1-2i)$$

For the sake of convenivence, assume that $n = 2k$, where k is any positive integer, such that $\left\lfloor \frac{n}{2} \right\rfloor = k$. Then the worst case time is

$$\frac{n^2}{2} - 2n + 2$$

.
Thus, the **worst case** time complexity is $\Theta(n^2)$.

Average case time complexity: Two assumptions:

i. $A[1, \cdots, n]$contains the numbers 1 through n.
ii. All $n!$ permutations are equally likely.

Suppose $A[i]$ should be swapped at position $j(1 \leq j \leq n)$.

When $1 \leq j \leq i$, we need $i - j$ comparisons to put $A[i]$ at position j. Since and integer in $[1, \cdots, i]$ is equally likely to be taken by j,i.e.,

$$P(j = 1) = P(j = 2) = \cdots = p(j = i) = \frac{1}{i}$$

When $i < j \leq n$, we need $j - i$ comparisons to put $A[i]$ at position j. Since and integer in $[i+1, \cdots, n]$ is equally likely to be taken by j,i.e.,

$$P(j = n+1) = P(j = n+2) = \cdots = p(j = n) = \frac{1}{n-i}$$

The expectation number of comparisons for put element $A[i]$ in its proper position, is

$$\sum_{j=1}^{i} \frac{i-j}{i} + \sum_{j=i+1}^{n} \frac{j-i}{n-i} = \frac{n}{2}$$

The average number of comparisons performed by Alforithm CocktailSort is

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

Thus, the **average case** complexity is $\Theta(n^2)$

Space complexity:

All the operations related to array A are completed in the memory of array A, so no extra space is needed. Thus, the space complexity of CocktailSort is $O(1)$.

(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.
Considering the best case, an intuitive optimization is to try to seperate the unsorted sequence into to two subsequence nearly equally. The key point is to select an appropriate *pivot* when the time complexity of partition is $O(nlogn)$

Optimization method:

(a) Using the principle of *CountingSort*, store the number of times the integer i appears in the unsorted sequence, where i is any number between 1 to the maximum number of the unsorted sequence.

(b) Use a *for* loop tranversing the array B which stores the counting result and find the ***median number*** to be the *pivot*.

Explanation of the pseudocode:
Line$1-4$ is the progress of finding the maximum of the unsorted array A.

Line$5-7$ is to count the total times of integer i appearing in the arry A, which is stored in $B[i]$.

Line$8-13$ is to find the *pivot* which seperate the array A equally.

Line$14-21$ is the common step of *QuickSort*.

Time complexity analysis:

The time complexity of the four *for* loop is $\Theta(n)$, so we could easily get this equation:

$$T(n) = 2T(n/2) + \Theta(n)$$

and the solution is

$$T(n) = \Theta(n \log n)$$

Algorithm 3: QuickSort-Optimized for the worst case

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

```
1  max ← 0;
2  for k ← 1 to n do
3  |   if A[k] > max then
4  |   |   max ← A[k];
5  Initialize an array B[1, ··· , max];
6  for t ← 1 to n do
7  |   B[A[t]] ← B[A[t]] + 1;
8  sum ← 0;
9  for l ← 1 to max do
10 |   sum ← sum + B[l];
11 |   if sum > n/2 then
12 |   |   pivot = B[l];
13 |   |   Break;
14 i ← 1;
15 for j ← 1 to n − 1 do
16 |   if A[j] < pivot then
17 |   |   swap A[i] and A[j];
18 |   |   i ← i + 1;
19 swap A[i] and A[n];
20 if i > 1 then QuickSort(A[1, ··· , i − 1]);
21 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 Ranking of functions is as follows:

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

Proof. I will prove the above relationship in order from left to right
$1 < \log(\log n))$*:*

$$\lim_{n \to \infty} \frac{1}{n} = 0$$

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

$$\lim_{n\to\infty} \frac{\log(\log n))}{\log_4 n} = \lim_{n\to\infty} \log 4 \times \frac{\log n}{n} = 0$$

$\log_4 n = \log n$:

$$\lim_{n\to\infty} \frac{\log_4 n}{\log n} = \frac{1}{\log 4}$$

$\log n \prec \sqrt{n}$:

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = 0$$

$\sqrt{n} \prec 2^{\log n}$:

$$\lim_{n\to\infty} \frac{\sqrt{n}}{2^{\log n}} = \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{n^{\log 2}} = \lim_{n\to\infty} \frac{1}{n\log 2 - \frac{1}{2}} = 0$$

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

$$\lim_{n\to\infty} \frac{2^{\log n}}{n} = \lim_{n\to\infty} \frac{1}{n^{1-\log 2}} = 0$$

$n \prec \log n!$: Considering $n! = \omega(a^n)$(I will prove it later), where a is a constant integer and $a > 1$

$$0 \leq \lim_{n\to\infty} \frac{n}{\log n!} \leq \lim_{n\to\infty} \frac{1}{\log \exp(n)} = \lim_{n\to\infty} \frac{1}{n} = 0 \Rightarrow \lim_{n\to\infty} \frac{n}{\log n!} = 0$$

$\log n! = n \log n$:

$$\lim_{n\to\infty} \frac{\log n!}{n \log n} = 1$$

$n \log n \prec n^2$:

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

$n^2 \prec 2^n$:

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

$2^n \prec 4^n$:

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

$4^n \prec n!$:
In this part, I will prove that for any positive integer a, where $a > 1$, such that $\lim_{n\to\infty} \frac{a^n}{n!} = 0$

$$0 \leq \lim_{n\to\infty} \frac{a^n}{n!} = \lim_{n\to\infty} \frac{a^n}{a! \times (a+1) \times \cdots \times n} \leq \lim_{n\to\infty} \frac{a^2}{a!} \times \frac{a}{n} = 0$$

$n! \leq (2n)!$:

$$\lim_{n\to\infty} \frac{n!}{(2n)!} = \lim_{n\to\infty} \frac{1}{(n+1) \times (n+2) \times \cdots \times 2n} = 0$$

$(2n)! \leq 2^{2^n}$:
Let $a_n = \frac{(2n)!}{2^{2^n}}$,

$$\frac{a_{n+1}}{a_n} = \frac{(2n+1)(2n+2)}{2^{2^n}} \leq \frac{1}{2}$$

Thus, $a_n < (\frac{1}{2})^n$ and

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

□

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