Lab01-AlgorithmAnalysis

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

∗ If there is any problem, please contact TA Shuodian Yu.
∗ Name: Hongjie Fang Student ID: 518030910150 Email: galaxies@sjtu.edu.cn

1. Please analyze the time complexity of Alg. 1 with brief explanations.

Algorithm 1: PSUM

Input: $n = k^2$, k is a positive integer.
Output: $\sum_{i=1}^{j} i$ for each perfect square j between 1 and n.

```
1 k ← √n;
2 for j ← 1 to k do
3 |   sum[j] ← 0;
4 |   for i ← 1 to j^2 do
5 |   |   sum[j] ← sum[j] + i;
6 return sum[1 ··· k];
```

Solution. Assume that $k = \sqrt{n}$. The outer **for** is executed k times, and the inner **for** is executed $1^2, 2^2, ..., k^2$ times. Thus, total iteration times $T(n)$ can be calculated as follows.

$$T(n) = \sum_{j=1}^{k} \sum_{i=1}^{j^2} 1 = \sum_{j=1}^{\sqrt{n}} j^2 = \frac{1}{6}\sqrt{n}(\sqrt{n}+1)(2\sqrt{n}+1) = \Theta(n^{1.5})$$

As a result, the time complexity of Alg. 1 is $\Theta(n^{1.5})$. □

2. Analyze the **average** time complexity of QuickSort in Alg. 2.

Algorithm 2: QuickSort

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

```
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])
```

Solution. We define $T(n)$ as the **average** number of comparisons of sorting n elements $A[1, \ldots, n]$ with the QuickSort Algorithm (Alg. 2). Obviously we have $T(0) = 0$, since we don't have to sort at all. Without loss of generality, we make the following assumptions:

- $A[1, \ldots, n]$ contains the numbers 1 through n.
- All $n!$ permutations are equally likely.

With the assumptions above, we know that the the probabilities of all the elements to become the pivot (which is, to appear at place $A[n]$) are the same, which is,

$$Pr(x \text{ become pivot}) = \frac{1}{n} \quad (1 \leq x \leq n) \tag{1}$$

When x is chosen to be a pivot, we use a **for** loop, which will make $(n-1)$ comparisons, to divide the elements into two groups: numbers which are smaller than x and numbers which are greater than x. The two groups have $(x-1)$ and $(n-x)$ elements respectively. Then we do the QuickSort Algorithm (Alg. 2) to the two groups recursively, which means the average additional comparison times of $T(x-1)$ and $T(n-x)$. Hence,

$$\begin{aligned} T(n) &= \sum_{x=1}^{n} Pr(x \text{ become pivot}) \cdot ((n-1) + T(x-1) + T(n-x)) \\ &= (n-1) + \frac{1}{n} \sum_{x=1}^{n} (T(x-1) + T(n-x)) \\ &= (n-1) + \frac{2}{n} \sum_{x=0}^{n-1} T(x) \qquad (n \geq 1) \end{aligned} \tag{2}$$

Let $S(n)$ be the prefix summation of sequence $\{T(0), T(1), \ldots, T(n)\}$, which is,

$$S(n) = \sum_{x=0}^{n} T(x) \qquad (n \geq 0)$$

Combine Equation (2) and the definition of $S(n)$ above together we can get:

$$S(n) = \frac{n+1}{2} \cdot (T(n+1) - n) \qquad (n \geq 0)$$

Thus,

$$\begin{aligned} T(n) &= S(n) - S(n-1) \\ &= \frac{n+1}{2} \cdot (T(n+1) - n) - \frac{n}{2} \cdot (T(n) - (n-1)) \qquad (n \geq 1) \\ \implies \quad T(n+1) &= \frac{n+2}{n+1} \cdot T(n) + \frac{2n}{n+1} \qquad (n \geq 1) \end{aligned} \tag{3}$$

Solve the recursive equation (3) we can get the general formula of $T(n)$ as follows.

$$T(n) = \begin{cases} 0 & (n = 0, 1) \\ \dfrac{2(n-1)}{n} + 2(n+1) \displaystyle\sum_{i=1}^{n-1} \frac{i-1}{i(i+1)} & (n \geq 2) \end{cases} \tag{4}$$

With Equation (4) we can obtain the following equation.

$$(n+1)H_{n-1} \leq T(n) \leq 2 + 2(n+1)H_{n-1} \qquad (n \geq 2) \tag{5}$$

where H_n is the n-th harmonic number and $H_n \sim \log n$.

Therefore, the **average** time complexity of QuickSort Algorithm (Alg. 2) is $T(n) = \Theta(n \log n)$. □

3. The BubbleSort mentioned in class can be improved by stopping in time if there are no swaps during an iteration. An indicator is used thereby to check whether the array is already sorted. Analyze the **average** and **best** time complexity of the improved BubbleSort in Alg. 3.

Algorithm 3: BubbleSort

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

```
1 i ← 1;sorted ← false;
2 while i ≤ n − 1 and not sorted do
3     sorted ← true;
4     for j ← n downto i + 1 do
5         if A[j] < A[j − 1] then
6             interchange A[j] and A[j − 1];
7             sorted ← false;
8     i ← i + 1;
```

Solution. The **best** case of the improved BubbleSort Algorithm is that the initial array $A[1, \ldots, n]$ is already sorted non-decreasingly. Under this circumstance, after a inner **for** loop the algorithm come to the conclusion that the array is already sorted, and then set flag *sorted* to *true* to complete the sorting process. Thus, total comparison time is $n - 1 = \Omega(n)$. Thus the **best** time complexity is $\Omega(n)$.

The **worst** case of the improved BubbleSort Algorithm is that the initial array $A[1, \ldots, n]$ is in reverse order. In this situation, the other **while** loop will be executed $(n - 1)$ times and the comparison times is $\sum_{i=1}^{n-1}(n - i) = \frac{n(n-1)}{2}$. Thus the **worst** time complexity is $O(n^2)$.

Then we discuss the **average** case of the improved BubbleSort. Without loss of generality, we make the following assumptions.

- $A[1, \ldots, n]$ contains the numbers 1 through n.
- All $n!$ permutations are equally likely.

We also define some notations and names as follows.

- "An inversion": a pair of elements $(A[i], A[j])$ satisfying $A[i] > A[j]$ while $1 \leq i < j \leq n$.
- $T(n)$: the **average number of comparisons** of sorting n elements $A[1, \ldots, n]$ with the method of improved BubbleSort Algorithm (Alg. 3);
- $S(n)$: the **average number of interchanges** of sorting n elements $A[1, \ldots, n]$ with the method of improved BubbleSort Algorithm (Alg. 3);

Then we give the following useful lemmas.

> **Lemma 1.** *The number of interchanges in BubbleSort Algorithm (Alg. 3) equals to the number of inversions in the initial sequence* $A[1, \ldots, n]$.
>
> **Proof.** According to the algorithm, we only perform an interchange when we find a pair of adjacent elements which is also an inversion. After interchange, we only reduce the number of inversions by one (the pair $(A[j - 1], A[j])$). Therefore, the number of interchanges equals to the number of inversions in the initial sequence. □

Lemma 2. *In BubbleSort Algorithm (Alg. 3), the average number of interchanges can't exceed the average number of comparisons, that is,* $S(n) \leq T(n)$.

Proof. Consider every situation of the sequence $A[1, \ldots, n]$, an interchange only happens after a comparison in BubbleSort Algorithm (Alg. 3). So in every situation, the number of interchanges can not exceed the number of comparisons. Therefore, in average situation, the average number of interchanges can't exceed the average number of comparisons, that is, $S(n) \leq T(n)$. □

Lemma 3. *The average number of inversions in the initial sequence* $A[1, \ldots, n]$ *is* $\frac{n(n-1)}{4}$.

Proof. With the assumptions above, $A[1, ..., n]$ is a permutation of 1 to n. Consider each pair of elements (i, j) satisfying $1 \leq j < i \leq n$. Only when i appears before j, the ordered-pair (i, j) will become an inversion. The probability that i appears before j is equal to the probability that j appears before i, hence they're both 50%. So every pair of elements (i, j) satisfying the conditions above has 50% chance to become an inversion. That means the average number of inversions in the initial sequence s can be counted as follows:

$$s = \sum_{j=1}^{n-1} \sum_{i=j+1}^{n} \frac{1}{2} = \frac{n(n-1)}{4} \tag{6}$$

With Equation (6), obviously the lemma is correct. □

On one hand, With Lemma 1, Lemma 2 and Lemma 3, we know that $\frac{n(n-1)}{4} \leq T(n)$. On the other hand, the average number of comparisons can not exceed the number of comparisons in the **worst** case, which means $T(n) \leq \frac{n(n-1)}{2}$. Thus,

$$\frac{n(n-1)}{4} \leq T(n) \leq \frac{n(n-1)}{2} \tag{7}$$

Equation (7) leads us to the answer: the **average** time complexity of the improved BubbleSort Algorithm (Alg. 3) is $T(n) = \Theta(n^2)$.

Summary: the time complexity of improved BubbleSort Algorithm (Alg. 3).

- **Best case**: $\Omega(n)$;
- **Average case**: $\Theta(n^2)$;
- **Worst case**: $O(n^2)$.

□

4. 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 $\log_2 n$.

$2^{\log n}$	$(\log n)^{\log n}$	n^2	$n!$	$(n+1)!$
2^n	n^3	$\log^2 n$	e^n	2^{2^n}
$\log \log n$	$n \cdot 2^n$	n	$\log n$	$4^{\log n}$

Solution. We can draw the following conclusions.

- $\log\log n \prec \log n$, since

$$\begin{aligned} & \log n < n && (n \geq 2, n \in \mathbb{N}) \\ \implies \quad & \log\log n < \log n && (n \geq 2, n \in \mathbb{N}) \\ \implies \quad & \log\log n = o(\log n) \end{aligned}$$

- $\log n \prec \log^2 n$, since

$$\lim_{n\to+\infty} \frac{\log n}{\log^2 n} = \lim_{n\to+\infty} \frac{1}{\log n} = 0 \quad \implies \quad \log n = o(\log^2 n)$$

- $\log^2 n \prec n$, since

$$\lim_{n\to+\infty} \frac{\log^2 n}{n} = 0 \quad \implies \quad \log^2 n = o(n)$$

- $n = 2^{\log n}$, since

$$2^{\log n} = n \quad \implies \quad n = \Theta(2^{\log n})$$

- $n \prec n^2$, since

$$\lim_{n\to+\infty} \frac{n}{n^2} = \lim_{n\to+\infty} \frac{1}{n} = 0 \quad \implies \quad n = o(n^2)$$

- $n^2 = 4^{\log n}$, since

$$4^{\log n} = 2^{2\log n} = 2^{\log(n^2)} = n^2 \quad \implies \quad n^2 = \Theta(4^{\log n})$$

- $n^2 \prec n^3$, since

$$\lim_{n\to+\infty} \frac{n^2}{n^3} = \lim_{n\to+\infty} \frac{1}{n} = 0 \quad \implies \quad n^2 = o(n^3)$$

- $n^3 \prec (\log n)^{\log n}$, since

$$\begin{aligned} & \log(n^3) = 3\log n < \log\log n \cdot \log n = \log\left((\log n)^{\log n}\right) && (n > 256, n \in \mathbb{N}) \\ \implies \quad & n^3 < (\log n)^{\log n} && (n > 256, n \in \mathbb{N}) \\ \implies \quad & n^3 = o((\log n)^{\log n}) \end{aligned}$$

- $(\log n)^{\log n} \prec 2^n$, since

$$\begin{aligned} & \log\left((\log n)^{\log n}\right) = \log\log n \cdot \log n < n = \log(2^n) && (n > 2, n \in \mathbb{N}) \\ \implies \quad & (\log n)^{\log n} < 2^n && (n > 2, n \in \mathbb{N}) \\ \implies \quad & (\log n)^{\log n} = o(2^n) \end{aligned}$$

- $2^n \prec n \cdot 2^n$, since

$$\lim_{n\to+\infty} \frac{2^n}{n \cdot 2^n} = \lim_{n\to+\infty} \frac{1}{n} = 0 \quad \implies \quad 2^n = o(n \cdot 2^n)$$

- $n \cdot 2^n \prec e^n$, since

$$\lim_{n\to+\infty} \frac{n \cdot 2^n}{e^n} = \lim_{n\to+\infty} n \cdot \left(\frac{2}{e}\right)^n = 0 \quad \implies \quad n \cdot 2^n = o(e^n)$$

- $e^n \prec n!$, since

$$0 < \frac{e^n}{n!} = e \cdot \frac{e}{2} \cdot \prod_{i=3}^{n} \frac{e}{i} < \frac{e^2}{2} \cdot \left(\frac{e}{3}\right)^{n-3} \longrightarrow 0 \quad (n \to +\infty)$$

$$\Longrightarrow \quad \lim_{n\to+\infty} \frac{e^n}{n!} = 0$$

$$\Longrightarrow \quad e^n = o(n!)$$

- $n! \prec (n+1)!$, since

$$\lim_{n\to+\infty} \frac{n!}{(n+1)!} = \lim_{n\to+\infty} \frac{1}{n+1} = 0 \quad \Longrightarrow \quad n! = o((n+1)!)$$

- $(n+1)! \prec 2^{2^n}$, since

$$\log(n+1)! = \sum_{i=1}^{n+1} \log i < \sum_{i=1}^{n+1} i < (n+1)^2 < 2^n = \log 2^{2^n} \quad (n > 10, n \in \mathbb{N})$$

$$\Longrightarrow \quad (n+1)! = o(2^{2^n})$$

As a result, we are able to derive the answer of the question, as formula (8) shows.

$$\begin{array}{ccccccccccc} & \log\log n & \prec & \log n & \prec & \log^2 n & \prec & n & = & 2^{\log n} \\ \prec & n^2 & = & 4^{\log n} & \prec & n^3 & \prec & (\log n)^{\log n} & \prec & 2^n \\ \prec & n \cdot 2^n & \prec & e^n & \prec & n! & \prec & (n+1)! & \prec & 2^{2^n} \end{array} \tag{8}$$

□

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